ELSEVIER

Artificial Intelligence 77 (1995) 203-247

Artificial Intelligence

Reasoning, nonmonotonicity and learning in connectionist networks that capture propositional knowledge *

Gadi Pinkas

Amdocs Inc., 1161 Des Peres Rd., Suite 170, St Louis, MO 63131, USA

Received March 1992; revised January 1994

Abstract

The paper presents a connectionist framework that is capable of representing and learning propositional knowledge. An extended version of propositional calculus is developed and is demonstrated to be useful for nonmonotonic reasoning, dealing with conflicting beliefs and for coping with inconsistency generated by unreliable knowledge sources. Formulas of the extended calculus are proved to be equivalent in a very strong sense to symmetric networks (like Hopfield networks and Boltzmann machines), and efficient algorithms are given for translating back and forth between the two forms of knowledge representation. A fast learning procedure is presented that allows symmetric networks to learn representations of unknown logic formulas by looking at examples. A connectionist inference engine is then sketched whose knowledge is either compiled from a symbolic representation or learned inductively from training examples. Experiments with large scale randomly generated formulas suggest that the parallel local search that is executed by the networks is extremely fast on average. Finally, it is shown that the extended logic can be used as a high-level specification language for connectionist networks, into which several recent symbolic systems may be mapped. The paper demonstrates how a rigorous bridge can be constructed that ties together the (sometimes opposing) connectionist and symbolic approaches.

1. Introduction

Humans seem to be able to reason about the surrounding world from a noisy and incomplete knowledge with remarkably high speed. They are astoundingly good at inferring useful and reliable information even from conflicting beliefs and from a knowledge that is self-contradicting and sometimes even erroneous.

* The work was supported by NSF grant: R-9008012.

SSDI 0004-3702(94)00032-V

It has been more than a decade now that AI has realized that the analysis of such reasoning mechanisms is a major task. As a result, many nonmonotonic systems have been proposed as formal models for this kind of reasoning. Some well known examples are circumscription [32] and default logic [53].

Research in nonmonotonic reasoning has tried to understand the basic mechanisms and the rationale behind our intuition when dealing with incomplete description of the world. Recent nonmonotonic systems are quite successful in capturing our intuitions (for examples see [10,62]). Most systems, however, are still plagued with intractable computational complexity, sensitivity to noise and inflexibility to adjust themselves to new situations, to revise their knowledge and to develop personal intuitions. The symbolic approach is too rigid and specialized and is too constrained to be able to deal with exceptions to rules or with the fuzzy or approximate aspects of knowledge [35].

Connectionist systems may be the missing link. They can provide us with a fast, noise-tolerant, adaptive platform, and their ability to learn may be used to dynamically change the knowledge base, to achieve robustness and to accelerate the system's performance in familiar situations.

While scientists in traditional, symbolic AI were concentrating on development of powerful knowledge representation systems, connectionists were concentrating on powerful learning and adaptation mechanisms. Connectionism was criticized for lacking mechanisms like compositionality and systematicity, which are essential for high-level cognitive tasks and are easy for symbolic approaches [8]. We would like to have systems that have sufficient expressive power, that perform fast and that are capable of learning and adjusting. "Clearly, the ultimate goal for both scientific approaches is to find efficient learning procedures for representationally powerful systems" [16]. Once symbolic computations are mapped into the connectionist platform, the hope is that the fuzzy, heuristic and adaptive capabilities of the connectionist approach will be more naturally integrated with the expressive but rigid symbolic approach. [1]

This article concentrates on the somewhat neglected problem of symbolic knowledge representation in connectionist networks. In particular it studies networks that can represent and learn unrestricted propositional [2] rules.

One big difference between connectionist networks and symbolic knowledge representations is that symbolic systems need an interpreter to process the information expressed in the representation, and to reason with it. Connectionist networks have no such interpreter. The interpreter and the control mechanism should be included in the knowledge that is being represented. We strive therefore to find a connectionist representation that is capable of representing the information, as well as the procedural knowledge that is needed for controlling the reasoning process.

Among the different connectionist models, I choose to consider those with a symmetric matrix of weights. This family of models includes Hopfield networks [20,21], Boltzmann machines [17], harmony theory [63], mean field theory [15], and other

[1] Following Marvin Minsky's call for synthesizing the symbolic and connectionist approaches [35].

[2] The approach can be extended to represent predicate logic as well [43].

variations. The reasons for selecting *symmetric* connectionist networks (SCNs) are the following:

(1) symmetric networks can be characterized by energy functions which make it easier to specify the networks' behavior [7];

(2) symmetric networks were used to express and approximate "hard" problems [22];[3]

(3) symmetric networks are capable of representing a large set of asymmetric networks [42]; therefore they are quite powerful, and we will not lose expressive power if we restrict ourselves to the symmetric case.[4]

(4) recent, successful, heuristic repair techniques [33,57] have very similar structure and may be seen as sequential variations of the symmetric paradigm.

Ideally, we would like a wide range of logical formalisms to be representable in connectionist networks; however, we will be satisfied even if only some but general nonmonotonic paradigms will be mapped. Also, it would be beneficial if we had a formal, declarative language that is capable of describing the knowledge encapsulated in a network. Such high-level, declarative language may then be used for specification and "programming" of connectionist networks. It may be used also as an intermediate level of abstraction between high-level cognitive processes and their low-level neural implementation.

My purpose in this article is to show that (1) propositional nonmonotonic knowledge can be captured naturally in SCNs; (2) any knowledge that is encapsulated in any SCN can be described by an extended version of propositional logic; (3) SCNs can learn representations of unknown propositional formulas by looking at examples of truth assignments that satisfy the formulas; (4) SCNs can be used as inference mechanisms that are able of capturing both the information embedded in the logic formulas as well as the procedural knowledge used for control; and finally, (5) the algorithm used by SCNs can be compared favorably to the best-known algorithms for propositional logic satisfiability.

The paper is organized in the following way: Section 2 presents penalty logic, its semantics and its proof theory. The section demonstrates the usefulness of the new logic for nonmonotonic reasoning. In Section 3, symmetric connectionist networks are introduced and the energy paradigm is reviewed. Section 4 defines equivalence between forms of knowledge representation and proves a strong equivalence between penalty logic formulas and SCNs. It shows how to represent sentences of penalty logic in SCNs and how every SCN may be described using the extended logic. Section 5 sketches a connectionist inference engine that uses the representation discussed. Section 6 discusses a learning algorithm that enables SCNs to learn representations of unknown propositional formulas inductively. Section 7 reports experiments that were performed to evaluate the performance of the approach. Section 8 discusses related work, and Section 9 concludes.

[3] The TSP experiments of Hopfield and Tank [22] were criticized by Wilson and Pawley [67]; however, later formulations of the energy function as well as modifications to the Hopfield architecture, provided better and more encouraging results [2,23,39].

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

2. Penalty logic

I'll extend propositional calculus in order to make it useful for nonmonotonic reasoning and for coping with inconsistency. Later, this calculus will be mapped into SCNs, and a strong relationship between the logic and the networks will be revealed.

The extended calculus is capable of expressing strength of belief, reliability of sources of knowledge, etc., by adding a real positive number (penalty) to every belief. This penalty may be assigned a variety of interpretations, for example, the numbers may represent "certainties" or "likelihoods" as in [6,61], priorities as in [3,29] or maximal entropy constraints as in [11]. When the knowledge sources are unreliable, a penalty may represent a measure of reliability [51]. Note that some of these systems compute the penalties from less explicit information, while other systems let the user specify the penalties explicitly.[5] I do not insist on a particular use or interpretation of the penalties, since the intention is to develop a *general* framework into which many logicist systems could be reduced (possibly with a variety of interpretations).

2.1. Extending propositional calculus

Definition 2.1. A *penalty logic well formed formula* (PLOFF) ψ is a finite set of pairs. Each pair is composed of a real positive number, called *penalty*, and a standard propositional formula, called *assumption* (or belief); i.e., $\psi = \{\langle \rho_i, \varphi_i \rangle \mid \rho_i \in \mathbb{R}^+, \ \varphi_i \text{ is a WFF}, \ i = 1, \ldots, n\}$.

The set of the beliefs that are in ψ (denoted by $\mathcal{U}_\psi$) is $\mathcal{U}_\psi = \{\varphi_i \mid \langle \rho_i, \varphi_i \rangle \in \psi\}$.

Example 2.2. The Nixon diamond can be stated as:

1000	$N \rightarrow R$	Nixon is a republican
1000	$N \rightarrow Q$	Nixon is also a quaker
10	$R \rightarrow \neg P$	republicans tend not to be pacifist
10	$Q \rightarrow P$	quakers tend to be pacifist
3000	N	the person we reason about is Nixon.

An illustration of the example is shown in Fig. 1.

The set of the beliefs in the example is inconsistent; however, the penalties in this example reflect the strength with which we believe in each proposition. High penalty is given to strict logic rules (facts), like the one that states that Nixon is a republican. We cannot allow strict facts to be defeated. The last fact (N) states that Nixon is the one we reason about. This fact receives the highest penalty of all since it is considered as *evidence*. The evidence is not usually part of our knowledge base and we would like to "jump" to conclusions once it is given. The evidence in this case is considered temporary (corrigible) but very certain (infallible). Lower penalties are given to "defeasible" rules ("tend to be" rules), like the one that states that republicans tend not to be pacifist.

[5] The penalties may be acquired by learning; thus, subjective intuition is captured.

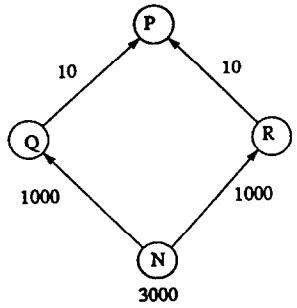

Fig. 1. An illustration of the Nixon diamond as an inheritance network: nodes represent atomic propositions; the numbers are the penalties.

When we know that somebody is a republican, we tend to believe that the person is not pacifist (by default); however, this "jumping" to conclusion is blocked, if we know that the person is an exception to the rule. For example, we wouldn't like to conclude that a quaker is pacifist, if the person is also a republican, or if it was explicitly mentioned that the person is not pacifist. Clearly, we would like to conclude that Nixon is both a republican (R) and a quaker (Q); however, we would not like to conclude anything about the pacifism of Nixon. There is no adequate reason to believe either in P or in $\neg P$; therefore P is considered ambiguous.

If we want to express the belief that religious ideas are stronger than political affiliations in influencing one's pacifism, then we may increase the penalty for $Q \rightarrow P$ to 15; leaving the penalty for $R \rightarrow \neg P$ unchanged (10).

Example 2.3.

1000	$N \rightarrow R$	Nixon is a republican
1000	$N \rightarrow Q$	Nixon is also a quaker
10	$R \rightarrow \neg P$	republican tend not to be pacifist
15	$Q \rightarrow P$	quakers tend to be pacifist
3000	N	the person we reason about is Nixon.

In the revised set of assumptions, we have two competing arguments. One argument supports the pacifism of Nixon while the other supports its negation. The pacifism of Nixon is not ambiguous in this case, since the argument that supports P wins the competition (the winning argument is stronger and therefore manages to defeat the disagreeing argument [30]).

2.2. *Model theory*

There are many ways to interpret the penalties and the assumptions in our formalism. I shall give one such interpretation that is convenient yet general.

Given a knowledge base $\psi = \{\langle \rho_i, \varphi_i \rangle\}$, the PLOFF ψ determines a ranking over the set of all possible models (truth assignments of n atomic propositions). This ranking

reflects "normality" or "goodness" we tend to associate with possible models of the world (see [60]). By specifying ψ, we mean informally that models that satisfy many "important" assumptions are "better" than models that satisfy fewer or less important assumptions. Every two models may always be compared by looking at the assumptions (in ψ) that are violated. Two models that violate the same set of assumptions are considered to be "equally good". Even if the models violate different or intersecting sets of assumptions but the sum of the penalties of both sets is the same, then the two models are "equally good". A model is more "normal" (or "better") than another model if the sum of the penalties of the violated assumptions of the first is less than the sum of the penalties violated by the second. For further motivation for summing the penalties see [6, 11].

This interpretation of the penalties induces a ranking function that assigns a real value (rank) to all the possible models. The ranking function that is induced is called the violation rank of ψ:

Definition 2.4. The *violation rank* of a PLOFF ψ is the function ($Vrank_\psi$) that assigns a real-valued rank to each of the truth assignments. The $Vrank_\psi$ for a truth assignment $\vec{x}$ is computed by summing the penalties for the assumptions of ψ that are violated by the assignment; i.e., $Vrank_\psi(\vec{x}) = \sum_{\vec{x} \not\models \varphi_i} \rho_i$.[6]

Definition 2.5. The models that minimize the $Vrank_\psi$ function are called the *preferred models* of ψ; i.e., $\{\vec{x} \mid \min_{\vec{y}}\{Vrank_\psi(\vec{y})\} = Vrank_\psi(\vec{x})\}$. The set of all preferred models is denoted by Γ_ψ.

Definition 2.6. Let φ, ψ be PLOFFs, a PLOFF ψ *semantically entails* φ ($\psi \models \varphi$) iff all the preferred models of ψ are also the preferred models of φ; i.e., $\Gamma_\psi \subseteq \Gamma_\varphi$.

Note that a sentence ψ therefore entails φ iff any model that minimizes the violation rank of ψ, also minimizes the violation rank of φ.

In the Nixon example, there are only two preferred models: (N, R, Q, P) and $(N, R, Q, \neg P)$. Examples of some valid conclusions are therefore $N, R \wedge Q$, etc. (since these conclusions are satisfied by all the preferred models). The pacifism of Nixon is ambiguous, since P holds in one preferred model while $\neg P$ holds in the other.

2.3. Merging PLOFFs, evidence and background knowledge

The operator *merge* ($\overset{*}{\cup}$) in the metalanguage, plays the role of $\wedge$ (AND) in classic propositional logic. It allows us to combine two PLOFFs simply by merging them.

Definition 2.7. The *merge* operation ($\overset{*}{\cup}$) is defined:

$$\psi_1 \overset{*}{\cup} \psi_2 = (\psi_1 - \psi_2) \cup (\psi_2 - \psi_1) \cup \{\langle 2\rho_i, \varphi_i\rangle \mid \langle \rho_i, \varphi_i\rangle \in \psi_1 \cap \psi_2\}.$$

[6] A score (a number) is actually computed by the function *Vrank*. This score is later used to determine the relative rank of the truth assignment.

The meaning of a merge of two or more PLOFFs is simply obtained by adding the appropriate *Vrank* functions. The reader may check that

$$Vrank_{\psi \overset{*}{\cup} \psi'} = Vrank_{\psi} + Vrank_{\psi'}.$$

The merge operation therefore allows us an incremental update of the knowledge.

Later, after the equivalence of networks and logic formulas is established, we'll see that this property allows one to add (delete) a PLOFF to an existing network only by adding (deleting) the relevant energy terms. There is no need to re-compute the new network all over again when some updates occur.

Nonmonotonic systems "jump" to conclusions based on a given evidence, and later may retract those conclusions based on a new evidence. It is therefore convenient to decompose the knowledge from which we want to reason, into "background" knowledge and "evidence" [10,49]. The background knowledge is relatively fixed, and there should be an easy way to combine evidence with it. In our formalism, combining the evidence is simply done by merging the evidence and the background or adding together the two ranking functions.

Definition 2.8. Let ψ, e, φ be PLOFFs. Evidence e entails φ with respect to a background knowledge ψ ($e \models^{\psi} \varphi$), iff $\psi \overset{*}{\cup} e \models \varphi$. The *consequence relation* induced by ψ is the set of all pairs $\langle e, \varphi \rangle$ such that $e \models^{\psi} \varphi$.

One special case of this definition is when the evidence is strict; i.e., its validity is certain. This special case is very useful, and indeed, in most reasoning systems the evidence is never defeated, and the agent draws conclusions based on the absolute validity of the evidence (e.g., [10]). To represent a strict evidence e in penalty logic, the set $\mathcal{U}_e$ should be consistent and the penalties that are assigned to the assumptions should be higher than any other combination of background beliefs (the penalties are practically infinite).

Definition 2.9. Strict evidence is a PLOFF $e = \{\langle \infty, e_i \rangle\}$ such that the set $\mathcal{U}_e = \{e_i\}$ is consistent and ∞ represents a large penalty (larger than any combination of background beliefs).

Example 2.10. In the Nixon diamond the following beliefs are considered background:

1000	$N \rightarrow R$	Nixon is a republican.
1000	$N \rightarrow Q$	Nixon is also a quaker.
10	$R \rightarrow \neg P$	republicans tend not to be pacifist.
10	$Q \rightarrow P$	quakers tend to be pacifist.

The fact $\langle 3000, N \rangle$ is strict evidence that triggers for example the conclusion of $Q \wedge R$. Another example of strict evidence is $\langle 3000, \neg Q \rangle$ that triggers the conclusion $\neg N$.

Penalty logic is nonmonotonic since sometimes conclusions need to be retracted when new evidence is added. In the example, when there is an evidence that someone is a

quaker the conclusion is that he or she is also pacifist. If in addition, we add the evidence that Nixon is that someone, we need to retract the conclusion.

Loyal to the goal of being as general as possible, I'll not restrict the evidence to being strict. Such generalization is not mere formality, and has its direct uses. Defeasible evidence is a phenomenon encountered in many practical applications. For example, when the evidence is obtained via sensory devices that are unreliable, redundant and noisy, our agent may "not believe its own eyes" if the evidence conflicts with some highly reliable facts of the background knowledge.

As with evidence, conclusions need not be strict. Most symbolic systems treat a conclusion as a strict proposition that either follows from the background knowledge, or its negation follows, or is ambiguous. Penalty logic allows conclusions to be stated as PLOFFs (with penalties). Thus, such conclusions may arrive for example as queries via noisy channels. The query we wish to prove may therefore be redundant and unreliable exactly as the evidence and the background knowledge. Thus, the query itself may be self-contradicting, yet we prove it (by definition) iff all the preferred models of the background evidence knowledge are also preferred models of the query.

2.4. *Proof theory*

A sound and complete proof theory can be shown for penalty logic. This proof theory is based solely on syntactic considerations, and gives a clarifying look on the reasoning process in penalty logic.

Instead of ranking the models and using the "best" models for the reasoning process, we can rank consistent subsets of the assumptions of ψ, and use the "best" (preferred) consistent subsets to perform deduction. A conclusion is made in the proof theory iff all the preferred consistent subsets entail it.

Definition 2.11. A set T is called a *theory* of a PLOFF ψ iff T is a *consistent* subset of the assumptions in ψ; i.e., the set $T \subseteq \mathcal{U}_\psi$ has at least one satisfying model.

Definition 2.12. The *penalty function* of a theory T of ψ is the function obtained by summing the penalties of the assumptions in ψ that are not included in T; i.e., $penalty_\psi(T) = \sum_{\varphi_i \in (\mathcal{U}_\psi - T)} \rho_i$.

A ranking is therefore induced by ψ over the set of theories of ψ. This ranking is computed by summing the penalties of the missing assumptions.

Definition 2.13. A *preferred* theory of ψ is a theory T that minimizes the penalty function of ψ; i.e., The set of the preferred theories of ψ is $T_\psi = \{T \mid penalty_\psi(T) = \min_S\{penalty_\psi(S) \mid S \text{ is a theory of } \psi\}\ T \text{ is a theory of } \psi\}$.

Definition 2.14. Let ψ, φ be PLOFFs, let $T_\psi = \{T_i\}$ the set of all preferred theories of ψ, and let $T_\varphi = \{T_i'\}$ the set of all preferred theories of φ. We say the ψ *entails*[7] φ

[7] Note that the deductive closure of preferred theories roughly resemble "extensions" (as in [53]). The definition of entailment in penalty logic resembles therefore entailment by intersection of all extensions.

(denoted by $\psi \vdash \varphi$) iff all the preferred theories T_i of ψ entail (in the classical sense) the disjunction of all the preferred theories of φ; i.e.,

$$\psi \vdash \varphi \text{ iff } \bigvee_{T \in T_\psi} T \vdash \bigvee_{T' \in T_\varphi} T'$$

As a special case, consider the case where the conclusion φ is strict (φ is a consistent propositional well formed formula). A PLOFF ψ entails φ iff every preferred theory of ψ entails φ in the classical sense of entailment.

In the Nixon example, the assumptions are conflicting (inconsistent set); however, there are $2^5 - 2$ non-empty consistent subsets where at least one belief of ψ is missing. If we rank each of the consistent subsets by summing the penalties of the missing beliefs, we get that the preferred theories are $T_1 = \{N, N \rightarrow Q, N \rightarrow R, Q \rightarrow P\}$ and $T_2 = \{N, N \rightarrow Q, N \rightarrow R, R \rightarrow \neg P\}$. These preferred theories are each ranked 10 since only one belief in ψ (of strength 10) is missing in each such theory.

Each of the two preferred theories entails the obvious conclusions (like $N, Q \wedge R$), but neither P nor $\neg P$ can be concluded, since the two preferred theories agree on neither. The reasoning process can be intuitively understood as a competition among consistent subsets. The subsets that win are those theories with minimal penalty. A conclusion is entailed only if all the winners conclude it independently.

We'll need the next two lemmas to show that the proof theory is sound and complete.

Lemma 2.15. *Let $T \subseteq \mathcal{U}_\psi$ be a consistent subset of the assumptions in ψ. The subset T is maximal-consistent*[8] *in ψ iff every model that satisfies T has a violation rank equal to the penalty of T; i.e., T is a maximal-consistent subset iff $(\forall \vec{x})$ if $\vec{x} \models T$ then $penalty_\psi(T) = Vrank_\psi(\vec{x})$.*

Proof. If T is a maximal-consistent subset of ψ, the assumptions in ψ that are left out of T are also the assumptions that are violated by any model $\vec{x}$ that satisfies T (otherwise such assumptions are consistent with T and therefore T is not maximal). Also, every assumption that is violated by a model $\vec{x}$ that satisfies T cannot be in T. Therefore, if T is a maximal-consistent subset then for every model $\vec{x}$ of T, the set of missing assumptions in T is equal to the set of assumptions violated by $\vec{x}$. Therefore $penalty_\psi(T) = Vrank_\psi(\vec{x})$.

Assume that every model $\vec{x}$ that satisfies T has $Vrank_\psi(\vec{x}) = penalty_\psi(T)$. If T is not maximal-consistent then there is an assumption in ψ that can be included into T and have a model $\vec{x}$ satisfying both T and the new assumption. The violation rank of $\vec{x}$ must be lower than the penalty of T since the set of assumptions not included in T subsumes the set of assumptions violated by $\vec{x}$ and contains at least one assumption not violated by $\vec{x}$; i.e., $Vrank_\psi(\vec{x}) < penalty_\psi(T)$. This is a contradiction with the assumption that $penalty_\psi(T) = Vrank_\psi(\vec{x})$. □

[8] A subset T is maximal-consistent if no other assumption of ψ can be added to T while still preserving the consistency of the set.

The reader may observe that any preferred theory of ψ is a maximal-consistent subset of $\mathcal{U}_\psi$ and therefore the penalty of a preferred theory is equal to the violation rank of its satisfying models. This allows us to use a proof-theoretic ranking function ($penalty_\psi$) instead of the model-theoretic function ($Vrank_\psi$).

The next lemma establishes the relationship between preferred models and preferred theories.

Lemma 2.16. *A model $\vec{x}$ is a preferred model of a PLOFF ψ iff model $\vec{x}$ satisfies some preferred theory of ψ.*

Proof. If $\vec{x}$ is a preferred model of ψ then it minimizes $Vrank_\psi$. Let T be the set composed of all assumptions in ψ that are satisfied by $\vec{x}$. Since the assumptions that are violated by $\vec{x}$ are exactly those that are not included in T, we deduce that $Vrank_\psi(\vec{x}) = penalty_\psi(T)$. But if T is not a preferred theory then there exists a preferred theory T' such that $penalty_\psi(T') < penalty_\psi(T)$. By Lemma 2.15, the models $\vec{y}$ that satisfy T' have $Vrank_\psi(\vec{y}) = penalty_\psi(T')$, and we conclude that

$$Vrank_\psi(\vec{y}) = penalty_\psi(T') < penalty_\psi(T) = Vrank_\psi(\vec{x}).$$

This is a contradiction to the minimality of $Vrank(\vec{x})$.

If $\vec{x}$ is a model of a preferred theory T of ψ then T minimizes the penalty function and is maximal-consistent. By Lemma 2.15, $Vrank_\psi(\vec{x}) = penalty_\psi(T)$. If $\vec{x}$ is not a preferred model of ψ then there must be a preferred model $\vec{y}$ such that $Vrank_\psi(\vec{y}) < Vrank_\psi(\vec{x})$. Let T' the set of all assumptions of ψ satisfied by $\vec{y}$. The set T' has $penalty_\psi(T') = Vrank_\psi(\vec{y})$, since the set of the assumptions violated by $\vec{y}$ is equal to the set of assumptions not included in T'. Therefore,

$$penalty_\psi(T') = Vrank_\psi(\vec{y}) < Vrank_\psi(\vec{x}) = penalty_\psi(T),$$

in contradiction to the minimality of $penalty_\psi(T)$. □

Theorem 2.17. *The proof procedure is sound and complete; i.e., $\psi \models \varphi$ iff $\psi \vdash \varphi$.*

Proof. If $\psi \models \varphi$ then every preferred model of ψ is also a preferred model of φ. Based on lemma 2.16, every preferred model of ψ satisfies some preferred theory T of ψ and also satisfies some preferred theory of φ. Therefore, every preferred model of ψ satisfies the disjunction of the preferred theories of φ. From lemma 2.16 every model that satisfies a preferred theory T of ψ is also a preferred model of ψ and therefore satisfies the disjunction of the preferred theories of φ; i.e., $T \vdash \bigvee_{T'_j \in \Gamma_\varphi} T'_j$. We conclude therefor that $\psi \vdash \varphi$.

If $\psi \vdash \varphi$ then every model that satisfies a preferred theory T of ψ also satisfies a preferred theory T' of φ. From lemma 2.16, a model that satisfies T' is also a preferred model of φ and therefore, every model that satisfies T is also a preferred model of φ. Based on lemma 2.16 every preferred model T of ψ satisfies some preferred theory of ψ and therefore is a preferred model of φ ($\Gamma_\psi \subseteq \Gamma_\varphi$). We therefore conclude that $\psi \models \varphi$. □

This sound and complete proof mechanism of competing theories is useful for both dealing with inconsistency in the knowledge base and for defeasible reasoning. For example, when we detect inconsistency, we usually want to adopt a theory with maximum cardinality (since we assume that only a minority of the observations are erroneous). Indeed, in penalty logic, when all the penalties are one, the theories that win have maximal cardinality and only a minority of the assumptions is defeated. Thus, minimum penalty means maximum cardinality. Penalty logic is therefore a generalization of the maximal cardinality principle which is useful when coping with noisy knowledge sources. For defeasible reasoning, the notion of conflicting theories can be used to decide between conflicting sets of arguments. Intuitively, a set of arguments A_1 defeats a conflicting set of arguments A_2 if A_1 is supported by a "better" theory than all the theories that support A_2 (see [30, 62] for a discussion on argument systems).

Example 2.18. Two levels of blocking (from [3]):

1	meeting	I usually go to the Monday meeting.
10	sick $\rightarrow$ ($\neg$meeting)	If I'm sick I usually don't go to the meeting.
100	cold-only $\rightarrow$ meeting	If I have only a cold then I tend go to the meeting.
1000	cold-only $\rightarrow$ sick	If I have a cold it means I'm sick.

Without any additional evidence, all the assumptions are consistent, and we can infer that "meeting" is true (from the first assumption). However, given the evidence that "sick" is true, we prefer theories that falsify "meeting" and "cold-only", since the second assumption has greater penalty than the competing first assumption (the only theory that wins does not include the first assumption). If we include the evidence "cold-only" then the theory that previously won loses now, and the new winner is the theory that does not include the second assumption. As a result, the conclusion "meeting" is drawn despite the fact that "sick" is also concluded.

3. Symmetric models and energy functions

This section reviews symmetric connectionist models and the energy minimization paradigm. Later, we'll show the relationship between this paradigm and penalty logic.

3.1. What are symmetric networks?

A symmetric connectionist network (SCN) is characterized by a weighted undirected graph whose nodes represent processing units and whose arcs represent weighted connections (see Fig. 2). There are two kinds of arcs: pairwise arcs that link two nodes, and monadic arcs that are each attached to a single node. A pairwise arc represents a weighted connection ($w_{i,j}$), while a monadic arc represents a bias (a threshold with reverse sign) given to a single unit. The weights of the 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., $w_{i,j} = w_{j,i}$).

An SCN may be viewed as searching for a global minimum of some quadratic function called the energy. Each unit asynchronously computes the gradient[9] of the function and adjust its activation value, so that energy decreases gradually.[10] The network eventually reaches equilibrium, settling on either a local or a global minimum.

There is a direct mapping between these networks and the quadratic energy functions they minimize. 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 nonzero 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 and for the remainder of this paper, I will not distinguish between them. The terms "networks" and "energy functions" will be used interchangeably.

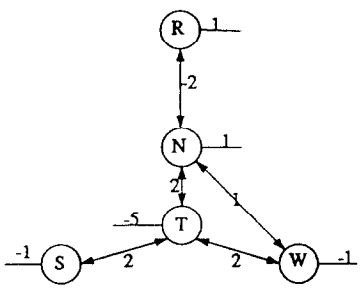

Fig. 2. A symmetric network that represents the function $E = 2RN - 2NT - 2ST - 2WT - WN + 5T + W + S - N - R$.

3.2. Activation functions

Each unit in the network 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

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

[9] The weighted sum of the inputs minus the threshold is actually the partial derivative $-dE/dX_i$, where E is the energy function.

[10] In the stochastic models, noise is introduced and the energy may not be decreasing all the times.

The sum is then used as the input for some activation function F (usually nonlinear and nondecreasing)

$$X_i = F(net_i),$$

whose task is to change the activation value according to the energy steepness. Different connectionist models may have different activation functions. 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 subsections.

3.2.1. The discrete Hopfield model

The discrete Hopfield model [20] uses binary-valued units whose activation values are either zero or one. The activation 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;[11] however, many times this local minimum will be a shallow one, and the network will not be able to escape to a deeper minimum.

3.2.2. The analog model of Hopfield and Tank

In Hopfield and Tank networks [21] 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 several problems. Hopfield and Tank use an analog circuit for their implementation and, therefore the energy function had to be modified slightly:

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

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

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

and g^{-1} is the inverse of g. At high (infinite) gain the minima lie at the corners of the search space in the same locations as those of the discrete Hopfield model. The discrete and analog models collapse, therefore, into one at infinite gain.

[11] Fast on average but exponential in worst case [24].

3.2.3. Boltzmann machines

The Boltzmann machine [17] has binary units as in the discrete Hopfield model. The important difference is that the activation 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. A Boltzmann machine can theoretically be run long enough to guarantee that a global minimum is found [9]; 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 an "any-time" answer. The system is not guaranteed to find a global minimum at the end of such time quota; however, as more time is given, deeper minima may be found and the probability of finding a global solution increases.

3.2.4. Deterministic Boltzmann machines—mean field theory

A mean field method suggested by Peterson and Hartman [38] 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:

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

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 a desired any-time property.

3.2.5. Heuristic repair methods

Recently, repair methods based on local search have been proposed for NP-hard search problems. The techniques were shown to be successful for large scale (hard) distributions of problems such as constraint satisfaction, n-queen, scheduling and 3-SAT [33,57].

In these methods, the distance between the current state and a goal is measured and the function is being minimized using local search. Each of the variables of the problem

is checked for the effect of changing its value on the distance function. Usually, the change that reduces the distance the most is executed. When the energy function is taken as the distance function, heuristic repair may be considered as a sequential variation of the connectionist algorithm implemented in symmetric networks.

Heuristic repair methods can be used therefore to implement the formalism proposed in this article. The distance function is the high-order energy function (weighted sum of violated constraints) and the problem variables are the nodes of the networks (atomic propositions).

3.3. High-order energy functions

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

High-order connectionist networks have sigma-pi units [55] with multiplicative connections. Symmetric networks can be easily extended to handle high-order connections. Naturally, such networks may be viewed as minimizing high-order energy functions [56].

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

$$E^k(x_1,\ldots,x_n) = \sum_{1\leqslant i_1<i_2<\cdots<i_k\leqslant n} -w_{i_1,\ldots,i_k}X_{i_1}\cdots X_{i_k} + \sum_{1\leqslant i_1<\cdots<i_{k-1}\leqslant n} -w_{i_1,\ldots,i_{k-1}}X_{i_1}\cdots X_{i_{k-1}} + \cdots + \sum_{1\leqslant i\leqslant n}^{1} -w_iX_i.$$

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

$$\sum_{1\leqslant i<j\leqslant n} -w_{ij}X_iX_j + \sum_{i\leqslant n} -w_iX_i.$$

In the high-order model each node is assigned a sigma-pi unit that updates its activation value by first computing the partial derivative of the energy function and then update the activation value accordingly:

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

$$a_i = F(net_i),$$

where $a_i = F(net_i)$ is the standard update rule that is unique to the model we wish to extend. In the discrete Hopfield model for example, $F(net_i) = 1$ if $net_i > 0$, and

$F(net_i) = 0$ otherwise. A high-order network (see Fig. 3) is a hyper-graph, where k-order terms are translated into hyper-arcs connecting k nodes. The arcs are not directed (the weight is the same for every node that is part of the arc) and the weight of an arc is determined by the weight of the corresponding term in the energy function (with an opposite sign). As in the quadratic case, there is a translation back and forth between k-order energy functions and symmetric high-order networks with k-order sigma-pi units.

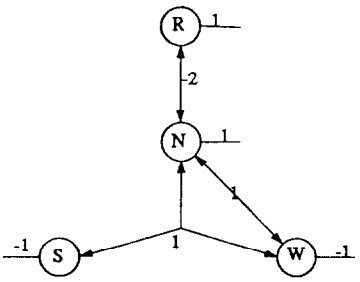

Fig. 3. A cubic network that represents $E = -NSW + 2RN - WN + W + S - R - N$ using sigma-pi units and a cubic hyper-arc (it is equivalent to the network of Fig. 2, without the hidden unit T).

We can arbitrarily divide the variables of an energy function into two sets: visible variables and hidden variables. The hidden variables correspond to the hidden units of the network, and the visible variables correspond to the visible units. An energy function with both hidden and visible variables is denoted usually as a function $E(\vec{x}, \vec{t})$, where $\vec{x}$ represents the visible variables and $\vec{t}$ represents the hidden variables.

An assignment of zeros and ones to the visible variables is called a visible state. The values of the visible units after an equilibrium is reached, are considered as the "answer" of the network.

Later in this article, I'll interpret visible states as truth assignments: the visible variables are viewed as atomic propositions: "1" is interpreted as "true" and "0" is interpreted as "false".

We call the set of minimizing vectors projected onto the visible variables, "the visible solutions" of the minimization problem; i.e.,

$$\{\bar{x} \mid (\exists \bar{t}) E(\bar{x}, \bar{t}) = \min_{\bar{y},\bar{z}} \{E(\bar{y}, \bar{z})\}\}.$$

Models like Boltzmann machines, harmony theory, mean field theory, may be viewed as searching for a *global minimum*[12] of the corresponding energy functions. Local minima or spurious memories may exist. In general however, local minima are considered to be undesirable phenomena, and cause a degradation in the performance of the network. This article will ignore local minima that are not global, and usually they will not represent any meaningful knowledge.

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

Definition 3.1. Let E be a symmetric network with energy function $E(\vec{x}, \vec{t})$, where $\vec{t}$ designates the hidden variables. The *characteristic* function of the network is the function: $Erank_E(\vec{x}) = \min_{\vec{y}}\{E(\vec{x}, \vec{y})\}$.

The $Erank_E$ function defines the energy of all visible states. The energy of a visible state is the energy level obtained when the visible units are clamped with the state values, and the hidden units are free to settle so that a minimum is reached. This $Erank_E$ function characterizes the network's behavior: it is independent of the hidden units and it is also independent of the exact topology of the original network. There may be many possible networks with the same characteristic function. The next section uses the characteristic function to show equivalence between different networks.

3.4. The equivalence between high-order networks and low-order networks

The following subsection is a review of results reported in [40].

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

I'll show now an algorithm to convert any high-order network into a strongly equivalent low-order one, with additional hidden units. In addition, any energy function with hidden variables can be converted into a strongly equivalent, (possibly) higher-order network by eliminating some or all of the hidden units. These algorithms allow us to trade the computational power of sigma-pi units for additional simple units and vice versa. As a result we'll see that the expressive power of high-order networks is the same as that of low-order networks with hidden units.

Readers who are not interested in the technical details of the constructions may skip now to the next subsection. They may keep in mind only that the constructions for both directions are possible.

Theorem 3.2.

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

$$w\prod_{i=1}^{k-1} X_i - \left(\sum_{i=1}^{k-1} 2wX_iT\right) + 2wX_kT + (2k-3)wT,$$

generating a strongly equivalent energy function of order $k-1$ *with one additional hidden variable T.*

The proof appears in [44].

Example 3.3. The following is a 4-order energy function with a 4-order term $XYZU$. It can be converted into a quadratic energy function using two additional hidden variables T and T'.

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

The symmetric transformation, from low-order into high-order functions by eliminating any subset of the variables, is also possible (of course we are interesting in eliminating only hidden variables).

To eliminate T, bring the energy function to the form: $E = E' + oldterm$, where $oldterm = (\sum_{j=1}^{k} w_j \prod_{i=1}^{l_j} X_{j_i})T$.

Consider all assignments S for the variables ($\hat{X} = x_{i_1} \cdots x_{i_l}$) in *oldterm* (not including T), such that $\beta_S = \sum_{j=1}^{k} w_j \prod_{i=1}^{l_j} x_{j_i} < 0$.

Let

$$L_S^j = \begin{cases} \text{“}X_{i_j}\text{”}, & \text{if } S(X_{i_j}) = 1, \\ \text{“}(1 - X_{i_j})\text{”}, & \text{if } S(X_{i_j}) = 0, \end{cases}$$

it is the expression “X_i” or “$(1 - X_i)$” depending whether the variable is assigned 1 or 0 in S. The expression $\prod_{j=1}^{l} L_S^j$ therefore determines the state S, and the expression

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

represents the disjunction of all the states that cause a reduction in the total energy. The new function $E' + newterm$, is therefore equivalent to $E' + oldterm$ and does not include T.

Example 3.4.

Let T be the hidden variable to be eliminated, then:

$$AB + TAC - TA + 2TB - T = AB + T(AC - A + 2B - 1).$$

The following assignments for (A, B, C) cause β to be less then zero:

$$\begin{aligned}
\beta_{(0,0,0)} &= -1, & \beta_{(0,0,1)} &= -1, \\
\beta_{(1,0,0)} &= -2, & \beta_{(1,0,1)} &= -1.
\end{aligned}$$

The new term equals:

$$\begin{aligned}-(1-A)(1-B)(1-C) &- (1-A)(1-B)C \\ &- 2A(1-B)(1-C) - A(1-B)C \\ = -ABC + AB + AC - A + B - 1.&\end{aligned}$$

Therefore:

$$AB + TAC - TA + 2TB - T \approx -ABC + 2AB + AC - A + B.$$

4. The equivalence between penalty logic and energy minimization

This section defines equivalence between different forms of knowledge representation (that use ranked-models semantics), and use this definition to show the relationships between penalty logic and SCNs.

4.1. Reasoning with ranking functions

A ranking function over a set of models is a function that assigns a real value (rank) to every model in the set. The ranking of a model may be considered as a grade for the "normality" or the "goodness" of the model.

As we saw in previous subsection, every SCN E is characterized by the ranking function $Erank_E$. Similarly, every ranking function is equal to some high-order energy function and therefore characterizes some SCN.[13] The search performed by the SCN for a global minimum may be viewed thus as a search for a model that minimizes the ranking function.

Penalty logic formulas, classical logic WFFs, and SCNs may be interpreted as representations of ranking functions. It may be useful therefore to define our reasoning mechanism independently of the knowledge representation form:

Definition 4.1. Let $\mathcal{W} = \{0,1\}^n$ be the set of models defined over a set of n atomic propositions. A *ranking function* $k : \mathcal{W} \rightarrow \mathcal{R}$ is a function that maps models into reals. A ranking function k is *strict* iff the domain of k is $\{0, \infty\}$ (where ∞ represents a large positive number). A *preferred model* $\vec{x}$ of a ranking function k is a model that minimizes k; i.e., $k(\vec{x}) = \min_{\vec{y}}\{k(\vec{y})\}$.

The set of preferred models of k is denoted Γ_k.

Definition 4.2. Let f, k, e be ranking functions. f is entailed from k ($k \models f$) iff $\Gamma_k \subseteq \Gamma_f$. f is entailed from the background knowledge k using the evidence e ($e \models^k f$) iff $k + e \models f$.

The *consequence relation* induced by k is the set of all pairs $\{\langle e, f\rangle \mid e \models^{\psi} f\}$.

The model-based reasoning mechanism defined for penalty logic in Section 2 is consistent with the above definitions if $Vrank_{\psi}$ is taken as the ranking function.

[13] There is no guarantee however, that the the size of the network that represents an arbitrary ranking function is polynomial in n (the number of visible variables).

4.2. Calculi to describe ranking functions

Our next step is to describe symbolically the knowledge that is encapsulated in a ranking function. This subsection defines several languages for describing ranking function and shows their equivalence. Sentences of such languages are interpreted using ranked-models semantics, and transformations are allowed from one knowledge representation into another if some basic properties are preserved.

The following definitions establish the relationship between a form of knowledge representation and its meaning.

Definition 4.3. A calculus is a triple $\langle \mathcal{L}, m(\,), M \rangle$, where $\mathcal{L}$ is a language, M is a set of possible models and $m : \mathcal{L} \rightarrow \{k \mid k \text{ is a ranking function}\}$ is a function that assigns a ranking function for each sentence of the language $\mathcal{L}$. The function $m(s)$ is called the *interpretation* of the sentence s. Let $s, s', e, k \in \mathcal{L}$; a model $\vec{x}$ is a *preferred model* of s ($\vec{x} \models s$) iff $\vec{x}$ is a preferred model of the ranking function $m(s)$. A sentence s *entails* sentence s' ($s \models s'$) if the ranking function $m(s)$ entails the ranking function $m(s')$. Similarly, a combination of a background sentence with an evidence sentence is interpreted as the addition of their corresponding ranking functions; i.e., $e \models^k s$ iff $(m(e) + m(k)) \models m(s)$.

The consequence relation of k is the set of all pairs $\{\langle e, s \rangle \mid e \models^k s\}$.

Both classic predicate logic and propositional logic can be viewed as calculi whose languages describe strict ranking functions.

Example 4.4. Propositional calculus is $\langle \mathcal{L}, m(\,), \{0,1\}^n \rangle$, where $\mathcal{L}$ is the language of propositional well formed formulas (WFFs) and $m(s)$ outputs the function $(\infty(1 - H_s(\vec{x})))$, given a formula s (∞ represents a large positive real). $H_s(\vec{X})$ is the characteristic function of the WFFs and is recursively defined as:

$$H_s(\vec{X}) = \begin{cases} X_i, & \text{if } s = X_i \text{ is an atomic proposition,} \\ 1 - H_{s'}(\vec{X}), & \text{if } s = \neg s', \\ H_{s_1}(\vec{X}) \times H_{s_2}(\vec{X}), & \text{if } s = s_1 \wedge s_2, \\ H_{s_1}(\vec{X}) + H_{s_2}(\vec{X}) - H_{s_1}(\vec{X}) \times H_{s_2}(\vec{X}), & \\ & \text{if } s = s_1 \vee s_2. \end{cases}$$

The reader may easily observe, that any propositional WFF describes a strict ranking function that returns 0 for truth assignments that satisfy the WFF, and ∞ for assignments that do not satisfy it.

Example 4.5. Penalty logic is a calculus $\langle \mathcal{L}_p, m, \{0,1\}^n \rangle$ such that $m(\psi) = Vrank_\psi$.

Definition 4.6. Let $s \in \mathcal{L}_1$ and $s' \in \mathcal{L}_2$ be sentences of two (possibly different) calculi $\langle \mathcal{L}, m, M \rangle$ and $\langle \mathcal{L}', m', M \rangle$; we define three kinds of equivalence relations between them:

(1) s is *strongly equivalent* to s' ($s \approx^{s} s'$) iff their corresponding ranking functions are equal, up to a constant difference; i.e., $m(s) = m'(s') + c$. We call this equivalence "magnitude preserving" or s-equivalence.

(2) s is p-equivalent to s' ($s \approx^{p} s'$) iff their associated ranking functions induce the same ordering over the set of models; i.e., $\forall \vec{x}, \vec{y}, m(s)(\vec{x}) < m(s)(\vec{y})$ iff $m'(s')(\vec{x}) < m'(s')(\vec{y})$. We call this equivalence "preference preserving" or p-equivalence.

(3) s is weakly equivalent to s' ($s \approx^{w} s'$) iff their corresponding ranking functions have the same sets of satisfying models; i.e., $\Gamma_{m(s)} = \Gamma_{m'(s')}$. We call this equivalence "minima preserving" or w-equivalence.

Observations.

(1) If two background sentences are strongly equivalent, then for any given evidence e, the two corresponding sentences entail the same set of conclusions; i.e., if $s \approx^{s} s'$ then for every evidence e and every conclusion c, $(m(s) + m(e)) \models m(c)$ iff $(m'(s') + m'(e)) \models m'(c)$. Therefore, two strongly equivalent sentences have the same induced consequence relation. i.e., In addition, the probabilistic meaning that sometimes is associated with the ranking function is preserved (e.g., Boltzmann machine, [5]), since

$$\frac{P(\vec{x})}{P(\vec{y})} = e^{(m(s)(\vec{x}) - m(s)(\vec{y}))} = e^{((m'(s')(\vec{x}) + c) - (m'(s')(\vec{y}) + c))}.$$

(2) If two background sentences are p-equivalent, then for every *strict* evidence e, the two sentences entail the same set of conclusions; i.e., if $dom(e) = \{0, \infty\}$ and $s \approx^{p} s'$, then for every conclusion c, $(m(s) + e) \models m(c)$ iff $(m'(s') + e) \models m'(c)$. We can't guarantee this property for any non-strict evidence.

(3) If two sentences s, s' are weakly equivalent, then the two sentences entail the same set of direct conclusions; i.e., $m(s) \models m(c)$ iff $m'(s') \models m'(c)$. We can't guarantee this property to hold once we try to add evidence.

The reader may easily observe that if two sentences are strongly equivalent then they are also p-equivalent and if they are p-equivalent they are also weakly equivalent.

If all we want is to preserve the set of conclusions achievable from a piece of knowledge, we may use transformations which only preserve the minima (weak equivalence). If however, we would like to be able to combine strict evidence to our transformed knowledge, we need to perform "preference preserving" transformations. We need "magnitude preserving" transformations (strong equivalence) if we want to combine *any* evidence or give probabilistic interpretation to our transformed knowledge. Most of our transformations in the reminder of this paper are "magnitude preserving" (strongly equivalent). Strong equivalence of two forms of knowledge representation means that the ranking functions that are induced by either these representations are the same (up to a constant difference).

We define now an equivalence between two calculi.

Definition 4.7. A calculus $C_1 = \langle \mathcal{L}, \{0,1\}^n, m \rangle$ is (s-/p-/w-) equivalent to a calculus

$\mathcal{C}' = \langle \mathcal{L}', \{0,1\}^n, m' \rangle$ iff for every $s \in \mathcal{L}$ there exists an (s-/p-/w-) equivalent $s' \in \mathcal{L}'$ and for every $s' \in \mathcal{L}'$ there exists an (s-/p-/w-) equivalent $s \in \mathcal{L}$.

We thus can use the language $\mathcal{L}$ to represent every ranking function that is representable using the language $\mathcal{L}'$, and vice versa. In the sections to come, I shall present several equivalent calculi and show that all of them describe the knowledge embedded in SCNs.

4.3. Some examples of equivalent calculi

Example 4.8 (*The calculus of energy functions*). The algebraic notation that was used to describe energy functions as sum-of-products can be viewed as a language for describing ranking functions. The *calculus of energy functions* is therefore $\langle \{E\}, \{0,1\}^n, m() \rangle$, where $\{E\}$ is the set of all strings representing energy functions written as sum-of-products, and $m(E) = Erank_E$. Two special cases are of particular interest: the calculus of quadratic functions and the calculus of high-order energy functions with no hidden variables.

In Section 3.4, algorithms were given that (1) convert high-order energy functions to strongly equivalent[14] low-order ones with additional hidden variables, and (2) convert energy functions with hidden variables into strongly equivalent (possibly) higher-order ones without those hidden variables. We may therefore conclude that the calculus of high-order energy functions with no hidden units is strongly equivalent to the calculus of quadratic functions. Thus, we can use the language of high-order energy functions with no hidden units to describe any symmetric connectionist network (SCN) with arbitrary number of hidden units and vice versa. Note also that the calculus of SCNs, whose language describes graphs, weights and thresholds, is of course also strongly equivalent to the calculus of quadratic energy functions.

Example 4.9 (*Propositional calculus*). In [40], I showed that the satisfiability of propositional calculus is equivalent to quadratic energy minimization. I claim that this is a weak equivalence. The energy function E_φ is obtained from φ using the following algorithm:

(1) Convert the WFF into a conjunction of subformulas, each of at most three variables.[15] This is done by adding additional hidden atomic propositions, and "naming" binary subexpressions of the formula using the new propositions. For example, $(((A \vee B) \vee \neg C) \rightarrow (D \vee E))$ is converted into $(T_1 \leftrightarrow A \vee B) \wedge (T_2 \leftrightarrow T_1 \vee \neg C) \wedge (T_2 \rightarrow D \vee E)$.

(2) Assuming the result is of the form $\bigwedge_j \varphi_{i_j}$, the energy function is computed to be $\sum_j H_{\neg\varphi_{i_j}}$, where H_φ is the characteristic function defined in Example 4.4.

(3) Convert the cubic terms in the result to quadratic ones using a high-order to low-order procedure of Section 3.4.

[14] In these papers we were concerned only with weak equivalence, but it is easily shown that strong equivalence holds.

[15] In contrast to the familiar 3-SAT, connectives in a subformula are not limited to disjunctions of literals.

The global minima of the energy function are exactly equal to the satisfying models of the WFF. Propositional calculus is therefore weakly equivalent to the calculus of quadratic energy functions and can be used as a high-level language to describe SCNs. However, two limitations exist: (1) the algorithm (in [40]) that converts an energy function to a satisfiable WFF may generate an exponentially long WFF; and (2) the equivalence is *weak*. It means that although the WFF and the energy function have the same set of satisfying models, neither evidence can be added nor the probabilistic interpretation is preserved.

4.4. The equivalence of penalty logic and SCNs

This section shows that penalty logic and SCNs are strongly equivalent: Every penalty logic formula can be represented *efficiently* in an SCN and every SCN can be described *efficiently* by a penalty logic formula.

When a PLOFF ψ is strongly equivalent to a network described by an energy function E then:

(1) The set of global minima of E is equal exactly to the set of the preferred models of φ.

(2) Both knowledge representations induce the same order on the possible models; i.e., s is "better" than s' iff $Erank_E(s) < Erank_E(s')$ iff $Vrank_\psi(s) < Vrank_\psi(s')$.

(3) Knowledge update is cumulative. An addition (deletion) to the knowledge base can be done by merging (subtracting) the new PLOFF with the existing one. The equivalent operation in the energy space is adding the energy terms of the new PLOFF to the energy function representing the old one. The update of a network with a new piece of knowledge is therefore modular and simple.

4.4.1. Representing penalty logic using SCNs

Theorem 4.10. *For every PLOFF* $\psi = \{\langle \rho_i, \varphi_i \rangle \mid i = 1, \ldots, n\}$ *there exists a strongly equivalent quadratic energy function* $E(\vec{x}, \vec{t})$*; i.e., there exist a constant* c *such that* $Vrank_\psi = Erank_E + c$*. The size of the network that is generated by* E *is of the same order as the length of* ψ*; i.e., the number of symbols in* ψ*.*

Construction. We can construct E from ψ using the following procedure:

(1) Start with an empty set of assumptions ψ'. For every pair $\langle \rho_i, \varphi \rangle$ in ψ, create a new hidden variable T_i, "name" φ_i using $T_i \leftrightarrow \varphi_i$ and add the pairs $\langle \infty, T_i \leftrightarrow \varphi_i \rangle$ and $\langle \rho_i, T_i \rangle$ into ψ'. The penalty ∞ represents a real value that is large enough to *force* the "naming" constraint to be satisfied. The original penalty ρ_i causes the T_i's to compete with each other; while the high penalty ∞ guarantees that if T_i holds (among the winning theories) then φ_i also holds. ψ' is therefore strongly equivalent to ψ and the T_i's may be considered as hidden variables.

(2) Construct the energy function $\sum_i \infty E_{T_i \leftrightarrow \varphi_i} - \sum_j \rho_j T_j$, where E_φ is the function generated by the algorithm described in Example 4.9.

Proof. To show $Vrank_{\psi} = Erank_E + c$: If the hidden units T_i in E are free to settle to a minimum, then for any clamping of the visible variables, $E_{\infty T_i \leftrightarrow \varphi_i}$ always obtains the minimum value c_i of this function by setting T_i to true if φ_i is satisfied, and to false if φ_i is violated. Therefore,

$$\begin{aligned} Erank_E(\vec{x}) &= \sum_{i=1}^{n} c_i - \sum_{\vec{x} \models \varphi_i} \rho_i = \sum_{i=1}^{n} c_i - \sum_{i=1}^{n} \rho_i + \sum_{\neg(\vec{x} \models \varphi_i)} \rho_i \\ &= \sum_{\neg(\vec{x} \models \varphi_i)} \rho_i + c = Vrank_{\psi} + c. \quad \square \end{aligned}$$

The "naming" of the first step is needed only if the number of variables in an assumption φ_i is greater than three. If this is the case and we do not "name" φ_i, then the second step of the algorithm might generate more then one "triple". Each triple will have a penalty that will contribute to the energy function independently of the other triples, and the constraint as a whole will not have the atomicity we expect. Thus, the ranking function that will be generated will not be the one we wished. The high penalty we use for the "naming" causes the system to *always* find solutions that satisfy the "naming" constraints. Once we guarantee that all the "naming" constraints are satisfied, all that is needed is to make the T_i's compete as if they were the original assumptions. When the number of variables is less or equal to three, the way we construct the energy function guarantees that only one triple is generated. Thus, either the constraint is satisfied as a whole (with zero penalty) or it is not satisfied (and the penalty is ρ_i); i.e., the splitting of one constraint into more then one "triple" does not happen, and the atomicity is preserved.

The network that is generated can be seen as performing a search for a preferred model of ψ. According to the sound and complete proof theory, it can also be seen as searching for a preferred theory of ψ; i.e., the T_i's that win the competition correspond to the assumptions in the preferred theory found.

In the following example the assumptions have less than four variables, thus "naming" is not needed.

Example 4.11 (*The Nixon diamond case of Example 2.2*). The PLOFF that is to be converted is:

$$\psi = \{\langle 3000, N\rangle, \langle 1000, N \rightarrow Q\rangle, \langle 1000, N \rightarrow R\rangle, \langle 10, Q \rightarrow P\rangle, \langle 10, R \rightarrow \neg P\rangle\}.$$

No "naming" is needed, so $\psi' = \psi$. Each of the pairs is converted to an energy function:

$$\begin{aligned} &1000 \quad N \rightarrow R \qquad 1000(E_{\neg N \vee R}) = 1000(N - NR), \\ &1000 \quad N \rightarrow Q \qquad 1000(E_{\neg N \vee Q}) = 1000(N - NQ), \\ &10 \quad R \rightarrow \neg P \qquad 10(E_{\neg R \vee \neg P}) = 10(RP), \\ &10 \quad Q \rightarrow P \qquad 10(E_{\neg Q \vee P}) = 10(Q - QP), \\ &3000 \quad N \qquad 3000(E_N) = 3000(-N). \end{aligned}$$

Summing the energy terms together:

$$E = -1000NQ - 1000NR + 10RP - 10QP - 1000N + 10Q.$$

The corresponding network appears in Fig. 4.

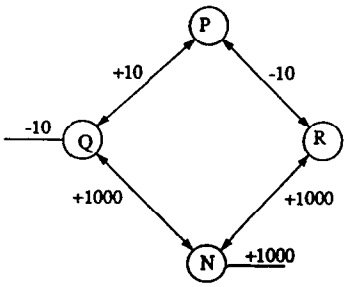

Fig. 4. The network that represents the Nixon diamond example. It corresponds to the energy function: $E = -1000NQ - 1000NR + 10RP - 10QP - 1000N + 10Q$.

Example 4.12. Converting the "meeting" example, we first show in Table 1 the general case with "naming" (it is used for demonstration purposes only, since the assumptions have less than four variables).

The energy function we get by summing the energy of the assumptions is:

$$1000T_2SM - 1000T_3CM - 1000T_4CS - 2000T_1M + 1000T_2S + 1000T_2M$$
$$+2000T_3C - 1000T_3M + 2000T_4C - 1000T_4S + 1000M - 2000C + 999T_1$$
$$-2010T_2 - 1100T_3 - 2000T_4.$$

It is shown as a cubic symmetric network in Fig. 5(a) and as a quadratic network in Fig. 5(b). Since the assumptions in our example have less than three variables each, we can generate a simpler (strongly equivalent) network from the energy function of $\sum_i \rho_i E_{\neg\varphi_i} = 1(-M) + 100(C - CM) + 1000(C - CS)$ (see Fig. 5(c)).

Table 1
Example 4.12: general case

Penalty	WFF	$E_{\varphi_i}(\bar{x})$
1000	$T_1 \leftrightarrow$ meeting	$1000(T_1 - 2T_1M + M)$
1000	$T_2 \leftrightarrow$ (sick $\rightarrow$ ($\neg$ meeting))	$1000(T_2SM - 2T_2 - S - M + T_2S + T_2M)$
1000	$T_3 \leftrightarrow$ (cold-only $\rightarrow$ meeting)	$1000(-T_3 - C + 2T_3C + M - T_3M - T_3CM)$
1000	$T_4 \leftrightarrow$ (cold-only $\rightarrow$ sick)	$1000(-T_4 - C + 2T_4C + S - T_4S - T_4CS)$
1	T_1	$-1T_1$
10	T_2	$-10T_2$
100	T_3	$-100T_3$
1000	T_4	$-1000T_4$

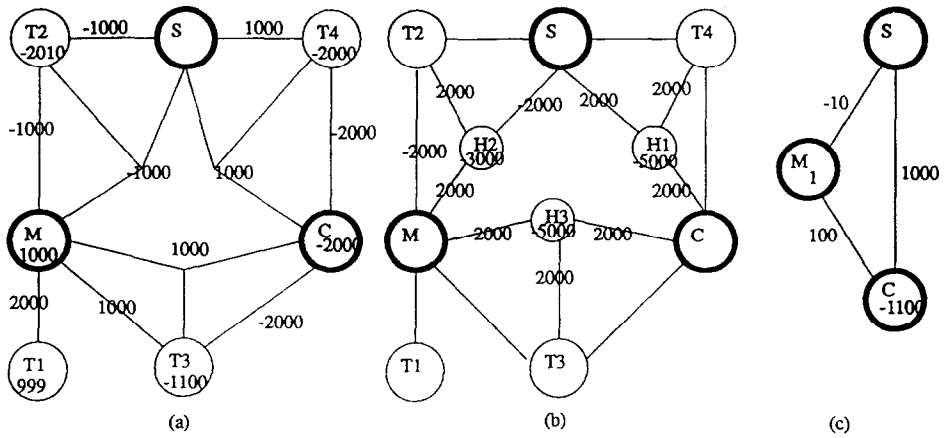

Fig. 5. Equivalent symmetric networks for the meeting example (the numbers in the circles are thresholds): (a) cubic; (b) quadratic; and (c) quadratic for the simple conversion (no naming).

Once it is possible to generate a network that searches for preferred models (or preferred theories), it is possible to construct a network that will reason according to our definition of entailment. A construction of such network is described in Section 5.

4.4.2. Representing SCNs as penalty logic formulas

This subsection shows that it is possible to describe efficiently any network by a penalty logic formula. The motivation here is to demonstrate that penalty logic is an efficient and compact language for specification of symmetric connectionist networks.

Theorem 4.13. *Every energy function E is strongly equivalent to some PLOFF ψ; i.e., there exists a constant c such that $Erank_E = Vrank_\psi + c$.*

Construction. The following algorithm generates a strongly equivalent PLOFF from an energy function:

(1) Eliminate hidden variables (if any) from the energy function, using the algorithm of Section 3.4.

(2) The energy function (with no hidden variables) is now brought into a sum-of-products form and is converted into a PLOFF in the following way: Let $E(\vec{x}) = \sum_{j=1}^{m} w_j \prod_{n=1}^{k_j} x_{j_n}$ be the energy function. We construct a PLOFF

$$\psi = \left\{ \left\langle -w_i, \bigwedge_{n=1}^{k_i} x_{i_n} \right\rangle \mid w_i < 0 \right\} \cup \left\{ \left\langle w_l, \neg \bigwedge_{n=1}^{k_l} x_{l_n} \right\rangle \mid w_l > 0 \right\}.$$

The formula that is generated is strongly equivalent to the original energy function (network). The size of the formula is in the order of the size of the original network (linear in the number of connections).

Proof. To show $Vrank_\psi = Erank_E + c$:

$$\begin{aligned} Vrank_\psi(\vec{x}) &= \sum_{w_i<0 \wedge \neg(\vec{x} \models \bigwedge X_{i_n})} -w_i + \sum_{w_l>0 \wedge \neg(\vec{x} \models \neg(\bigwedge X_{l_n}))} w_l \\ &= -\sum_{w_i<0 \wedge \vec{x} \models \neg(\bigwedge X_{i_n})} w_i + \sum_{w_l>0 \wedge \vec{x} \models (\bigwedge X_{l_n})} w_l \\ &= -\sum_{w_i<0} w_i + \sum_{w_i<0 \wedge \vec{x} \models \bigwedge X_{i_n}} w_i + \sum_{w_l>0 \wedge \vec{x} \models (\bigwedge X_{l_n})} w_l \\ &= \sum_{w_i<0} w_i \prod_n X_{i_n} + \sum_{w_l>0} w_l \prod_n X_{l_n} - \sum_{w_i<0} w_i \\ &= Erank_E + c. \quad \square \end{aligned}$$

Example 4.14. Looking at the network of Fig. 4, we would like to describe this network as a PLOFF. The energy function is:

$$E = -1000NQ - 1000NR + 10RP - 10QP - 1000N + 10Q.$$

The negative terms are:

$$\langle 1000, N \wedge Q \rangle, \quad \langle 1000, N \wedge R \rangle, \quad \langle 10, Q \wedge P \rangle, \quad \langle 1000, N \rangle.$$

The positive terms are:

$$\langle 10, \neg R \vee \neg P \rangle, \quad \langle 10, \neg Q \rangle.$$

The final PLOFF is therefore:

$$\langle 1000, N \wedge Q \rangle, \quad \langle 1000, N \wedge R \rangle, \quad \langle 10, Q \wedge P \rangle,$$
$$\langle 1000, N \rangle, \quad \langle 10, \neg R \vee \neg P \rangle, \quad \langle 10, \neg Q \rangle.$$

Note that as it is usually the case with reverse-compilation, the formula we get is not very meaningful; however, it is clear that a compact description exists for every network.

5. A connectionist inference engine

Suppose a background PLOFF ψ, an evidence PLOFF e, and a query which is a (strict) standard logic WFF φ. We would like to construct a connectionist network to answer one of the possible three answers: (1) $\psi \cup e \models \varphi$; (2) $\psi \cup e \models (\neg\varphi)$; or (3) both $\psi \not\models \varphi$ and $\psi \not\models (\neg\varphi)$ ("ambiguous").

Intuitively, our connectionist engine is built from two subnetworks, each of which is trying to find a satisfying model for $\psi \overset{*}{\cup} e$. The first subnetwork is biased to search for a preferred model which satisfies also φ, whereas the second subnetwork is biased to search for a preferred model which satisfies $\neg\varphi$. If two such models exist, then we conclude that φ is "ambiguous" ($\psi \overset{*}{\cup} e$ entails neither φ nor $\neg\varphi$). If no preferred model also satisfies φ, we conclude that $\psi \cup e \models \neg\varphi$, and if no model also satisfies $\neg\varphi$, we

Table 2

ψ	searches for a preferred model of ψ that satisfies also P
$\bigcup\psi'$	searches for a preferred model of ψ that satisfies also $\neg P$
$\bigcup\{\langle\varepsilon, (QUERY_P \rightarrow P)\rangle\}$	bias ψ to search for a model that satisfies P
$\bigcup\{\langle\varepsilon, (QUERY_P \rightarrow (\neg P'))\rangle\}$	bias ψ' to search for a model that satisfies $(\neg P')$
$\bigcup\{\langle\varepsilon, (P \wedge \neg P') \rightarrow AMBIGUOUS_P\rangle\}$	if two satisfying models exist that do not agree on P, we conclude *"AMBIGUOUS"*
$\bigcup\{\langle\varepsilon, (P \leftrightarrow P') \rightarrow (\neg AMBIGUOUS_P)\rangle\}$	if despite the bias we are unable to find two such satisfying models we conclude NOT ambiguous.

conclude that $\psi \cup e \models \varphi$. For simplicity let us first assume that the evidence e is a strict conjunction of literals (atomic propositions or their negation) and that φ is a single atomic proposition. Later we'll describe a general solution.

To implement this intuition we first need to duplicate our background knowledge ψ and create its copy ψ' by naming all the atomic propositions A using A'. For each atomic proposition P that might participate in a query, we then add two more propositions: *"$QUERY_P$"* and *"$AMBIGUOUS_P$"*. $QUERY_P$ is used to initiate a query P; it will be externally clamped by the user, when he or she inquires about P. The unit *"$AMBIGUOUS_P$"* represents the answer of the system. It will be set to TRUE if we can conclude neither that ψ entails P nor that ψ entails $\neg P$.

Our inference engine can be therefore described (in the language of penalty logic) as in Table 2

Using the algorithm of Theorem 4.10, we generate the corresponding network. The network that is generated for the Nixon example is shown in Fig. 6.

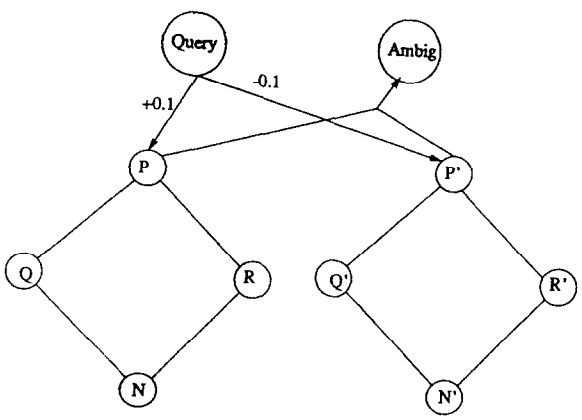

Fig. 6. Inference network for the Nixon diamond case: the two rings represents two similar subnetwork: One searches for a preferred model that satisfies the query and the other searches for a preferred model the falsifies the query.

To initiate a query about P the user externally clamps the unit $QUERY_P$. This causes a small positive bias ε to be sent to unit P and a small negative bias $-\varepsilon$ to be sent to P'. Each of the two subnetworks ψ and ψ', searches for a global minimum (a satisfying model) of the original PLOFF. The bias (ε) is small enough so it does not introduce new global minima for each of the subnetworks. It may however, constrain the set of

global minima. If a satisfying model that also satisfies the bias exists, then this model is in the new set of global minima of ψ. The new set of global minima is the set of all preferred models of ψ that also satisfy the query. If no preferred model also satisfies the query then the set of global minima is unaffected by the bias and the network searches for one of those models.

The network therefore tries to find models that satisfy also the bias rules. If it succeeds, we conclude *"AMBIGUOUS"*, otherwise we conclude that all the satisfying models agree on the same truth value for the query. The *"AMBIGUOUS"* proposition is then set to "false", and the answer whether $\psi \models \varphi$ or whether $\psi \models \neg\varphi$ can be found in the unit P. If P is "true" then the answer is $\psi \models \varphi$ since P holds in all satisfying models. Similarly, if P is false, we conclude that $\psi \models \neg\varphi$.

When the evidence is a strict conjunction of literals, the user may add it to the background network simply by clamping the appropriate atomic propositions whenever a new evidence is observed. In the general case we need to combine an arbitrary evidence e and an arbitrary query φ: We do this by building an inference network for $\psi \overset{*}{\cup} e \cup \{\langle\infty, P \leftrightarrow \varphi\rangle\}$ and by querying about P, a new atomic proposition.

The network that was generated converges to the correct answer if it manages to find a global minimum. An annealing schedule[16] like in [17] may be used for such search. A slow enough annealing is certain to find a global minimum and therefore the correct answer, but it might take exponential time. Since the problem is NP-hard, we will probably not find an algorithm that will always give us the correct answer in polynomial time. However, attempts to accelerate the search for special instances of the problem are continuously being made (see for example [45]). Traditionally in AI, knowledge representation systems trades the expressiveness of the language they use with the time complexity they allow [28],[17] and the accuracy of the answer is usually not sacrificed. The inference mechanism described in this section, as in [5], trades the time resources with the accuracy of the answer. Only limited time resources are given, and we wish to stop the search when this limit is reached. The annealing schedule can be planned to fit the time limitation, and an answer is always given at the end of the process. Although the answer may be incorrect, the system is able to improve its guess as more time resources are given.

6. Learning propositional 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 SCNs can learn unknown propositional formulas inductively, and develop incrementally a representation that is equal to the ones constructed by compiling 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

[16] There are also other techniques for improving the chances to escape from local minima [15,21].

[17] Connectionist systems like [59] and [19] trade expressiveness with time complexity.

learn is a satisfiable WFF. The task of the network is to update its weights in such a way that at the end of the learning process the energy function is equal to the one obtained by translating φ into E_φ. Clearly, by doing so, the set of global minima of the energy function is equal to the set of satisfying models of φ (Γ_φ) which is the training set.

We may look at the process as learning of associative memories: Given a set of vectors to be stored as memories, 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. [18]

The algorithm that will be described uses high-order units (hyper-arcs); however, the reader should remember that it is always possible to convert the hyper-arcs into pairwise connections by adding hidden units (see Section 3.4).

Definition 6.1. A k-CNF is a WFF that is formed as a conjunction (AND) of clauses, where each clause is a disjunction (OR) 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 shall present now a new learning rule for symmetric connections (possibly high-order arcs) and a fast learning algorithm that learns 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; they are called also 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 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 brought into a k-CNF form; thus, the algorithm works in theory for every set of presentations and for any unknown formula φ. It is not practical though, when k is too large. [19]

6.1. A learning rule for high-order symmetric connections

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

[18] The memories in such network are content-addressable: given partial description of a stored vector, the network searches to complete the rest of the bits.

[19] Fortunately, expert domains are regulated by relatively short rules and therefore small k is sufficient.

6.1.1. Checking whether to update the arc

The idea is that certain bit patterns in the training set should cause an updating of some weights if they are seen for the first time[20] A new combination of k bits in the training set, causes arcs that connect units involved in this bit combination to be updated. A hyper-arc is updated if it connects units that participate in the new pattern and the rest of the units of the pattern are not active; i.e., units of the pattern that are not in the arc should be zero instantiated.

6.1.2. Updating the weight

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

6.1.3. The k-clause learning rule (formally)

Let $Arc = \{X_{i_1}, \ldots, X_{i_l}\}$ be an l-order arc.

Given a presentation $\vec{x} = (x_1, \ldots, x_n)$ that instantiates the visible units to 0/1 values, the l-order arc Arc is updated iff there exists a new k-bit pattern $P = (X_{i_1} = x_{i_1}, \ldots, X_{i_l} = x_{i_l}, X_{j_1} = 0, \ldots, X_{j_{k-l}} = 0)$ that has never been seen in one of the earlier presentations, and that includes the units of Arc, such that the rest of the units $(X_{j_1}, \ldots, X_{j_{k-l}})$ are 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 6.2. Given the presentation $ABC = 011$, a 2-clause rule causes the following updates:

- The weight of arc AB is updated by $\triangle_{AB} = -1$, since the 2-bit combination $(A = 0, B = 1)$ is new and the arc contains an odd number of zeros.
- The weight of BC is updated by $\triangle_{BC} = +1$, since the 2-bit combination $(B = 1, C = 1)$ is new to the arc and the arc contains no zeros (even).
- The bias of unit B (which is the singleton arc $\{B\}$) is updated by $\triangle_B = +1$, since the 2-bit combination $A = 0, B = 1$ is new to the arc B, it is extended using zero-units $(A = 0)$ and the arc B includes no zeros (even).

The bias of A is not updated since it cannot be extended into a 2-bit combination by adding a zero unit. In a similar way, the arc AC is decremented (like AB), and the bias C is incremented (like the bias B).

[20] This is a 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 that a bit combination occurs is irrelevant to the rule we want to learn. A single presentation or a hundred repetitions generate the same representation.

6.2. Learning k-CNF

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

6.2.1. Algorithm to learn a k-CNF formula

Initialize all weights to zero.

For all the presentations in the training set using the k-clause rule, update the weights of all the l-order arcs $(0 < l \leqslant k)$ that needs to be updated.

Theorem 6.3. *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 of the k-CNF formula, and whose energy function is equal E_φ. The network is generated after a* single *pass over the presentations.*

Proof. Only a sketch of the proof is given:

The proof is based on showing that the algorithm is equivalent to Valiant's algorithm for learning k-CNF [68]. Valiant's algorithm starts with a list of clauses that consists of all possible k-clauses over n atomic propositions. For every presentation of a truth assignment that satisfies the unknown k-CNF (positive example) the algorithm eliminates clauses from the list that are not satisfied by the example.

It can be proved that in every step, the conjunction of the clauses in the list is a formula that is consistent with all the presentations seen so far and exactly these presentations. Therefore, when all the examples are seen, the k-CNF formula has been learned.

First, we need to show that the initial conjunction of all possible k-clauses can be represented by a network with zero weights (the initialization step of our algorithm). Later, we show that the clause elimination step is equivalent to the set of weight updates performed after each presentation.

An energy function of zero (or any constant energy function) represents formulas that are contradictions or tautologies that have the property that every model satisfies exactly the same number of clauses. The initial conjunction of all possible clauses in Valiant's algorithm is exactly such formula and therefore, the starting point of our algorithm (with the zero weights) represents the desired list.

The next step is to show that the set of weight updates performed by the k-clause rule after one presentation corresponds to Valiant's elimination step. A clause is eliminated in Valiant's algorithm if it is a disjunction

$$c = \bigvee_{x_{ij}=1} \neg X_{ij} \bigvee_{y_{ji}=0} Y_{ji}$$

of some k-bit pattern $P = (X_{i_1} = 1, \ldots, X_{i_m} = 1, Y_{j_1} = 0, \ldots, Y_{j_n} = 0)$, where the X_{i_l}'s are instantiated by the example to one, and the Y_{j_l}'s to zero. An elimination of a clause c is performed in the energy space by deleting the energy terms of E_c. Since the weights of the network are the energy terms with reverse signs, the weights are actually updated by adding:

$$E_c = H_{\neg c} = H_{\bigwedge X_{i_l} \bigwedge \neg Y_{j_l}} = \prod_{x_{i_l}=1} X_{i_l} \prod_{y_{j_l}=0} (1 - Y_{j_l}).$$

The arcs that are updated as a result of this addition, contain the units X_{i_l}'s in the pattern P that are instantiated with ones, while the rest of units are zeros (in P but not in the arc). If the number of zeros in the arc is odd, the sign of the term $\prod X_{i_j} \prod Y_{i_l}$ that corresponds to the arc is negative and the weight is decremented. If the number of zeros is even, the sign is positive and the weight is incremented. An arc is updated therefore if there exists a new bit combination P, whose one's are included in the arc (if any), while the rest of the bits of the combination (outside the arc) are all zeros. □

Example 6.4. Learning the XOR formula $(A \oplus B) \leftrightarrow C$, looking at the four satisfying truth assignments: $ABC \in \{011, 101, 000, 110\}$.

We need a 3-clause rule since we cannot express our formula in less than 3-CNF. The patterns we look for are therefore 3-bit combinations (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 Fig. 7. Its global minima are exactly the four presentations, and the high-order connection $(4ABC)$ can be replaced by second-order connections $(4AB - 8AH - 8BH + 8CH + 12H)$ by adding one hidden unit H.

The algorithm as described so far needs to know k a priori. Can we modify the algorithm to work when the learner does not have prior knowledge of k? To solve this problem, the general approach is to start with low k (for example $k = 1$), to activate the

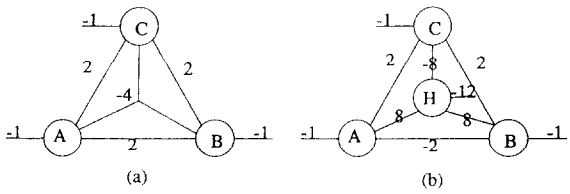

Fig. 7. The network for XOR $(A \oplus B) \leftrightarrow C$ that was constructed by the learning algorithm: (a) is the high-order network; (b) is the quadratic equivalent with one hidden unit H.

k-clause learning rule for an entire cycle and then to test whether the network performs well. If the network passes the test, the algorithm stops; otherwise, k is increased and another cycle of learning begins.

6.2.2. A general scheme for learning k-CNF when k is unknown

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

One approach to the TEST (line 3 of the above scheme) is discussed in [14] and is related to Valiant's notion of "probably approximately correct" (PAC) learning. In this model, positive and negative examples are given from some arbitrary distribution. The task is to find in polynomial time a network whose chances to have an error rate (on the same distribution) larger than ε are less than δ, for arbitrary small ε and δ. We would like also to make sure that the probability of the algorithm to use k larger than necessary are also less than δ. The approach in [14] satisfies the above conditions and guarantees that only polynomial time is needed (polynomial in n, $1/\delta$, $1/\varepsilon$, even if there are exponential number of satisfying models).

Another approach that is not PAC motivated but similar in style is when we are given a complete training set that contains all the satisfying assignments (only positive examples). The task is to generate a network that performs well (the probability of the network to be ε-bad is less than δ), and that is compact (the probability of using unnecessary large k, is also less than δ).

As in the PAC case, the test in this case is also a sequence of samples of the network behavior; each of the checks tests whether the network converges to a wrong global minimum. The test procedure allows up to m errors out of r checks.[21] If the number of errors is less than m then the test succeeds, otherwise, it fails and k is increased. A full discussion of these testing techniques is too lengthly, and the reader is referred to literature on learning theory and approximation theory.[22]

[21] m and r are computed from the epsilon-delta bounds using Chernoff bounds.

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

7. Experimental results

Experiments have been made on randomly generated propositional formulas. The formulas that were generated were conjunctions of 3-variable clauses (3-SAT).[23] The simulations have managed to find satisfying solutions to large scale 3-SAT problems with remarkable speed. Performance comparison of several symmetric models and the GSAT algorithm [57] are provided.

7.1. Simulations

I have tried three types of algorithms inspired from Hopfield networks, Boltzmann machines and mean field networks.[24]

7.1.1. The Hopfield version

Perform MAXTRIES tries or until a solution is found:
In each try:

- Randomly set the values of the units (zeros and ones).
- Perform Hopfield cycles until either MAXCYCLES cycles have been executed, a solution is found or until MAXCONST 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 value is flipped.

7.1.2. The Boltzmann version

Do until MAXT tries or until a solution is found:
In each try:

- Assign random (zero/one) values to the units.
- Anneal the system, starting with temp=1 until temp=0:
  - Perform a Boltzmann cycle;
  - Reduce temp by 1/TEMPSTEPS.

  When the temperature is zero, Hopfield cycles are executed until either MAXCYCLES tries have been performed,[25] a solution has been found, or MAXCONST continuous cycles could not reduce the energy.
- If MAXTRIES tries have not been executed and a solution hasn't been found, another annealing begins with TEMPSTEPS=TEMPSTEPS+DELTAT (annealing slows).

[23] 3-SAT problems are convenient for a benchmark because performance results of several other algorithms are available to compare.

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

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

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 Section 3.2.3).

7.1.3. The mean field version

As in the Boltzmann version, the simulator tries MAXTRIES annealings (each time the annealing is slower); however, the first annealing is done using mean field theory (MFT) cycles (see Section 3.2.4), while the rest of the annealings are done using Boltzmann cycles.[26]

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.

7.2. 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 [34].[27] 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 Table 3.

During the final stages of experiments[28] two recent algorithms that perform a very similar local search for satisfiability and may be seen as sequential variations of Hopfield networks became known to us [13,57]. In GSAT [57], maximum MAXTRIES tries are executed. In each try a random truth assignment is generated and variable "flips" are performed until either a solution is found or MAXTRIES 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*

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

[27] The way we generate the formulas is different from [34]. Our generator forces the formulas to be satisfiable, whereas in [34], random formulas are generated that are not forced to be satisfiable and later the Davis–Putnam algorithm [4] (which is based on resolution) is used to eliminate the unsatisfiable formulas. Our approach seems to make the distribution generated easier than that of [34] (B. Selman, private communication). Nevertheless, the comparisons to GSAT are still valid since all experiments were conducted with the same set of formulas. It remains to be seen whether similar results happen in unforced distributions.

[28] 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 [34].

Table 3

n	m	MAXTRIES	MAXCYCLES	MAXCONST	TEMPSTEPS	DELTAT
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

Table 4

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

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 [4]. We have implemented GSAT for the purpose of comparing the approaches. In [13], all the variables which increase the number of satisfied clauses are flipped. The algorithm of [13] was not directly implemented by us because of its close similarity to the Hopfield version.[29] For the purpose of fair comparison with GSAT, the values for n, m and the MAXTRIES and MAXCYCLES parameters were taken from the experiments reported in [57]. The rest of the parameters (MAXCONST, TEMPSTEPS and DELTAT) were intuitively taken according to the size of the problem.[30] No fine tuning of these parameters was done.

Table 4 gives the average number of cycles in which each of the algorithms found a solution.

Table 5 shows the percentage of experiments in which each of the algorithms managed to find a solution in the first trial.

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. The number of flips in a cycle was not counted

[29] The difference is that in [13] nodes are visited in a predefined order, and the vector that is generated when the algorithm fails is not truly random.

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

Table 5

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%

since all the flips in a cycle are done in one parallel step (constant time).[31] Note also that GSAT was not designed for parallel execution; the comparison could have been very different had we measured flips and not cycles.

For parallel execution, the connectionist approaches are clearly leading. Not surprisingly, MFT has the best performance for first-hit.

8. Related work and discussion

8.1. Connectionist approaches

Derthick [5] observed that weighted logical constraints (which he called "certainties") can be used in massively parallel architecture. Derthick translated those constraints into special energy functions and used them to implement a subset of the language KL-ONE. The approach described in this paper has some similarities with his system. Looking at his reduction from logic to energy functions (Derthick uses different energy functions and no hidden units), there are however, several basic differences: (1) Derthick's "mundane" reasoning is based on finding a most likely *single* model; his system is never skeptical. The system described in this paper is more cautious and closer in its behavior to symbolic nonmonotonic systems, described in recent literature (see Section 8.2). (2) The system that is described here can be implemented with standard low-order units, using relatively well-studied architectures like Hopfield networks or Boltzmann machines. It is possible therefore to take advantage of the hardware implementations as well as of the learning algorithms that were developed for these networks. (3) Formal proofs of two-way equivalence are given so that every network can be described as a PLOFF and not just the reverse. (4) A learning algorithm is given that achieves the same networks that are obtained from direct compilation.

Another connectionist nonmonotonic system is that of Shastri [58]. It uses evidential reasoning based on maximum likelihood to reason in inheritance networks. My approach is different; I use standard low-level connectionist models and am not restricted to

[31] Constant time for a cycle is certainly true for the parallel 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.

inheritance networks.[32] Shastri's system is guaranteed to work and has a polynomial time complexity, whereas the system described here tries to solve an intractable problem and trades correctness with time; i.e., a correct solution (a global minimum) is not guaranteed; however, the chance of finding one improves as more time is given.

This article shares with [1, 19, 59, 65] the implementationalist motivation [48]. These systems implement small subsets of predicate calculus by either spreading activation or by rule firing. The expressive power of these mechanisms is limited by performance and tractability considerations, and they all stress the problems of representing complex structures, syntax sensitivity and multi-place predicates. In this article I had no intention of attacking these problems; rather, I intended to show how to represent *any* propositional constraint, and how networks can cope naturally with conflicting beliefs. The technique however, *can* be extended further for representing first-order predicate calculus [43].

We may look at penalty logic as one of the layers of abstraction that are needed between descriptions of high-level cognitive processes and low-level neural implementations. Thus, penalty logic may be seen as a first level of abstraction that is higher than the neural implementation (see [1] for a nice discussion on the multi-span approach). Using the language described in this paper we can map several of the systems mentioned above into penalty logic, and then compile them into symmetric networks (possibly by sacrificing efficiency) [42].

8.2. *Symbolic systems*

Penalty logic is along the lines of work done in preferential semantics [60], and is related in particular to systems with preferential semantics that use ranked models, like those of Lehmann and Magidor [27], Lehmann [26] or Pearl [37].[33] Lehmann and Magidor's results about the relationship between rational consequence relations and ranked models can be applied to our paradigm: A strict consequence relation (induced by a PLOFF ψ) is a binary relation between a strict evidence and a strict conclusion. It is therefore a set of pairs $R_\psi = \{\langle\varphi', \varphi\rangle \mid \varphi' \models^\psi \varphi\}$, where both φ' and φ are strict WFFs. Lehmann and Magidor defined a *rational* consequence relation as one that satisfies certain conditions (inference rules), and proved that a consequence relation is rational iff it is defined by some ranking function. As a result, we may conclude a rather strong conclusion for our system: Every rational consequence relation is implementable in a symmetric network. Also, any symmetric network may be viewed as implementing some rational consequence relation. We can therefore be sure that every implementation of our connectionist inference engine induces a rational consequence relation.

Penalty logic is not based on Bayes reasoning. It was developed using the notion of preferential semantics. On the surface, it does not compute probabilities nor does it based on Cox axioms. However, tight relationships have been discovered between systems based on preferential semantics and systems based on epsilon semantics[34] that

[32] We can easily extend our approach to handle inheritance networks by looking at the atomic propositions as predicates with free variables. Those variables are bound by the user during query time.

[33] These systems are related in turn to probabilistic reasoning (Bayes systems) by means of epsilon semantics.

[34] In epsilon semantics probabilities *approach* zero or one.

are based on the Bayesian approach [10–12]. One such system (based on probabilistic interpretation) that can be reduced directly to penalty logic is that of Goldszmidt and Pearl [12] which actually computes the penalties from a given conditional knowledge based on maximal entropy considerations (the user does not specify any penalty). The system uses the same ranking function as the one described in this article; i.e., summing the penalties of violated beliefs.

Penalty logic has some similarities with systems that are based on priorities (given to beliefs). One such system [3] is based on levels of reliability. Brewka's system for propositional logic can be mapped (approximately) into penalty logic by selecting large enough penalties. Systems such as that of Poole [50] (with strict specificity) can also be implemented using our architecture, and as in [11], the penalties can be generated automatically. Another system that is based on priorities is system Z^+ [12] where the user does specify the penalties, but there is a "ghost" that changes them so that several nice properties hold (e.g. specificity). Penalty logic can only approximate priority systems by assigning scaled penalties.[35] Every conclusion that is entailed in a priority system such as system Z^+ will also be entailed by the approximating penalty logic knowledge base. However, some conclusions that are ambiguous in a priority system may be drawn decisively in penalty logic. In this sense penalty logic can be considered as bolder (less cautious) than those which are based on priorities [47].

For example consider the "penguins and the wings" case [12]. We are given the following defaults: birds fly; birds have wings; penguins are birds and penguins do not fly. Many systems based on priorities (like Z^+) will not be able to conclude that penguins have wings. Penalty logic in contrast will conclude according to our intuition; i.e., that penguins do have wings despite the fact that penguins do not fly. The reason for this intuitive deduction is that penalty logic considers the models where penguins do not fly but have wings to be more "normal" than models where penguins do not fly and also have no wings (as in [11]). Priority-based system will be ambiguous since they don't have such fine preference.

For another example consider the Nixon case (Example 2.2) when we add to it: $\langle 1000, N \rightarrow FF\rangle$ and $\langle 10, FF \rightarrow \neg P\rangle$ (Nixon is also a football fan and football fans tend not to be pacifist). Most other nonmonotonic systems will still be skeptical about P [10, 26, 30, 37, 64]. Our system boldly, and in contrast with intuition, decides $\neg P$ since it is better to defeat the one assumption supporting P than the two assumptions supporting $\neg P$. In this particular case however, we can correct this behavior by changing the penalty for $Q \rightarrow P$ (multiplying by two). Further, a network like our system that learns, may adjust the penalties autonomously and thus develop its own intuition and nonmonotonic behavior.

Because we do not allow for arbitrary partial orders [10, 60] of the models, there are other fundamental[36] problematic examples where our system (and all systems with ranked-models semantics) boldly concludes, while other systems are skeptical (these are cases where the intuition tell us that skepticism is the right behavior).

[35] The penalties are scaled so that there is no subset of low-priority assumptions whose sum exceeds the penalty of a higher priority.

[36] Hector Geffner (private communication).

The following is an example for which we have clear intuition; nevertheless, it is possible to prove that no ranking function exists that induces the intuitive behavior we wish:

Example 8.1. Assume the following defeasible rules: $A \rightarrow D$, $B \rightarrow \neg D$ and $C \rightarrow \neg D$. The intuition we have states that:

- Given A, C, D we should conclude $\neg B$; therefore,

$$rank(A\bar{B}CD) < rank(ABCD).$$

- Given A, B, C we should conclude that D is ambiguous; therefore,

$$rank(ABCD) = rank(ABC\bar{D}).$$

- Given $A, C, \bar{D}$ we should conclude that B is ambiguous; therefore,

$$rank(ABC\bar{D}) = rank(A\bar{B}C\bar{D}).$$

- Given $A, \bar{B}, C$ we should conclude that D is ambiguous; therefore,

$$rank(A\bar{B}C\bar{D}) = rank(A\bar{B}CD).$$

This is a contradiction since $rank(A\bar{B}CD) < rank(A\bar{B}CD)$. Thus, the intuition as stated by the examples cannot be implemented by any ranked model.

9. Conclusions

The main contributions of this paper are: (1) the development of theoretical foundations for a connectionist inference engine that is capable of representing and learning propositional knowledge; (2) rigorously relating and unifying two (sometime opposing) knowledge representation mechanisms: connectionist networks and propositional logic; (3) demonstrating (experimentally) the efficiency of the algorithm used by SCNs for large scale, randomly generated 3-SAT problems.

Along these lines I have introduced penalty logic and showed mappings between its sentences and SCNs. Penalty logic may be used as a framework for defeasible reasoning and inconsistency handling. Several systems can be mapped into this paradigm and therefore suggest settings of the penalties. When the right penalties are given, penalty logic features a nonmonotonic behavior that (usually) matches our intuition. It is possible to show, though, that some intuitions cannot be expressed as ranking functions.

A strong equivalence between sentences of penalty logic and symmetric networks is formally proved. This two-way equivalence serves two purposes: (1) we can translate a sentence of penalty logic into an equivalent network (this serves the basic construction of our inference engine); (2) any symmetric network (and also any asymmetric non-oscillating network) can be described by penalty logic sentences. The logic may thus be used as a specification language (at higher level of abstraction), and gives another clarifying look at the dynamics of such networks.

Several equivalent high-level languages can be used to describe SCNs: (1) quadratic energy functions; (2) high-order energy functions with no hidden units; (3) propositional logic; and finally (4) penalty logic. All these languages are expressive enough to describe any SCN and every sentence of such languages can be translated into an SCN; however, penalty logic has properties that make it more attractive than the other languages. Algorithms are given for translating between any two of the languages above.

An inference engine is constructed that is capable of answering whether a query follows the formula (knowledge) or not. When a query is clamped, the global minima of the network correspond exactly to the correct answers.

The engine can obtain its knowledge either by compiling a symbolic formula or by learning it inductively from examples. Any unknown k-CNF formula can be learned in linear time (in the length of the size of the training set) providing k is a small constant. The learning algorithm is shown to be equivalent to a powerful symbolic algorithm developed within the PAC paradigm.

Revision of the knowledge and adding new evidence are easy tasks if we use penalty logic to describe the network: adding (or deleting) a PLOFF is simply computing the energy function of the new PLOFF and then adding (deleting) the energy terms to the function that describes the existing knowledge. Thus, a local change to the PLOFF describing the network is translated to a local change in the network.

The mappings given in this paper are limited to propositional knowledge; however, their potential exceeds the propositional case, and allows also for higher-level more expressive languages (like first-order predicate logic) to be represented [43].

I have implemented several nonmonotonic toy problems (like Nixon, Penguins, etc.) on a Boltzmann machine simulator, and the network managed to always find a global minimum. I have not noticed any problems with local minima although they definitely exist. The technique scales well for large randomly generated 3-SAT problems and there are evidence that similar heuristic repair methods provide good results to other NP-hard problems (e.g., n-queens [33]). Advances in heuristic repair techniques (e.g., [36]) or in connectionist energy minimization (e.g., [39,45,46] may add additional speed to the approach.

The ability of these networks to learn and make adjustments in the energy landscape, may provide a new research direction. Learning algorithms may be used to speed up the network convergence time, by eliminating local minima and widening global minima.[37] If such research is successful, the techniques described in this article may be used to build networks that perform fast symbolic constraint satisfaction and are able to accelerate their own speed with time.

Acknowledgment

Thanks to Jon Doyle, Hector Geffner, Sally Goldman, Dan Kimura, Stan Kwasny, Fritz Lehmann, Ron Loui, Judea Pearl and Dave Touretzky for helpful discussions.

[37] Similar techniques have been tried for improving local search [36].

References

[1] J.A. Branden, Encoding complex symbolic data structures with some unusual connectionist techniques, in: J.A. Branden and J.B. Pollack, eds., *Advances in Connectionist and Neural Computation Theory* 1: *High-level connectionist models* (Ablex, New York, 1991).

[2] R.D. Brandt, Y. Wang, A.J. Laub and S.K. Mitra, Alternative networks for solving Traveling Salesman problem and the list matching problem, in: *Proceedings IEEE International Conference on Neural Networks*, San Diego, CA (1988).

[3] G. Brewka, Preferred sub-theories: an extended logical framework for default reasoning, in: *Proceedings IJCAI-89*, Detroit, MI (1989) 1043–1048.

[4] M. Davis and H. Putnam, A computing procedure for quantification theory, *J. ACM* **7** (1960) 201–215.

[5] M. Derthick, Mundane reasoning by parallel constraint satisfaction, Ph.D. Thesis, CMU-CS-88-182, Carnegie Mellon University, Pittsburgh, PA (1988).

[6] M. Derthick, Mundane reasoning by parallel constraint satisfaction, *Artif. Intell.* **46** (1–2) (1990) 107–157.

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

[8] J.A. Fodor and Z.W. Pylyshyn, Connectionism and cognitive architecture: a critical analysis, *Cognition* **28** (1988) 3–71.

[9] S. Geman and D. Geman, Stochastic relaxation, Gibbs distributions and the Bayesian restoration of images, *IEEE Trans. Pattern Anal. Mach. Intell.* **6** (1984) 721–741.

[10] H. Geffner, Defeasible reasoning: causal and conditional theories, Ph.D. Thesis, Department of Computer Science, University of California, Los Angeles, CA (1989).

[11] M. Goldszmidt, P. Morris and J. Pearl, A maximum entropy approach to nonmonotonic reasoning, in: *Proceedings AAAI-90*, Boston, MA (1990) 646–652.

[12] M. Goldszmidt and J. Pearl, System Z^+: a formalism for reasoning with variable-strength defaults, in: *Proceedings AAAI-91*, Anaheim, CA (1991) 399–404.

[13] J. Gu, Efficient local search for large satisfiability problem, *SIGART Bull.* **3** (1) (1992) 8–12.

[14] D. Haussler, M. Kearns, N. Littlestone and M. Warmuth, Equivalence of models for polynomial learnibility, *Inf. Comput.* (to appear); also in: *Proceedings Workshop on Computational Learning Theory* (1988) 42–55; also Technical Report UCSC-CRL-88-06 (1988).

[15] G.E. Hinton, Deterministic Boltzmann learning performs steepest descent in weight space, *Neural Comput.* **1** (1) (1989).

[16] G.E. Hinton, Preface to the Special Issue on Connectionist Symbol Processing, *Artif. Intell.* **46** (1990) 1–4.

[17] G.E. Hinton and T.J. Sejnowski, Learning and re-learning in Boltzman Machines, in: J.L. McClelland, D.E. Rumelhart and the PDP Research Group, eds., *Parallel Distributed Processing: Explorations in the Microstructure of Cognition* **1** (MIT Press, Cambridge, MA, 1986) 282–317.

[18] S. Hölldobler, A structured connectionist unification algorithm, in: *Proceedings AAAI-90*, Boston, MA (1990); also ICSI Technical Report TR-90-012 (1990).

[19] S. Hölldobler, CHCL, a connectionist inference system for Horn logic based on connection method and using limited resources, International Computer Science Institute TR-90-042 (1990).

[20] J.J. Hopfield, Neural networks and physical systems with emergent collective computational abilities, *Proc. Nat. Acad. Sci.* **79** (1982) 2554–2558.

[21] J.J. Hopfield, Neurons with graded response have collective computational properties like those of two-state neurons, *Proc. Nat. Acad. Sci.* **81** (1984) 3088–3092.

[22] J.J. Hopfield and D.W. Tank, Neural computation of decisions in optimization problems, *Biol. Cybern.* **52** 144–152.

[23] A.B. Kahng, Traveling salesman heuristics and embedding dimension in the Hopfield model, in: *Proceedings International Joint Conference on Neural Networks* (1989) 513–520.

[24] S. Kasif, S. Banerjee, A. Delcher and G. Sullivan, Some results on the computational complexity of symmetric connectionist networks, Technical Report JHUCS-89-10, Department of Computer Science, Johns Hopkins University, Baltimore, MD (1989).

[25] T.E. Lang and M.G. Dyer, High-level inferencing connectionist network, *Connection Sci.* **1** (2) (1989) 181–217.
[26] D. Lehmann, What does a conditional knowledge base entail?, in: *Proceedings International Conference on Knowledge Representation and Reasoning*, Toronto, Ont. (1989) 212–222.
[27] D. Lehmann and M. Magidor, Rational logics and their models: a study in cumulative logic, Technical Report TR-86-16, Leibnitz Center for Computer Science, Hebrew University, Jerusalem, Israel (1988).
[28] H.J. Levesque, A fundamental tradeoff in knowledge representation and reasoning, in: *Proceedings CSCSI-84*, London, Ont. (1984) 141–152.
[29] V. Lifschitz, Computing circumscription, in: *Proceedings IJCAI-85*, Los Angeles, CA (1985).
[30] R.P. Loui, Defeat among arguments: a system of defeasible inference, *Comput. Intell.* **3** (3) (1987).
[31] J. McCarthy, Programs with commonsense, in: M. Minski, ed., *Semantic Information Processing* (MIT Press, Cambridge, MA, 1968) 403–418.
[32] J. McCarthy, Circumscription—a form of nonmonotonic reasoning, *Artif. Intell.* **13** (1980) 27–39.
[33] S. Minton, M.D. Johnson and A.B. Phillips, Solving large scale constraint satisfaction and scheduling problems using a heuristic repair method, in: *Proceedings AAAI-90*, Boston, MA (1990) 17–24.
[34] D. Mitchell, B. Selman and H.J. Levesque, Hard and easy distribution of SAT problems, in: *Proceedings AAAI-92*, San Jose, CA (1992) 459–465.
[35] M. Minsky, Logical versus analogical, symbolic versus connectionist, neat versus scruffy, *AI Mag.* **12** (2) (1991).
[36] P. Morris, The breakout method for escaping from local minima, in: *Proceedings AAAI-93*, Wasington, DC (1993) 40–45.
[37] J. Pearl, System Z: a natural ordering of defaults with tractable applications to nonmonotonic reasoning, in: *Proceedings Theoretical Aspects of Reasoning about Knowledge*, Pacific Grove, CA (1990) 121–135.
[38] C. Peterson and E. Hartman, Explorations of mean field theory learning algorithm, *Neural Networks* **2** (6) (1989).
[39] C. Peterson and B. Söderberg, A new method for mapping optimization problems onto neural networks, In *Int. J. Neural Syst.* **1** (1989) 3–22.
[40] G. Pinkas, Energy minimization and the satisfiability of propositional calculus, *Neural Comput.* **3** (2) (1991); also in: D.S. Touretzky, J.L. Elman, T.J. Sejnowski and G.E. Hinton, eds., *Proceedings of the 1990 Connectionist Models Summer School* (Morgan Kaufmann, San Mateo, CA, 1990).
[41] G. Pinkas, Propositional non-monotonic reasoning and inconsistency in symmetric neural networks, in: *Proceedings IJCAI-91*, Sydney, Australia (1991).
[42] G. Pinkas, Converting binary threshold networks into symmetric networks, Technical Report WUCS-91-31, Computer Science Department, Washington University, St. Louis, MO (1991).
[43] G. Pinkas, Constructing proofs in symmetric networks, in: J.E. Moody, I.J. Hanson and R.P. Lipman, *Advances in Neural Information Processing Systems IV* (1992) 217–224.
[44] G. Pinkas, Logical inference in symmetric connectionist networks, Doctoral Thesis, Washington University, St. Louis, MO (1992).
[45] G. Pinkas and R. Dechter, A new improved connectionist activation function for energy minimization,, in: *Proceedings AAAI-92*, San Jose, CA (1992) 434–439.
[46] G. Pinkas and R. Dechter, On improving connectionist energy minimization, *J. AI Research* (to appear).
[47] G. Pinkas and R. Loui, Reasoning from inconsistency: a taxonomy of principles for resolving conflicts, in: *Proceedings Third International Conference on Principles of Knowledge Representation and Reasoning*, Cambridge, MA (1992).
[48] S. Pinker and A. Prince, On language and connectionism: analysis of a parallel distributed processing model of language acquisition, *Cognition* **28** (1988) 73–193.
[49] D. Poole, On the comparison of theories: preferring the most specific explanation, in: *Proceedings IJCAI-85*, Los Angeles, CA (1985) 144–147.
[50] D. Poole, A logical framework for default reasoning, *Artif. Intell.* **36** (1988) 27–47.
[51] N. Rescher and R. Manor, On inference from inconsistent premises, *Theory Decision* **1** (1970) 179–217.
[52] N. Rescher, *Plausible Reasoning* (Van Gorcum, 1976).
[53] R. Reiter, A logic for default reasoning, *Artif. Intell.* **13** (1980) 81–132.
[54] J.A. Robinson, A machine-oriented logic based on the resolution principle, *J. ACM* **12** (1) (1965) 23–41.

[55] D.E. Rumelhart, G.E Hinton and J.L. McClelland, A general framework for parallel distributed processing, in: J.L. McClelland, D.E. Rumelhart and the PDP Research Group, eds., *Parallel Distributed Processing: Explorations in the Microstructure of Cognition* **1** (MIT Press, Cambridge, MA, 1986).

[56] T.J. Sejnowski, Higher-order Boltzman machines, neural networks for computing, *Proc. Amer. Inst. Phys.* **151**, Snowbird, UT (1986) 39–84.

[57] B. Selman, H.J. Levesque and D. Mitchell, A new method for solving hard satisfiability problems, in: *Proceedings AAAI-92*, San Jose, CA (1992) 440–446.

[58] L. Shastri, *Semantic Networks: An Evidential Formulation and Its Connectionist Realization* (Pitman, London, 1988).

[59] L. Shastri and V. Ajjanagadde, A step toward modeling reflexive reasoning, *Behav. Brain Sci.* **16** (3) (1993) 477–494.

[60] Y. Shoham, *Reasoning about Change* (MIT Press, Cambridge, MA, 1988).

[61] E.H. Shortliffe, *Computer-Based Medical Consultation, MYCIN* (Elsevier, New York, 1976).

[62] G. Simari and R.P. Loui , Mathematics of defeasible reasoning and its implementation, *Artif. Intell.* **53** (1992) 125–157.

[63] P. Smolensky, Information processing in dynamic systems: foundations of harmony theory, in: J.L. McClelland, D.E. Rumelhart and the PDP Research Group, eds., *Parallel Distributed Processing: Explorations in the Microstructure of Cognition* **1** (MIT Press, Cambridge, MA, 1986).

[64] D.S. Touretzky, *The Mathematics of Inheritance Systems* (Pitman, London, 1986).

[65] D.S. Touretzky and G.E. Hinton A distributed connectionist production system, *Cognitive Science* **12** (3) (1988) 423–466.

[66] R.J. Williams, The logic of activation functions, in: J.L. McClelland, D.E. Rumelhart and the PDP Research Group, eds., *Parallel Distributed Processing: Explorations in the Microstructure of Cognition* **1** (MIT Press, Cambridge, MA, 1986).

[67] G.V. Wilson and G.S. Pawley, On the stability of the travelling salesman problem algorithm of Hopfield and Tank, *Biol. Cybern.* **58** (1988) 63–70.

[68] L.G. Valiant, A theory of the learnable, *Commun. ACM* **27** (1984) 1134–1142.