Washington University in St. Louis

Washington University Open Scholarship

All Computer Science and Engineering Research

Computer Science and Engineering

Report Number: WUCS-93-11

1993-01-01

Logical Interference in Symmetric Connectionist Networks

Gadi Pinkas

This work delineates the relation between logic and symmetric neural networks. The motivation is two-fold: 1) to study the capabilities and limitations of connectionist networks with respect to knowledge representatoin; and 2) to develop a new kind of inference negine that is expressive, massively parallel, capable of coping with nonmonotonic or noisy knowledge and capable of learning. The thesis shows that propositional logic can be implemented efficiently in networks where hidden units allow the representation of arbitrary constraints. An inference engine is constructed which can obtain its knowledge either by compiling symbolic rules or by learning them inductively from... Read complete abstract on page 2.

Follow this and additional works at: https://openscholarship.wustl.edu/cse_research

Part of the Computer Engineering Commons, and the Computer Sciences Commons

Recommended Citation

Pinkas, Gadi, "Logical Interference in Symmetric Connectionist Networks" Report Number: WUCS-93-11 (1993). *All Computer Science and Engineering Research.*
https://openscholarship.wustl.edu/cse_research/298

Department of Computer Science & Engineering - Washington University in St. Louis
Campus Box 1045 - St. Louis, MO - 63130 - ph: (314) 935-6160.

This technical report is available at Washington University Open Scholarship: https://openscholarship.wustl.edu/cse_research/298

Logical Interference in Symmetric Connectionist Networks

Gadi Pinkas

Complete Abstract:

This work delineates the relation between logic and symmetric neural networks. The motivation is two-fold: 1) to study the capabilities and limitations of connectionist networks with respect to knowledge representatoin; and 2) to develop a new kind of inference negine that is expressive, massively parallel, capable of coping with nonmonotonic or noisy knowledge and capable of learning. The thesis shows that propositional logic can be implemented efficiently in networks where hidden units allow the representation of arbitrary constraints. An inference engine is constructed which can obtain its knowledge either by compiling symbolic rules or by learning them inductively from examples. The approach is then extended to support unrestricted first-order logic and nonmonotonic reasoning. Experiments with large-scale, randomly generated satisfiability problems indicate that the method is superior to other existing algorithms (when executed in parallel). Finally, the thesis studies the feasibility of finding sound solutions: a new distributed algorithm is given that efficiently finds a global solution for certain network topologies; negative results are proved for networks with less restricted topologies.

Logical Inference in Symmetric Connectionist Networks

Gadi Pinkas

WUCS-93-11

March 1993

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

A thesis presented to the Sever Institute of Washington University in partial fulfillment of the requirements for the degree of Doctor of Science.

WASHINGTON UNIVERSITY
SEVER INSTITUTE OF TECHNOLOGY

ABSTRACT

LOGICAL INFERENCE IN SYMMETRIC CONNECTIONIST NETWORKS

by Gadi Pinkas

ADVISOR: Professor Ronald P. Loui

December, 1992

Saint Louis, Missouri

This work delineates the relation between logic and symmetric neural networks. The motivation is two-fold: 1) to study the capabilities and limitations of connectionist networks with respect to knowledge representation; and 2) to develop a new kind of inference engine that is expressive, massively parallel, capable of coping with nonmonotonic or noisy knowledge and capable of learning. The thesis shows that propositional logic can be implemented efficiently in networks where hidden units allow the representation of arbitrary constraints. An inference engine is constructed which can obtain its knowledge either by compiling symbolic rules or by learning them inductively from examples. The approach is then extended to support unrestricted first-order logic and nonmonotonic reasoning. Experiments with large-scale, randomly generated satisfiability problems indicate that the method is superior to other existing algorithms (when executed in parallel). Finally, the thesis studies the feasibility of finding sound solutions: a new distributed algorithm is given that *efficiently* finds a global solution for certain network topologies; negative results are proved for networks with less restricted topologies.

TABLE OF CONTENTS

LOGICAL INFERENCE IN SYMMETRIC CONNECTIONIST NETWORKS

1. INTRODUCTION

1.1. Background

Symbolic artificial intelligence (AI) and connectionism both aim towards the ambitious goals of creating intelligent machines and understanding human cognition. The two approaches start, however, from different positions. The traditional symbolic approach argues that algorithmic manipulation of symbol systems is a good framework for modeling cognition [Newell 80], [Fodor 76], [Pylyshyn 84]. Connectionists, on the other hand, restrict themselves to architectures inspired by the brain and argue that this approach has the potential to overcome the rigidity of symbolic systems and to more accurately model cognitive tasks that may only be approximated, if at all, by the symbolic paradigm [Smolensky 88], [Churchland, Sejnowski 89].

Both approaches have their successes and failures. To quote Minsky, "connectionist research has shown considerable promise in mimicking some of what we admire in the behavior of lower animals ... However, their performance has been far weaker in precisely the areas in which symbolic systems have successfully mimicked much of what we admire in high-level human thinking" [Minsky 91]. It may be under debate how much of language processing and other aspects of intelligence is better modeled by symbolic structures; however, it is widely believed that some aspects of cognition such as goal-directed problem solving and logical reasoning require such structures [Chandrasekaran et al. 88], [Fodor, Pylyshyn 88], [Pinker, Prince 88].

It is my belief (shared with many others) that the solution lies somewhere in between the two extremes. The approaches should be integrated and unified, and more research should be done in order to build a suitable bridge between them [Minsky 91], [Hinton 90b]. In order to truly take advantage of the best of the two paradigms, we should find ways which enable fully connectionist networks to have the representational powers of symbolic systems (for more elaborate discussions see [Smolensky 88], [Smolensky 90], [Smolensky 91], [Hinton 90b]). This thesis concentrates on the question of how to extract powerful symbolic representation and reasoning abilities out of a connectionist architecture.

The focus will be on the problem of *symbolic reasoning*. The ability to reason symbolically is central to many of the high-level cognitive processes and problem solving skills that humans possess. It is believed to be fundamental to tasks like goal-directed generation of plans, natural language understanding and explanation. Humans are able to learn, acquire new knowledge, update it, resolve ambiguities and inconsistencies and, most importantly, *use* it for obtaining goals and for inferring conclusions. This thesis provides results on the ability of connectionist systems to reason and presents techniques that allow networks to perform symbolic computation.

1.2. The Problem

The question we face is how to represent and reason with *complex* symbolic knowledge in networks. The set of primitives available for the connectionist is so radically different from what is usually assumed by symbolic algorithms that an answer to this question is far from trivial. It is expected, therefore, that symbolic ability will be implemented in "ways that have not been anticipated within the standard symbol processing tradition" [Hinton 90b]. Such an implementation will potentially contribute to the theory and practice of traditional symbolic processing as well as lead to a better understanding of the capabilities of connectionist networks. The hope is that the connectionist approach will have some added value in terms of speed, robustness, adjustability and flexibility.

Several key problems prevent us from using standard techniques when implementing symbolic computation in connectionist networks: the network's inability to perform concatenations, its weak and static structure, and the lack of procedural control and synchronization. The first problem is the inability of connectionist networks to perform *concatenations*[1] in a natural way.[2] This inability is in contrast to symbolic systems, which use concatenation extensively [Van Gelder 89]. The second problem is the weak structure and the static nature of the network's topology and weights. Weights (long-term memory) are usually not allowed to change during run-time and the only dynamic change that may occur is the pattern of unit activations (short-term memory).[3] The weak structure assumed by many connectionist systems not only limits the ability of a network to deal with complex knowledge and procedures, but also limits the amount of parallelism that can be achieved [Minsky 91]. The third problem is that symbolic systems have a *procedural control mechanism* to process the information expressed in the representation, while connectionist models have no such mechanism. We need, therefore, to find a connectionist representation that is capable of expressing the data as well as the procedural knowledge that is needed for controlling the reasoning process. These three differences enable symbolic systems to handle quite easily tasks that seem very hard for networks. Some examples of such tasks are dynamic creation of complex data structures (e.g., stacks, trees), performing ordered search (e.g., backtracking), dynamic

[1]The use of pointers to represent complex structures is viewed here as a technique to perform concatenation.

[2]A unnatural way is to simulate traditional symbolic techniques (e.g., concatenation) using neurons.

[3]Fast weight updating is possible in certain cases but is not usually assumed.

binding of variables to values or variables to other variables (e.g., unification), and systematically using general rules (e.g., transitive closure).

This thesis attacks the above problems. It presents a general scheme that allows one to implement both declarative and procedural knowledge on a structured, fully connectionist architecture. The solutions proposed allow for variable binding, dynamic creation of complex data structures and systematic usage of rules. However, they do not use concatenations, pointers, recursion, synchronization, stacks, lists, trees or any other of the primitives typically used in traditional sequential processing. Most of the thesis is concerned with obtaining expressive power (declarative and procedural) out of connectionist networks, thus allowing networks to have representational powers similar to some of the most expressive symbolic systems available without paying too much in space complexity. A smaller portion of the thesis complements the expressiveness results by discussing issues such as learning, robustness, efficiency of computation and experimental results.

1.3. The Engineering Prospect

The goal of performing symbolic computation in a connectionist architecture is not just intellectually intriguing and scientifically justified, there is also a lot to gain from an engineering perspective.

In the future, many believe that massively parallel computers will be much more affordable and accessible than they are now. It is, therefore, important to study whether symbolic computation (as opposed to scientific computation, which is currently the main client of SIMD architectures) can utilize such architectures.

In this section I describe a pragmatic vision of a new kind of knowledge-based system, which I conjecture *will* be feasible in the near future (unlike the ambitious long-term goals of AI). The specifications for such a vision do not include all that may be achieved in a unified approach; however, its strengths are in its engineering usefulness and in its conjectured feasibility.

The motivation for this kind of knowledge system is clearly engineering; however, its creation, I believe, will be a major step towards the scientific goal of bridging and unifying the connectionist and symbolic approaches. The engineering vision is used in this thesis as a medium-term desideratum that drives and guides the research to more practical tracks.

A Connectionist Knowledge-based System

The new knowledge system will be based on a connectionist architecture and will have the following properties:

1. **Powerful representation:** We would like the mechanism to be as expressive as first-order predicate logic (FOL). In other words, the engine should have the ability to represent unrestricted FOL rules and facts and reason with them. This ability should be present even if tractability dictates that only a subset of the language is

used. The motivation is that an intelligent agent should have as few constraints as possible on its ability to represent knowledge even if for some problem instances performance degrades drastically. We would like the mechanism to be used as a platform that is expressive and flexible enough so that many useful symbolic systems may be mapped into it. The representation should be *compact*, and the size of the network should be linear in the size of the knowledge-base (KB).

2. **Ability to cope with conflict and inconsistency:** The knowledge-base may be inconsistent as a result of noisy, unreliable and redundant sources of information. We would like the mechanism to produce rational conclusions despite inconsistencies. The mechanism should also be able to cope with ambiguities and conflicts that are generated by common sense domains. It should represent, for example, frame axioms, defaults and exceptions and warrant intuitive, nonmonotonic conclusions from its knowledge-base. We would like the mechanism to provide a flexible framework so that existing and hopefully future nonmonotonic systems may use it as a platform.

3. **Incremental updating:** When the knowledge-base (KB) is locally updated (adding or deleting a single rule for example), the internal representation of our mechanism should also be locally updated. An incremental change to the KB should result in an incremental change to the representation since one does not want to re-compile the entire KB when a minor deletion or addition occurs.

4. **Robustness:** The mechanism should allow graceful degradation in system functionality and performance as noise and hardware faults are introduced.

5. **Learning capabilities:** Some (but not all) of the knowledge should be extracted from examples, rather than be compiled from a symbolic representation. This will allow the system to continuously adjust its knowledge in a changing environment. As new data are gathered and more training is done, the system should become more fine-tuned to its actual environment and task. For example, in the short-term, the system may allow the following: 1) grounding of low level (primitive) predicates in the sensory data, i.e., the conditions under which a predicate (grounded in sensory data) holds should be inductively learned; 2) simple propositional rules should be learned from examples and immediately used in the inference process.

6. **Fast execution:** The massively-parallel architecture should give us a real performance advantage over sequential approaches. Although we try to cope with an inherently intractable problem, average-case performance should be good. To cope with the inherent intractability of the problems, an *any-time* algorithm should be adapted, i.e., the system should give an educated guess about a conclusion when an arbitrary time limit is reached. The system should improve the probability of providing a correct guess as more time resources are given. As an optimization tool, the system should improve its approximation with time. The "any-time" property is valuable for real-time decision making.

The work discussed in this thesis touches on all of the above specifications.[4] It concentrates primarily on the first three properties: expressiveness, conflict and inconsistency resolution, and incremental updating; however, some aspects of robustness, learning and fast execution are discussed as well.

1.4. The Thesis

1.4.1. Guidelines

Two principles guide the research:

1. **Expressiveness:** We would like to push the issue of the representational power of connectionist networks to the extreme. The question that is asked is how expressive (symbolically) can networks be made, regardless of speed and (time) complexity considerations.[5] My goal is to represent and reason with unrestricted FOL, which is one of the more expressive and ubiquitous formal languages known.

2. **Simplicity and Standardization:** We would like to show that *standard* networks can perform symbolic computation. Specialized ad-hoc architectures are not required. To be loyal to this principle means that one should use simple, popular and general purpose connectionist models. *Simplicity* allows us to implement the techniques on a wider range of models and makes the technique more easily realized in hardware. *Popularity* enable us to take advantage of the larger bulk of research done already (e.g., learning algorithms, hardware implementations, etc.), which will not be available if an ad-hoc model is used.

1.4.2. On the Use of Logic

I have chosen logic as a representation language because of its expressive power, compactness, mathematical neatness, elegance, lack of ambiguity, and long research history and because it can be subjected to rigorous analysis (see [Hayes 77], [Moore 82] for much more elaborated discussions). Nevertheless, this work has no particular commitment to logic as a representation language or as an account for what actually is the nature of the mind. Instead, this work is concerned with general schemes of how to represent and reason with complex symbolic structures. Logic is used to *evaluate* the representational power of the networks and to *demonstrate* how an extremely complex symbolic problem (theorem proving for example) could be approached in a connectionist architecture.

[4]The issue of overcoming the rigidity of symbolic AI [Minsky 91], [Smolensky 88] is not explicitly specified in the desideratum above. Nevertheless, properties that allow us to escape the brittleness of symbols are definitely desired and are the natural future directions of this line of research.

[5]Space complexity is still an important consideration in evaluating expressiveness. A powerful representation should be compact in space.

1.4.3. Main Contributions

- For the neural network researcher the thesis offers an elaborate and formal study of the limitations and capabilities of connectionist models with respect to representational powers. It offers an insight into the role of the hidden units and high-order connections, provides constructions that allow the implementation of arbitrary constraints, demonstrates how extremely complex problems can be represented, and explores the feasibility of finding sound algorithms (that search for global solutions).
- For the cognitive scientist the thesis shows how powerful symbolic modeling can be done in connectionist networks and opens the door for explaining phenomena that need integration and unification of the two paradigms. The thesis provides yet another answer to attacks on connectionism such as that of [Fodor, Pylyshyn 88], with the unique contribution of showing extreme expressive power. It demonstrates how to construct connectionist systems that feature productivity, compositionality and systematicity similar (in representational power) to some of the most powerful symbolic systems, yet, in a very different way.
- For automated reasoning and knowledge representation the thesis provides a new look at automatic deduction and gives a scheme for how to use a massively parallel architecture as an inference engine. It suggests a novel approach for representing and building knowledge-based systems and for performing monotonic and non-monotonic inferences.
- For the parallel algorithms researcher the thesis provides a general methodology for mapping complex, discrete problems into massively parallel architectures and discusses the feasibility of a connectionist algorithms for finding global solution. It provides a new and improved distributed, self-stabilizing algorithm that guarantees global solutions for certain network topologies and proves that no such algorithm exists for less restricted networks.
- For those who are interested in constraint satisfaction and/or optimization, the thesis provides a new motivation and new techniques for mapping problems into constraint optimization. The work provides experimental evidence that the proposed connectionist techniques may be used for solving *hard*, large-scale, satisfiability problems with a surprisingly high speed.
- For the general computer scientist, the thesis provides a radically different technique for performing symbolic computation. The experimental results of this thesis indicate that greedy algorithms inspired from connectionist networks may well be the best so far (on average) for solving certain hard classical problems.

1.4.4. Related Work

Many researchers have been intrigued by the problem of performing *high-level* cognitive tasks on a connectionist architecture. For examples see: [McClelland, Kawamoto 86]

[Touretzky, Geva 87], [Sumida et al. 88], [Dolan 89], [Lange, Dyer 89], [Miikkulainen, Dyer 89], [Chalmers 90], [Dyer 90], [Hinton 90a], [Kwasny, Faisal 90], [Pollack 90], [Touretzky 90], [Sun 91], [Blank et al. 92].

Fewer attempts have been made to perform high-level *reasoning* with *strong* knowledge level (symbolic) theory. Shastri's work on **evidential reasoning** [Shastri 88] encodes inheritance networks in a localist architecture of spreading activation. Shastri uses the maximal entropy principle to combine evidence and to compute the probabilities of the concepts in the network. **DCPS** [Touretzky, Hinton 88] uses a coarse-coded distributed representation to perform rule matching and firing in a production system based on a Boltzmann machine. Dolan and Smolensky later improved DCPS by using **tensor products** [Dolan, Smolensky 89]. **Conposit** [Barnden 91] is another rule-based system that is less restricted than DCPS. It is based on a localist architecture and takes an approach very similar to the one taken by traditional symbolic techniques. Barnden later extended the approach of Conposit to handle case-based reasoning [Barnden, Srinivas 91]. In **μ-Klone**, Derthick presents a system which is a restricted version of the frame-based system KL-ONE [Derthick 88]. **Frameville** is another frame-based system [Mjolsness et al. 89] that uses optimization and graph matching to reason about hierarchies of frames.

Even fewer attempts have been made to directly implement *logical* reasoning in networks. The first were Ballard and Hayes who designed a restricted resolution-based theorem proving using the energy paradigm [Ballard 86]. Shastri and Ajjanagadde present a very efficient system that allows goal-directed deduction from restricted forms of FOL rules and facts [Shastri, Ajjanagadde 90]. Finally, CHCL [Hölldobler 90] presents an inference system for Horn-logic that uses Bibel's connection method.

Common to all the systems above is that they are able to implement only a *restricted* subset of FOL. Yet, they provide us with insights on the problems that need to be solved and on the possible techniques that may be used for solving them. Further discussions on these systems and others are scattered among the pages of this work.

1.4.5. A Note for Theoreticians

Many of the problems discussed in this thesis are NP-complete. As such, complexity theory anticipates the existence of polynomial time reductions between any pair of them. When such reductions are discussed in this work, the emphasis[6] will be on the technique of performing the mapping, the linearity of space and time, the efficiency of the local heuristics, the robustness of the representation and the fine grain parallelism that is obtained. In general, the contribution is in understanding the theory and the techniques that allow us to represent well-known, difficult problems in neural networks (and not in providing new theory regarding the problems themselves).

[6]The feasibility of these reductions is not surprising.

1.5. Outline of the Thesis

Chapter 2 gives an introduction to symmetric connectionist models, which are the computational models used in this work.

In Chapter 3 high-order networks are introduced, and the first expressiveness result about the equivalence of high-order and low-order symmetric models is proved.

Chapter 4 shows how to represent and learn arbitrary propositional constraints in networks and how to perform model-theoretic inference with unrestricted propositional logic. Experimental results are given for large-scale, randomly generated satisfiability problems.

Chapter 5 discusses syntactic inferences and shows how a theorem-prover for first-order logic can be constructed. The variable-binding problem is presented and a solution is given, which handles general and dynamic unification of predicates, variables and functions. The chapter discusses the robustness of the solution and compares the solution to some recent connectionist systems for high-level symbolic reasoning.

Chapter 6 extends propositional logic and presents a nonmonotonic system that is implemented very naturally in networks. A model theory and a proof theory are given, and tight relationships to networks are identified.

Chapter 7 discusses the feasibility of finding massively parallel (connectionist style) algorithms that are guaranteed to find *global* solutions, thus performing *sound* symbolic reasoning. A positive result shows that an efficient algorithm exists for certain network topologies and leads to a new and improved activation function designed for optimization. Negative results show that there is no hope for guaranteeing a global solution for networks under weaker assumptions.

Chapter 8 concludes and discusses future research.

2. SYMMETRIC MODELS

In this chapter I present the computational model used throughout the thesis and give a brief tutorial of the family of models called symmetric models (SM).

2.1. Why Symmetric?

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

1. Symmetric networks can be characterized by energy functions, which make it easier to specify the desired behavior of the network in a declarative form [Feldman 85].
2. Symmetric networks are simple and popular. Vast research literature (about them) exists and both silicon and optical implementations have been developed [Alspector et al. 88], [Farhat et al. 85].
3. Symmetric networks were used successfully[1] to express and approximate "hard" problems [Hopfield, Tank 85]. Their ability to approximate "NP-hard" problems, such as the traveling salesperson problem, and their "any-time" algorithm (see previous chapter) make them promising candidates for serving *our* "hard" problems.[2]
4. Symmetric networks are expressive enough to represent the behavior of any non-oscillating recurrent network that is based on binary threshold units [Pinkas 91e]. They are, therefore, quite powerful, and we will not lose expressive power if we restrict ourselves to the symmetric case.[3]

2.2. Networks for Optimization

The brain seems to solve perceptual and cognitive tasks very quickly. Apparently, it does so by using parallel, analog hardware without synchronization. Many of these tasks are "hard" from a computational point of view, yet need to be solved quickly. A reasonable

[1]Disappointing results also have been reported. It seems that success is very much dependent on the mapping between the problem and the network.

[2]Reasoning is NP-hard in the propositional case and undecidable for FOL.

[3]Sometimes an asymmetric network will perform better than the symmetric form [Pinkas 91e]; therefore, for efficiency, we may consider not restricting ourselves to the symmetric case.

conjecture is that humans do *approximation* for many of these problems. This thesis explores the possibility of representing and approximating such hard cognitive tasks using parallel, analog architectures inspired by the brain.

A circuit that generates desired solutions can be designed to approximate a constrained optimization problem. The idea is that certain circuits may be viewed as performing optimization; therefore, the design method proposed is to map the problem into a constrained optimization problem and then implement it using parallel hardware [Hopfield, Tank 85], [Platt 89]. The objective function needs to be minimized while the state that is found needs to fulfill some desirable properties of the solution.

Symmetric models (SMs) may be viewed as performing a search for a minimum of some energy function that is associated with the network's topology and weights [Hopfield 82]. The energy paradigm was suggested as a framework for the analysis and specification of connectionist networks [Feldman 85]. The thrust of this paradigm has been that it allows the specification of behavior of a network to be decomposed into two distinct aspects: 1) specification of the desired states (for each input) as global minima, and 2) characterization of algorithms which will drive the network into a state of deep minimum (given an input). The two tasks can be dealt with separately: the first is concerned with the declarative content of the model, and the second is concerned with its dynamics.

In this thesis, I concentrate mainly on the declarative aspect.[4] The main questions to be studied relate to the kind of knowledge that can be expressed in (symmetric) networks and to the ways this knowledge can be used for reasoning.

2.3. Quadratic Energy Functions

I discuss a special family of objective functions that have a natural connectionist representation. Quadratic energy functions are multi-linear polynomials with the following general form:

$$E(\vec{X}) = -\sum_{1\leq i<j\leq n} w_{ij}X_iX_j - \sum_{1\leq i\leq n} w_iX_i.$$

The variables of an energy function can be divided into two sets:

1. Visible variables are usually of interest to an observer. An instantiation of these variables by the system is considered to be an answer (output) to the problem. In certain problems, the user may wish to clamp (instantiate) some of the visible variables (inputs). After clamping is done, the task is to optimize the new function that is obtained by replacing the clamped variables with the clamped values.

2. Hidden variables are usually not of interest to an external observer. Their task, as we'll see later, is merely to add expressive power.

[4] Chapter 7 is concerned with the dynamics.

Definition 2.3.1 The set of the minimizing vectors of the energy function, projected onto the visible variables, is called the set of the *visible solutions* of the minimization problem E with $\vec{X}$ visible variables and $\vec{T}$ hidden variables, i.e., $\Gamma_E = \{\vec{x} \mid (\exists \vec{t})E(\vec{x},\vec{t}) = min_{\vec{y},\vec{u}}\{E(\vec{y},\vec{u})\}\}$.

2.4. What are Symmetric Networks?

A symmetric network (SN) is characterized by a weighted undirected graph whose nodes represent processing units and whose arcs represent weighted connections (see Figure 2.1). There are two kinds of arcs. Pairwise arcs link two nodes and monadic arcs are each attached to a single node. A pairwise arc represents a weighted connection ($w_{i,j}$), while a monadic arc represents a bias (θ_i) given to a single unit. The weights of the

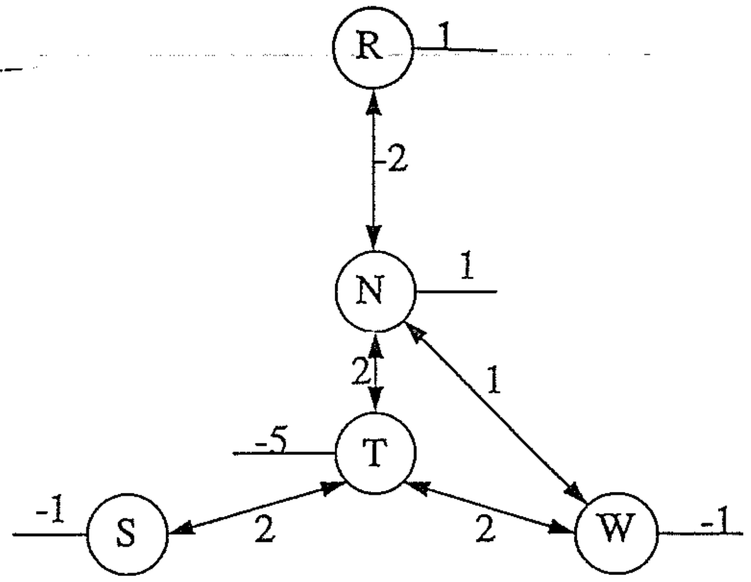

Figure 2.1: A quadratic network that represents the energy function $E = -2NT - 2ST - 2WT + 5T + 2RN - WN + W + S - R - N$. T is a hidden unit.

connections can be stored in a symmetric matrix whose diagonal is zero. The value of the i,j position within the matrix represents the weight of the connection that directs the output of unit i into unit j. The matrix is symmetric, since the weight from unit i to unit j is equal to the weight from unit j to unit i (i.e., $w_{i,j} = w_{j,i}$).

There is a direct mapping between the quadratic energy functions described in the previous subsections and the symmetric networks mentioned above. Given a function, we can construct the network that tries to minimize it, and given a network, we can generate the appropriate function that is minimized. The variables of the function map into nodes in the graph: hidden variables are mapped into hidden units and visible variables are mapped into visible units. Each node is connected by symmetric arcs to other units. Unit i is connected to unit j by a weight w iff the energy function includes a term of the form $-wx_ix_j$. A unit i has a non-zero bias θ (which is sometimes viewed as the threshold $-\theta$) iff the energy function includes a term of the form: $-\theta x_i$.

A network is fully specified by its energy function. For the remainder of this thesis, I will not distinguish between them. The terms "networks" and "energy functions" will be used interchangeably.

Example 2.4.1 The energy function $E = -2NT - 2ST - 2WT + 5T + 2RN - WN + W + S - R - N$ is represented by the symmetric network that appears in Figure 2.1.

2.5. Dynamics

Each unit computes an activation value (X_i) between zero and one as follows. The unit first computes the weighted sum of the inputs it receives from its neighbors, which is the gradient of the energy function with reverse sign

$$net_i = -\frac{dE}{dX_i} = \sum_{j=1}^{m} w_{i,j}X_j + \theta_i$$

The sum is then used as the input for some activation function F (usually non-linear and non-decreasing)

$$X_i = F(net_i),$$

whose task is to change the activation value according to the energy steepness. The network may be viewed, therefore, as performing a form of gradient descent on the energy landscape. Some of the most popular symmetric models are described in the following subsection.

2.5.1. The Discrete Hopfield Model

The discrete Hopfield model [Hopfield 82] uses binary-valued units whose activation values are either zero or one. The decision function F is

$$X_i = \begin{cases} 1 & \text{if } net_i > 0 \\ 0 & \text{otherwise} \end{cases}$$

This model searches the corners of the hyper-cube corresponding to the possible values of the units. The discrete Hopfield model finds a local minimum very quickly;[5] however, many times this local minimum will be a shallow one, and the network will not be able to escape to a deeper minimum.

[5]Fast on average but exponential in worst case [Kasif et al. 89].

2.5.2. The Analog Model of Hopfield and Tank

In Hopfield and Tank networks [Hopfield 84] the activation values are continuous between zero and one and the search takes place in the interior of the hyper-cube. By beginning near the center of the cube and searching using gradient descent, the network has better chances of finding a global minimum. There are no guarantees that a global minimum will be found, but good results have been reported for many problems. Hopfield and Tank use an analog circuit for their implementation and, therefore the energy function had to be modified slightly:

$$E = -\frac{1}{2}\sum_i \sum_j w_{i,j} X_j - \sum_i \theta_i X_i + \sum_i \frac{1}{R_i} \int_0^{X_i} g^{-1}(y)dy,$$

where R_i is the input resistance to unit i, $g(s)$ is the sigmoidal function with gain λ

$$g(s) = \frac{1}{1 + e^{2\lambda s}}$$

and g^{-1} is the inverse of g. The third term of this energy function approaches zero when the values of the units approach a corner of the hyper-cube (zeros and ones). It is minimized at $X_i = 1/2$. The system is described by differential equations, one for each unit:

$$\frac{dX_i}{dt} = -\frac{g^{-1}(X_i)}{R_i C} + \frac{dE}{dX_i}$$

where C is a constant that determines the speed of convergence. At high (infinite) gain the minima lie at the corners of the search space in the same locations as those of the discrete model. The discrete and analog models collapse, therefore, into one at infinite gain.

2.5.3. Boltzmann Machines

The Boltzmann machine [Hinton, Sejnowski 86] has binary units as in the discrete Hopfield model. The important difference is that the decision rule is stochastic and the system is annealed starting in a high temperature and slowly cooling it down to a lower temperature. The energy gradient net_i is used to determine the *probability* that a unit adopts the one state:

$$P(X_i = 1) = \frac{1}{1 + e^{-net_i/T}},$$

where T is the temperature of the annealing. With this stochastic rule, the network is more likely to adopt low energy states as the temperature cools down. The energy does not decrease monotonically as in the previous models; instead, it will go uphill randomly, with a frequency that is decreasing with the temperature. It can, therefore, search several minima at the same time, exploring a wide range of possibilities at high temperature but concentrating (spending more time) on deeper minima at lower temperature. It has been proven by [Geman 84] that after searching long enough with slow enough annealing, an equilibrium is obtained. At equilibrium the probabilities of the states are given by a function of the energy and the temperature, independently of the topology

$$\frac{P_\alpha}{P_\beta} = e^{-(E(\alpha)-E(\beta))/T}$$

A Boltzmann machine can theoretically be run long enough to guarantee that a global minimum is found; however, in practice it is not easy to find such "sure" annealing schedules.

An annealing schedule can be designed to fit a given time quota. Given a bound on time resources, we can make the system obtain its lowest temperature within the bound, thus providing the "any-time" property[6] that was mentioned in the desideratum of Section 1.3. This system is not guaranteed to find a global minimum at the end of such time quota; however, deeper minima will be found as more time is given, and the probability of finding a global solution increases. Given *enough* time the network is guaranteed to find a global solution.

2.5.4. Deterministic Boltzmann Machines

A new mean-field method suggested by [Peterson, Hartman 89] appears to reduce the excessive time in Boltzmann machines that is wasted on the stochastic hill climbing. The method is based on deterministically approximating the probability of a Boltzmann unit to be one and encoding this probability in the activation function:

$$X_i = tanh(\frac{net_i}{T}).$$

Mean-field annealing is performed similarly to the annealing in Boltzmann machines, but the process is not stochastic. Peterson and Anderson found this procedure to be 10-30 times faster than the stochastic process in Boltzmann machines and also reported that somewhat better results (deeper minima) were found. As in Boltzmann machines, mean-field annealing may be designed to fit the time resources, thus providing us with the desired any-time property.

2.6. Parallel Relaxation

The process of *relaxation* starts by assigning initial activation values to the units. After initialization, each unit computes the new activation value and continues to do so until a stable state (an equilibrium) is obtained, in which no more changes occur to the activation values. If the units are scheduled for execution asynchronously, SMs are guaranteed to arrive at a stable state and will not oscillate.

Inputs are given by clamping some of the visible units with fixed values. The output is obtained at the end of the relaxation process by observing the activation values of

[6]This is a "quantized" any-time algorithm. The algorithm performs many tries and in each try annealing is made. The answer is updated after each annealing and the probability to find a correct answer increases as more tries are performed.

the visible units. One of the questions asked in this thesis is how much symbolic computation can be done in a *single* parallel relaxation phase (as opposed to a sequence of relaxations as in [Touretzky, Hinton 88] and [Ballard 86]).

The stable states of models such as the discrete Hopfield network are the local minima of its energy function. However, models with simulated annealing or with deterministic annealing may be viewed as performing a search for a *global minimum.* Most of this thesis is, therefore, concerned with mappings between solutions of a specific problem and the global minima of an energy function that represents this problem. For many of the problems discussed in this thesis, I propose such mappings.

3. HIGH-ORDER AND LOW-ORDER NETWORKS

3.1. Introduction

This chapter introduces high-order symmetric models and shows an equivalence between them and the standard, quadratic models.

The main questions of this chapter are: What kind of meaning (semantics) can we give to the knowledge encapsulated in a symmetric network? Are low-order symmetric networks less expressive than high-order ones? What is the role of the hidden units? Are there meaning-preserving transformations between high-order networks and low-order networks?

The chapter is organized as follows. Section 3.2 introduces high-order networks. Section 3.3 proves an equivalence between high-order networks and low-order networks and provides transformations between them. Section 3.4 discusses the results.

3.2. High-Order Networks

3.2.1. Background

High-order connectionist networks use sigma-pi units with high-order multiplicative connections [Rumelhart et al. 86]. Each of the weights in a sigma-pi unit is associated with a subset of neighboring units (instead of a single neighbor). The unit computes the weighted product of the activation values of all the units in the subset ($wX_{i_1}X_{i_2}...X_{i_j}$) and sums all these products. It is common intuition that high-order networks (HONs) can better express high-order problems [Sejnowski 86] and may compute functions that are not trivially computable if only pairwise connections are allowed [Fahner 90], [Williams 86]. It is indeed easy to see that any boolean function can be computed using only one (linear) unit with high-order connections, since any boolean function can be implemented as a high-order polynomial. For further discussions on HONs see, for example, [Giles, Maxwell 87].

Symmetric networks, in particular, can be extended to handle high-order connections [Sejnowski 86]. Indeed, symmetric HONs were shown to have better memory capacity than standard quadratic networks [Bak, Little 88], [Lee et al. 86]. They also possess other advantages, such as faster learning and the ability to store higher-order predicates [Sejnowski 86]. Geffner and Pearl showed that symmetric networks with high-order connections are able to represent any probability distribution and thus have an expressive power equivalent to Bayes networks and Markov networks [Geffner, Pearl 87].

It is common knowledge that hidden units can increase the expressiveness of a network in some cases. Pearl showed (within the context of Bayes networks) that hidden units further increase the expressive power of second-order Bayes networks [Pearl 86]. A surprising negative result shown for constraint satisfaction networks proves that binary networks of constraints (i.e., with zero/one nodes) *cannot* express arbitrary constraints even if hidden nodes are added [Dechter 90].

This chapter explores the expressive power of high-order networks. It shows that second-order (quadratic) networks are equivalent to HONs if hidden units are added. There is nothing that can be expressed in high-order symmetric networks that cannot be expressed in low-order networks, if additional hidden units are used.

3.2.2. High-Order Energy Functions and Networks

A k-order energy function is a multi-linear polynomial $E : \{0,1\}^n \rightarrow \mathcal{R}$ that can be expressed as a weighted sum of products such that each product contains up to k variables. The sum of products for a k-order energy function is denoted by:

$$\begin{aligned} E^k(X_1,\ldots,X_n) = \; & -\textstyle\sum_{1\le i_1<i_2<\cdots<i_k\le n} w_{i_1,\ldots,i_k} X_{i_1}X_{i_2}\cdots X_{i_k} - \\ & \textstyle\sum_{1\le i_1<\cdots<i_{k-1}\le n} w_{i_1,\ldots,i_{k-1}} X_{i_1}X_{i_2}\cdots X_{i_{k-1}} \\ & - \cdots - \textstyle\sum_{1\le i\le n} w_i X_i. \end{aligned}$$

Symmetric models may be extended to minimize high-order functions [Sejnowski 86]. In the extended model the variables become nodes in a *hyper*-graph, and a weighted term $-w_{i_1\cdots i_k}X_{i_1}\cdots X_{i_k}$ of the function becomes a weight $w_{i_1,\ldots,i_k}$ of a symmetric hyper-arc that connects the nodes: $X_{i_1},\ldots,X_{i_k}$. Each node is a sigma-pi processing unit [Rumelhart et al. 86] that computes the energy gradient:

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

The unit then changes its activation value using:

$$a_i = F(net_i),$$

where F is the same activation rule (threshold, sigmoidal, stochastic, etc.) that is used in the specific model that was chosen for extension (discrete Hopfield, analog Hopfield, Boltzmann machine, etc.). For example , the discrete Hopfield model will set the activation value to one if $net_i > 0$; it will set it to zero if $net_i < 0$; no change occurs when $net_i = 0$.

A sigma-pi unit with hyper-arcs of order k multiplies up to $k-1$ inputs for each of its hyper-arcs. It is possible to show that in the extended model (as in the quadratic model), a stable state is guaranteed at either global or local minima [Sejnowski 86]. The multi-linearity (i.e., the lack of terms with powers of variables such as wXY^2Z or wY^3), causes the minimizing solutions to always be at the corners of the unit hyper-cube, even if the activation values are continuous between zero and one.

EXAMPLE 3.2.1 The cubic energy function $E = -NSW + 2RN - WN + W + S - R - N$ is represented by the hyper-graph of Figure 3.1. The units N, S, and W are connected by a hyper-arc and therefore they are sigma-pi units.

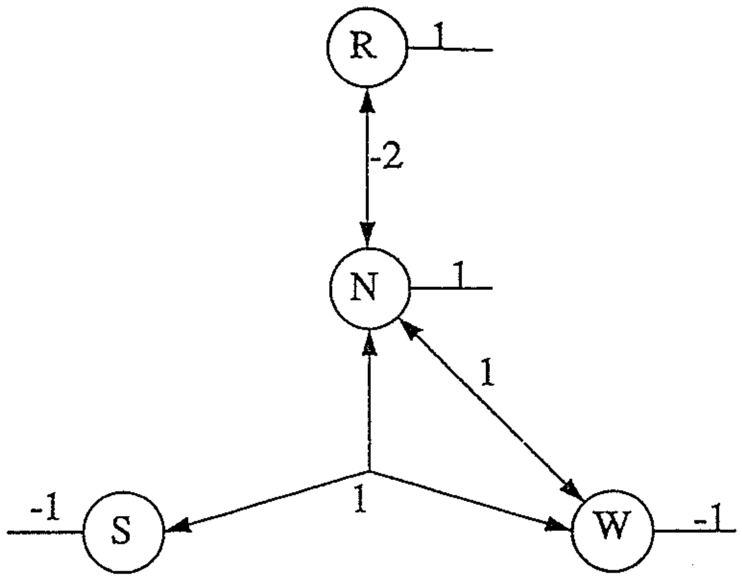

Figure 3.1: A cubic network that represents $E = -NSW + 2RN - WN + W + S - R - N$ using sigma-pi units.

3.3. The Equivalence Between High-Order and Low-Order Models

The next step is to show that high-order networks (HONs) and low-order networks (LONs) are strongly equivalent. In this section, an efficient algorithm is presented that transforms any HON into an equivalent LON by adding a maximum of $\sum_{i \in arcs}(k_i - 2)$ hidden units, where k_i is the order of the arc i. I also give an algorithm to eliminate hidden units by introducing higher level connections, generating an equivalent high-order network that has no hidden units.

3.3.1. Possible Semantics for SNs

First, we need to define in what sense two networks may be considered equivalent. The definition should reflect the equivalence of the knowledge encapsulated in the networks. This knowledge is the *interpretation* (the meaning) we assign to the network's topology and weights.

Assume first that every visible unit represents a proposition. The activation values in this case represent the truth values assigned to the propositions. An activation value $X_i = 1$ means that the proposition i is true, while $X_i = 0$ means that the proposition is false. The possible states of the visible units correspond to the possible models (truth assignments).[1] A network applies constraints on the possible models. This knowledge may be obtained by looking at the energy function associated with the network.

Let us define the characteristic function of a network:

Definition **3.3.1** The *characteristic* function of an energy function $E(\vec{X},\vec{T})$ with $\vec{X}$ visible variables and $\vec{T}$ hidden variables is the function

$$rank_E(\vec{X}) = min_{\vec{T}}\{E(\vec{X},\vec{T})\}.$$

Thus, the *rank* of a model is the energy level obtained by clamping the visible units with the truth values specified by the model and letting the hidden units be free to settle so that the minimum is obtained. It is thus capable of capturing the behavior of the network independent of the hidden units.

Let us consider three ways of interpreting this rank information:

1. We may assign meaning to the magnitude of the rank of each model. One possibility is to interpret the rank as a likelihood measure as in [Derthick 88], for example, the probability ratio of two states is a function of their energy difference; i.e., $\frac{P(\alpha)}{P(\beta)} = e^{rank_E(\alpha)-rank_E(\beta)}$. This is similar to the way the likelihood of the states of a Boltzmann machine at equilibrium is computed.

2. We may consider the *order* among the models which is induced by the rank function as the important information. The order gives us qualitative knowledge about preference of models. We prefer a model $\vec{x}$ over a model $\vec{y}$ iff $rank_E(\vec{x}) < rank_E(\vec{y})$. This interpretation is called *preferential semantics* and is used frequently for nonmonotonic model-theories [Shoham 88].

3. We may interpret the rank function as a mechanism that determines the set of "good" (preferred) models as opposed to the set of "bad" models. The good models are those that minimize the rank function. This interpretation is similar to the semantics which is usually given to standard propositional logic (truth tables).

Corresponding to the three possible types of interpretations, I define now three types of equivalence relations between networks.

Definition **3.3.2** Let E_1 and E_2 be energy functions/networks that have the same set of hidden variables.

1. E_1 is *strongly equivalent* to E_2 ($E_1 \overset{s}{\approx} E_2$) iff their corresponding characteristic functions are equal, up to a constant difference; i.e., $rank_{E_1} = rank_{E_2} + c$. This equivalence is called "magnitude preserving" since the two strongly equivalent networks have the same probabilistic interpretation.

[1] I'll use interchangeably the terms "model" and "truth-assignment", which are synonymous in this context.

2. E_1 is *p-equivalent* to E_2 ($E_1 \overset{p}{\approx} E_2$) iff their associated characteristic functions induce the same ordering over the set of truth assignments; i.e., $\forall \vec{x}, \vec{y}, rank_{E_1}(\vec{x}) < rank_{E_1}(\vec{y})$ iff $rank_{E_2}(\vec{x}) < rank_{E_2}(\vec{y})$. This equivalence is called "preference preserving", since both E_1 and E_2 induce the same partial order over the models.

3. E_1 is *weakly equivalent* to E_2 ($E_1 \overset{w}{\approx} E_2$) iff their corresponding characteristic functions have the same set of minimizing models; i.e., $\Gamma_{E_1} = \Gamma_{E_2}$. This equivalence is called "minima preserving", since both E_1 and E_2 have the same global minima.

The next observation assures us that if the transformation is magnitude preserving, then the other two interpretations are also preserved. In addition, if a transformation is preference preserving, then it is also minima preserving.

OBSERVATION:

1. If two networks are p-equivalent then they are also weakly equivalent.

2. If two networks are strongly equivalent, then they are also p-equivalent.

Proof:

1) If $E_1 \overset{p}{\approx} E_2$ then $rank_{E_1}(\alpha) < rank_{E_1}(\beta)$ iff $rank_{E_2}(\alpha) < rank_{E_2}(\beta)$. Therefore, the set of minimizing models of E_1 is exactly equal to the minimizing models of E_2.
2) If $E_1 \overset{s}{\approx} E_2$ then $rank_{E_1} = rank_{E_2} + c$. Therefore, $rank_{E_1}(\alpha) < rank_{E_1}(\beta)$ iff $rank_{E_2}(\alpha) < rank_{E_2}(\beta)$. □

The following example shows two strongly equivalent energy functions (X, Y, Z are the visible variables, T is hidden).

EXAMPLE **3.3.1** E_1 *approx* E_2 where
$E_1 = 5XY - 3YZ - XYZ$
$E_2 = 5XY - 3YZ - 2XT - 2YT - 2ZT + 5T$.
$E_1 = 5XY - 3YZ - XYZ \approx 5XY - 3YZ - 2XT - 2YT - 2ZT + 5T = E_2$.
The following table shows the values of E_1 and E_2 for all possible instantiations of the variables X, Y, Z, T:

XYZ	E_1	$XYZT$	E_2
000	0	0000	0
		0001	5
001	0	0010	0
		0011	3
010	0	0100	0
		0101	3
011	-3	0110	-3
		0111	-2
100	0	1000	0
		1001	3
101	0	1010	0
		1011	1
110	5	1100	5
		1101	6
111	1	1110	2
		1111	1

From the table we can conclude that $rank_{E_2} = rank_{E_1}$ and therefore the two functions are strongly equivalent. It is also possible to see that the order among the states is preserved and that the two functions have the same set of minimizing solutions $\{(011)\}$.

One of the most important operations we would like to use on networks is *update*. Adding energy terms to a given energy function enables us to change the constraints that are implemented by the network. In the next chapter we'll see that updating the knowledge encapsulated in a network is done by adding or deleting energy terms. Even a simple clamping operation can be interpreted as an addition of some large energy terms. Addition is therefore an important operation. We would like equivalent networks to remain equivalent after they are updated by addition.

The three types of equivalence differ in their behavior under the addition operation:
OBSERVATION:

1. If two networks E_1, E_2 are strongly equivalent, then for any given function e, $(E_1 + e)\overset{s}{\approx}(E_2 + e)$. This means that two strongly equivalent networks remain strongly equivalent after an arbitrary update.

2. A function e is **strict** iff $range(e) = \{0, \infty\}$, i.e., the function e produces only zero or infinity. If two networks are p-equivalent, then for every *strict* function e we add, the two updated networks are still p-equivalent; i.e., if $dom(e) = \{0, \infty\}$ and $E_1 \overset{p}{\approx} E_2$, then $(E_1 + e)\overset{p}{\approx}(E_2 + e)$. Note that as a special case, clamping unit X_i with either zero or one can be considered as adding the strict function $-\infty X_i$ or ∞X_i. Thus, networks that are p-equivalent remain so after clamping. We cannot guarantee this property for non-strict functions.

3. If two networks E_1, E_2 are only weakly equivalent, then we cannot guarantee that they will remain equivalent after any addition, even when we just clamp some units.

When we transform one network into another, we need to be aware of the need for updating. To be able to combine strict knowledge to our transformed knowledge, we will need to perform preference-preserving transformations. We will need "magnitude preserving" transformations (strong equivalence) if we want to perform arbitrary update or give a probabilistic interpretation to our transformed knowledge. The transformations in the remainder of this chapter are "magnitude preserving"; thus, additivity is preserved for any update. In the next chapter we will discuss transformations that are only minima-preserving (weak).

The next step is to show that there are magnitude preserving transformations between HONs and LONs.

3.3.2. Converting a High-Order Energy Function into a Lower-Order Function

Every k-order energy function E can be transformed into a strongly equivalent $(k-1)$-order energy function by adding extra hidden variables. The transformation is done by replacing each of the k-order terms in E with a sum of at most $(k-1)$-order expressions:

LEMMA **3.3.1** *Any k-order term $(\alpha \prod_{i=1}^{k} X_i)$, with a* NEGATIVE *coefficient α, can be replaced by a sum of quadratic terms of the form $\sum_{i=1}^{k} 2\alpha X_i T - (2k-1)\alpha T$ generating a strongly equivalent energy function with one additional hidden variable.*

Proof:
See Appendix A.1 $\square$

EXAMPLE **3.3.2**

$$XY - 3XYZU \approx XY - 6XT - 6YT - 6ZT - 6UT + 21T$$

LEMMA **3.3.2** *Any k-order term $(\alpha \prod_{i=1}^{k} X_i)$, with a* POSITIVE *coefficient α, can be replaced by a sum of terms (of order $(k-1)$) of the form $\alpha \prod_{i=1}^{k-1} x_i - \sum_{i=1}^{k-1} 2\alpha X_i T + 2\alpha X_k T + (2k-3)\alpha T$, generating a strongly equivalent energy function with one additional hidden variable.*[2]

Proof:
See Appendix A.2 $\square$

[2]A symmetric transformation is also possible at the cost of three hidden units (instead of just one).

EXAMPLE **3.3.3**

$$\begin{aligned}-XY+XYZU &\approx -XY+XYZ-2XT-2YT-2ZT+2UT+5T\\ &\approx -XY+XY-2XT'-2YT'+2ZT'+3T'-2XT-2YT-\\ &\quad 2ZT+2UT+5T\\ &= -2XT'-2YT'+2ZT'+3T'-2XT-2YT-2ZT+2UT+5T\end{aligned}$$

THEOREM **3.3.1** EVERY k-ORDER ENERGY FUNCTION E CAN BE TRANSFORMED INTO A STRONGLY EQUIVALENT $(k-1)$-ORDER ENERGY FUNCTION BY ADDING EXTRA HIDDEN VARIABLES. TRANSFORMATION IS DONE BY REPLACING EACH OF THE k-ORDER TERMS IN E BY A $(k-1)$-ORDER EXPRESSION, WHICH IS SPECIFIED BY LEMMA 3.3.1 AND LEMMA 3.3.2.

Proof:
Using the previous lemmas, we can reduce any k-order term of E into an expression of at most $(k-1)$-order. In each step we obtain a strongly equivalent function. □

EXAMPLE **3.3.4** The cubic energy function $E=-NSW+2RN-WN+W+S-R-N$ is equivalent to $-2NT-2ST-2WT+5T+2RN-WN+W+S-R-N$ by using additional hidden variable T. The corresponding networks appear in Figure 3.1 and Figure 2.1.

3.3.3. Eliminating Hidden Variables

The symmetric transformation, from low-order into high-order energy by eliminating hidden variables, is also possible.

THEOREM **3.3.2** EVERY k-ORDER ENERGY FUNCTION WITH ONE HIDDEN VARIABLE T, CAN BE TRANSFORMED INTO AN EQUIVALENT HIGHER ORDER ENERGY FUNCTION THAT DOES NOT INCLUDE T.

Method: Assume T is a hidden variable to be eliminated.
Let $\alpha_j T\prod_{i=1}^{l_j} X_{j_i}$ be a (l_j+1)-order term which connects the unit T.
Combine all these terms to form $(\sum_{j=1}^{k}\alpha_j\prod_{i=1}^{l_j} X_{j_i})T$ and call it "old-term".
Replace this "old-term" with a new term that is generated using the following procedure:

Consider all assignments $S=(x_{i_1},\ldots,x_{i_l})$, to the variables $X_{i_1},\ldots X_{i_l}$ such that

$$\beta_S=\sum_{j=1}^{k}\alpha_j\prod_{i=1}^{l_j} x_{j_i}<0.$$

An assignment S with $\beta_S<0$ is called a *negative assignment.*

If we clamp the visible variables using a negative assignment S, the unit T will settle on the value one (generating a negative energy contribution). All other instantiations will cause T to settle on zero (generating zero energy).

Let us now create a new term (without T) that generates the same energy (β_S) for the negative assignments, and zero energy for all other assignments.

For every instantiation $(x_{i_1}, \ldots, x_{i_l})$ obtained by assignment S, let the function L_S^j be:

$$L_S^j(\hat{X}) = \begin{cases} X_{i_j} & \text{if } x_i = 1 \\ 1 - X_{i_j} & \text{if } x_i = 0. \end{cases}$$

Then, generate the term:

$$newterm = \sum_{\text{S such that } \beta_S < 0} \beta_S \prod_{j=1}^{l} L_S^j(\hat{X}).$$

If we replace the old term: $(\sum_{j=1}^{k} \alpha_j \prod_{i=1}^{l_k} X_{j_i})T$ in E with the new term above, we obtain a function E', without the hidden variable T, which is strongly equivalent to E.

Proof sketch: The term $\prod_{j=1}^{l} L_S^j(\hat{X})$ is one for assignment S and zero for all the other assignments. Thus, the new term generates the same energy values (β_S) as the old term for all the assignments that cause T to be one. The rest of the assignments cause both terms to be zero. Therefore, if we instantiate all the variables (except T) and let T settle so that E is minimized, we obtain the same energy rank as was obtained from E' using the same instantiation. Therefore $rank_E = rank_{E'}$, and the two functions are strongly equivalent. $\square$

EXAMPLE **3.3.5** Let T be the hidden variable to be eliminated.
$AB + TAC - TA + 2TB - T = AB + T(AC - A + 2B - 1)$
The following assignments for (A, B, C) cause β to be less than zero:
$\beta_{(0,0,0)} = -1$; $\beta_{(0,0,1)} = -1$;
$\beta_{(1,0,0)} = -2$; $\beta_{(1,0,1)} = -1$;
The new term equals:
$-(1-A)(1-B)(1-C) - (1-A)(1-B)C - 2A(1-B)(1-C) - A(1-B)C$
$= -ABC + AB + AC - A + B - 1$
Therefore: $AB + TAC - TA + 2TB - T \approx -ABC + 2AB + AC - A + B$.

Note that $(1, 0, 0)$ uniquely minimizes both functions, ,and the energy of the two functions is equal for all assignments of the visible variables.

3.3.4. Analysis

Space Complexity. A tradeoff exists between the order of the connections and the number of hidden units needed to replace them. We saw that one k-order term can be converted into $O(k)$ terms of lower order with a single additional hidden variable that has a fan-out of k. To convert *any* k-order function into a quadratic one we may need $\sum_{i=3}^{k} \binom{n}{i}$ new variables; therefore the worst case to convert any n-order function into a quadratic one uses $O(2^n)$ new variables.[3] When the HON has l connections of

[3]Note that the original function may have an exponential number of terms.

maximal k-order, the equivalent quadratic network has maximum $l(k-2)$ hidden units and $O(lk^2)$ pairwise connections.

A tradeoff also exists between the order of the connections and the connectivity of the hidden units (the number of units that share an arc with a hidden unit) of the variables. Eliminating a variable with a connectivity of k may result in the creation of k-order connections, while reducing the order of a k-order term results in adding new variables with connectivity of k.

Time Complexity. The algorithm to convert HONs to LONs has time complexity $O(lk^2)$. It is linear in the number of connections, but it is quadratic in their order.

The algorithm to eliminate hidden units is $O(h2^m)$, where h is the number of hidden units to eliminate and m is their connectivity.

3.4. Discussion

The main result of this chapter is the identification of procedures to transform high-order energy functions into quadratic energy functions and vice versa, by adding or eliminating hidden variables. The process reveals a tradeoff between hidden units and the order of the connections in the network. We may conclude that sigma-pi units are not needed if we agree to add more hidden units to the network. Similarly, hidden units are not needed if sigma-pi capabilities are added to the visible units. As a result, all the networks presented in this thesis can be implemented using quadratic architectures.

We can construct a high-order architecture where all the units are connected to each other, but each unit maintains weights that are associated with subsets of the neighbor-nodes. Despite the equivalence of high-order and low-order connections, this high-order architecture has some properties that make it attractive:

1. The state space for HONs is smaller (fewer units) than that of the equivalent quadratic architectures. As a result, fewer spurious local minima exist.
2. The high-order architecture is universal; every function and arbitrary constraints (see next chapter) can be implemented in such architecture without changing its topology.

The problems with a universal high-order network are: 1) the need for an exponential number of weights, e.g., if we want to implement any boolean function[4]; 2) the difficulty in implementing multiplicative connections in analog hardware.

Using digital technology, however, we can implement a universal high-order circuit by wiring only pairs of units and storing multiplicative weights in the memory of each of the processing units. Although in theory we need exponential memory, a polynomial-size memory is enough for implementing any polynomial size energy function.

[4]Note that exponential number of weights is also needed in the quadratic case.

The transformations from HONs to LONs and from LONs to HONs preserve three possible interpretations that we may give to a network. One important property that is preserved is the ability to combine networks and to update their knowledge by adding or deleting energy terms. The result of an update applied to a transformed network is equivalent to the result of applying the same update to the original network. We will see later that this property is important for *incrementally updating* knowledge, one of the properties mentioned in the engineering desideratum of Section 1.3.

4. REPRESENTING AND LEARNING PROPOSITIONAL LOGIC

4.1. Introduction

The goal of this chapter is to start building formal foundations for possible mappings between symbolic mechanisms and connectionist networks. These foundations are based on a simple and well understood logical system: *propositional calculus.* The results of this chapter show tight relationships between propositional logic and symmetric networks (SNs).

The main questions to be answered in this chapter are: Can we efficiently express propositional logic formulas in SNs? Can we specify and describe a SN using the language of propositional logic? Can we construct a network to perform inferences in propositional logic? Can we incrementally update the knowledge captured in a network by local changes? Can SNs learn representations of propositional formulas inductively?

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 knowledge that is captured in SNs can be described by propositional formulas. 4) A connectionist implementation of a satisfiability problem solver has a better performance on average than standard approaches.

The chapter is organized as follows. Section 4.2 shows that propositional logic can be represented in SNs and that every SN can be described by a propositional formula. Networks are viewed as satisfiability problem solvers that search for an assignment that satisfies the constraints imposed by the formula. Section 4.3 provides surprising experimental data. Section 4.4 describes a learning algorithm that enables a network to obtain representations of formulas by observing examples, and Section 4.5 discusses the results and related work.

4.2. Propositional Logic and Energy Functions

4.2.1. Propositional Logic - A Review

DEFINITION **4.2.1** 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 **4.2.2** The *characteristic function* of a formula φ in n variables is defined to be $H_\varphi : 2^n \rightarrow \{0,1\}$ such that:

$\bullet H_{X_i}(X_1, \ldots, X_n) = X_i$

$\bullet H_{\neg\varphi}(X_1, \ldots, X_n) = 1 - H_\varphi(X_1, \ldots, X_n)$

$$\bullet H_{(\varphi_1 \vee \varphi_2)}(X_1, \ldots, X_n) = H_{\varphi_1}(X_1, \ldots, X_n) + H_{\varphi_2}(X_1, \ldots, X_n) - H_{\varphi_1}(X_1, \ldots, X_n) \times H_{\varphi_2}(X_1, \ldots, X_n)$$

$\bullet H_{(\varphi_1 \wedge \varphi_2)}(X_1, \ldots, X_n) = H_{\varphi_1}(X_1, \ldots, X_n) \times H_{\varphi_2}(X_1, \ldots, X_n)$

$\bullet H_{(\varphi_1 \rightarrow \varphi_2)}(X_1, \ldots, X_n) = H_{(\neg\varphi_1 \vee \varphi_2)}(X_1, \ldots, X_n)$

DEFINITION **4.2.3** 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 $\vec{x}$ *satisfies* a WFF φ, iff its characteristic function H_φ is evaluated to "one" given the vector $\vec{x}$.

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

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

DEFINITION **4.2.5** *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 $((A \vee \neg B) \wedge B) \models A$.
Note that the meaning assigned to a formula is weak in the sense that the characteristic function distinguishes only between models that satisfy the formula or that violate it (the third type of Section 3.3.1). A propositional formula does not induce a rank over the models nor an order.

4.2.2. Equivalence between Propositional Formulas

In order to convert a WFF *efficiently* into an energy function, we need an intermediate step. The WFF must be 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 (like the division of the variables of an energy function in the previous chapters). Atomic propositions that are of interest for a certain application are called "visible variables" and are denoted by $\vec{X}$. Additional atomic hidden propositions (denoted by $\vec{T}$) may be added without changing the set of relevant models that satisfy the WFF.

DEFINITION 4.2.6 The set of models that satisfy φ projected onto the visible variables is then called the *visible satisfying models*; i.e., $\{\vec{x} \mid (\exists\vec{t})H_\varphi(\vec{x},\vec{t}) = 1\}$.

DEFINITION 4.2.7 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 4.2.8 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.[1]

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 ($\leftrightarrow$). A formal algorithm and its proof are given in Appendix A.3.

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 4.2.1 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.

4.2.3. Constructing Energy Functions from WFFs

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

DEFINITION 4.2.9 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 $\Gamma_\varphi = \{\vec{x} \mid (\exists\vec{t})H_\varphi(\vec{x},\vec{t}) = 1\}$
$= \{\vec{x} \mid (\exists\vec{t})E(\vec{x},\vec{t})) = MIN_{\vec{y}}\{E(\vec{y})\}\} = \Gamma_E$.

[1]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 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 $E_\varphi : \{0,1\}^n \to \mathcal{N}$, that for every model $\vec{x}$, penalizes sub-formulas of the WFF that are not satisfied by $\vec{x}$. It computes the characteristic of the negation of every sub-formula φ_i (in the upper level of the WFF's conjunctive structure):

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

If $\vec{x}$ does not satisfy φ_i then $\vec{x}$ satisfies $\neg\varphi_i$ and therefore $H_{\neg\varphi_i}(\vec{x}) = 1$. Similarly, if $\vec{x}$ satisfies φ_i then $H_{\neg\varphi_i}(\vec{x}) = 0$. 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 **4.2.1** *φ is satisfied by $\vec{x}$ iff E_φ is minimized by $\vec{x}$ and the global minimum is zero.*

Proof:
By induction on the level (k) of nesting of φ
Base: $k = 1$
$E_{X_i} = 1 - H_{X_i} = 1 - X_i = 0$ iff $X_i = 1$ (satisfied)
Similarly, $E_{X_i} = 1$ iff $X_i = 0$ (unsatisfied)
Induction Step:
$\neg\varphi$ is satisfied by $\vec{x}$ iff $H_\varphi = 0$ iff $E_{\neg\varphi} = H_\varphi$ is minimized to zero.
$\varphi_1 \vee \varphi_2$ is satisfied by S, iff $H_{\varphi_1\vee\varphi_2} = 1$ iff $H_{\neg(\varphi_1\vee\varphi_2)} = 0$ iff $H_{\neg\varphi_1\wedge\neg\varphi_2} = 0$ iff $E_{\varphi_1\vee\varphi_2}$ is minimized to zero by $\vec{x}$.
$\varphi_1 \wedge \varphi_2$ is satisfied by S, iff $H_{\varphi_1\wedge\varphi_2} = 1$ iff $H_{\neg(\varphi_1\wedge\varphi_2)} = 0$ iff $H_{\neg\varphi_1\vee\neg\varphi_2} = 0$ iff $H_{\neg\varphi_1} + H_{\neg\varphi_2} = 0$ iff $E_{\varphi_1\wedge\varphi_2}$ is minimized to zero by $\vec{x}$.
$\varphi_1 \to \varphi_2$ is satisfied iff $\neg\varphi_1 \vee \varphi_2$ is satisfied, iff $\neg\varphi_1 \vee \varphi_2$ is satisfied iff $E_{\neg\varphi_1\vee\varphi_2}$ is minimized to zero . $\square$

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

EXAMPLE **4.2.2**

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

The corresponding high-order network appears in Figure 4.1. Note, that the symbols of the atomic propositions of φ are overloaded and used also as the variables of E.[2]

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

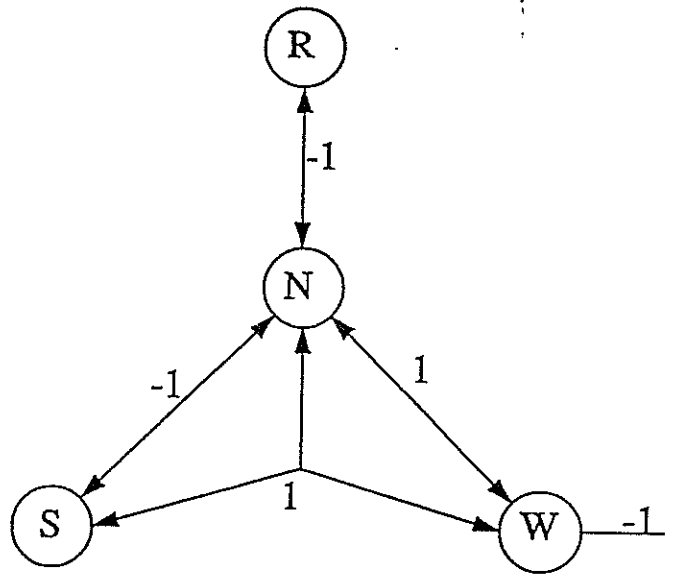

Figure 4.1: The network (cubic) that represents $((N \wedge S) \rightarrow W) \wedge (R \rightarrow \neg N) \wedge (N \vee \neg W)$.

Both $(-H_{\varphi})$ and E_{φ} may be used as the multilinear function whose minima correspond to the satisfying models. The function E_{φ} is preferred because it has heuristic knowledge about how "far" the energy of the current state is from a global minimum. E_{φ} counts the number of conjunctive sub-expressions that have not yet been satisfied. A direct use of $-H_{\varphi}$ with gradient descent will increase the chances of falling into a local minimum because the characteristic function does not give any indication for the direction in which to step when some constraints are not met. The function E_{φ}, on the other hand, suggests the direction that will satisfy as many subexpressions as possible. Another reason to prefer E_{φ} is that it generates lower-order energy terms.

THEOREM **4.2.1** IF φ IS SATISFIABLE THEN φ DESCRIBES E_{φ}.

Proof:
Immediately from Lemma 4.2.1 $\square$

COROLLARY **4.2.1** *φ is a tautology iff $E_{\varphi} = 0$.*

Proof:
If $E_{\varphi} = 0$, then for every instantiation $\vec{x}$, $E_{\varphi}(\vec{x}) = 0$.
Since E_{φ} describes φ, then every instantiation $\vec{x}$ satisfies φ and therefore φ is a tautology.
If φ is a tautology then any $\vec{x}$ satisfies φ. From Lemma 4.2.1, for all $\vec{x}$ $E_{\varphi}(\vec{x}) = 0$.
Let us now prove by induction that all the coefficients of the sum of products are 0.
Assume $E_{\varphi} = \sum w^{j}_{i_1,\ldots,i_j} \prod_{k=1}^{j} X_{i_k}$.
By taking $\vec{0}$ (the zero instantiation) all the variables become zero, and we can conclude

that the constant (w^0) in the sum of product is zero.
By induction, assume that $w^j_{i_1\dots,i_j} = 0$ for $j < k$. We can select an instantiation $\vec{x}$ that instantiates to zero all variables except $x_{i_1}, \dots, x_{i_k}$. Since $E(\vec{x}) = 0$ we can conclude that $w^k_{i1,\dots,x_{i_k}} = 0$ $\square$

EXAMPLE 4.2.3

$$\begin{aligned} E_{(A\vee(\neg A))} &= H_{((\neg A)\wedge A)} \\ &= (1-A)A = A - AA = 0 \end{aligned}$$

COROLLARY 4.2.2 *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.*

Proof:
From the definition of E_φ, the penalty of a conjunction of sub-formulas is equal to the sum of the penalties of each sub-formula. Each sub-formula has a maximum of k variables, and therefore its penalty is of order k (at most). $\square$

EXAMPLE 4.2.4

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

THEOREM 4.2.2 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:

1. Convert φ into CTF φ' (Section 4.2.2).

2. Convert the CTF φ' into a cubic energy function $E_{\varphi'}$ and simplify it to a sum of products form (Section 4.2.3).

3. Convert the cubic terms in $E_{\varphi'}$ into quadratic terms. Each of the triples generates only one new variable (Section 3.3.2 of the previous chapter).

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 4.2.5 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:

$$\begin{gathered} T_1 - 2AT_1 + A + T_2 - 2BT_2 + B + T_3 - T_1T_3 - T_2T_3 + T_1T_2 \\ -2T_1T_5 - 2T_2T_5 - 2T_3T_5 + 5T_5 + 2CT_4 + 2 - C - 2T_4 + T_3T_4 \end{gathered}$$

Corollary 4.2.3 *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.*

Proof:
The conversion of the WFF $(out \leftrightarrow \varphi)$, generates a network that sets the unit *out* to one iff the output of the function h given $\vec{x}$ is one. This is true since $h(\vec{x}) = 1$ iff $H_{\varphi}(\vec{x}) = 1$ iff $\vec{x}$ satisfies φ iff $\vec{x}$ satisfies *out*. $\Box$

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 on 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 $O(n)$.

4.2.4. Every Energy Function Describes Some Satisfiable WFF

This subsection shows that for any energy function E with no hidden variables, there exists a satisfiable WFF φ that describes E.

The procedure is first to find the set $\mu(E)$ of minimum energy states (the assignments $\vec{x}$ that minimize E). For each such state $\vec{x}$ create an n-way conjunctive formula of the variables or their negations depending on whether the variable is assigned 1 or 0 in that state. Each such conjunction $\bigwedge_{i=1}^{n} L_{\vec{x}}^{i}$ represents a minimum energy state, where

$$L_{\vec{x}}^{i} = \begin{cases} \text{“}X_i\text{”} & \text{if } x_i = 1 \\ \text{“}(\neg X_i)\text{”} & \text{if } x_i = 0 \end{cases} .$$

Finally the WFF is constructed by taking the disjunction of all the conjunctions:

$$\varphi_E = \bigvee_{\vec{x}\epsilon\mu(E)} (\bigwedge_{i=1}^{n} L_{\vec{x}}^{i}).$$

The satisfying truth assignments of φ_E correspond directly to the energy states of the network. The procedure is not efficient, but it does show the existence of such a formula for every network.

LEMMA **4.2.2** *For any k-order energy function E with no hidden variables there exists a satisfiable WFF φ_E that describes E.*

Proof:
$\vec{x} \in \mu(E)$ iff $\bigwedge_{i=1}^{n} L_{\vec{x}}^{i}$ is satisfied by $\vec{x}$ iff $\bigvee_{\vec{x}\in\mu E}(\bigwedge_{i=1}^{n} L_{\vec{x}}^{i})$ is satisfied by $\vec{x}$. $\square$
We therefore conclude:

THEOREM **4.2.3** EVERY ENERGY FUNCTION DESCRIBES SOME WFF.

The conversion is done by first eliminating the hidden variables using Theorem 3.3.2 of the previous chapter, and then computing the WFF by looking at the global minima as in Lemma 4.2.2.
Proof:
The elimination of hidden variables generates a strongly equivalent energy function E' that has no hidden variables. The function E' is then converted into the WFF $\varphi_{E'}$ that is satisfied for the same assignments that minimize E'. Since $E = E' + c$ (strong equivalence) $\vec{x}$ minimizes E' iff $\vec{x}$ minimizes E. $\square$

EXAMPLE **4.2.6** Let the energy function be:

$$E(X,Y) = -XY + 1.5X$$

Trying all instantiations:
$E(0,0) = 0$

$E(0,1) = 0$
$E(1,0) = 1.5$
$E(1,1) = 0.5$
The characteristic function of the WFF is:
$H(0,0) = 1$
$H(0,1) = 1$
$H(1,0) = 0$
$H(1,1) = 0$
The WFF that is described by E is therefore:

$$((\neg X \wedge \neg Y) \vee (\neg X \wedge Y))$$

4.3. 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.[3]

4.3.1. 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.
- 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

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

have been performed,[4] 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 Chapter 2.5.3).

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 Chapter 2.5.4), while the rest of the annealings are done using Boltzmann cycles.[5]

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 activation function for MFT (described in Chapter 2.5.4).

4.3.2. The Experiments

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

- 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].[6] 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.

[4]The number of cycles includes those executed during the annealing.

[5]Trying more MFT cycles is not a good strategy because MFT is deterministic.

[6]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 thesis.

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[7] 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 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.[8] For the purpose of fair comparison with GSAT, 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.[9] 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.

[7]Our experimental design which began after the presentation of the connectionist approach in the AAAI spring symposium of 1991, 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].

[8]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.

[9]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	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:

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.[10]

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.

4.4. 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

[10]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.

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 exactly equal to the vectors presented[11] 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 (see previous chapter).

DEFINITION 4.4.1 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.[12]

4.4.1. A Learning Rule for High-Order Symmetric Connections

Let an instantiation of the visible units $X_1, ..., X_n$ be a vector $\vec{x} = (x_1, ...x_n)$, 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

[11]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.

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

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.

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.[13] 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 $\vec{x} = (x_1, ..., x_n)$ 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 4.4.1 Given the presentation $ABC = 011$, a 2-clause rule causes the following updates:
- The weight of arc $\{AB\}$ is decremented $(\triangle_{AB} = -1)$, 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 $(\triangle_{BC} = +1)$, 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 $(\triangle_B = +1)$, 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 includes no zeros (even).
The bias of A is not updated since no 2-bit pattern contains a zero instantiated unit

[13]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.

other than A.
In a similar way, the arc $\{AC\}$ is decremented, and the bias C is incremented.

4.4.2. 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 **4.4.1** 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.

Proof:
See Appendix A.4 □

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 **4.4.2** 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 $ABC = 000$:
$\triangle_{ABC} = -1; \triangle_{AC} = +1; \triangle_{BC} = +1; \triangle_{AB} = +1; \triangle_{A} = -1; \triangle_{B} = -1; \triangle_{C} = -1;$
Given $ABC = 110$:
$\triangle_{ABC} = -1; \triangle_{AB} = +1;$

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 4.2(a). Its