Complexity of Connectionist Learning with Various Node Functions

J. Stephen Judd

COINS Technical Report 87-60

Department of Computer and Information Science
A305, Graduate Research Center
University of Massachusetts
Amherst, MA 01003
July 1987

This paper (without appendices) was presented at the First I.E.E.E. *International Conference on Neural Networks*, June 21 – 24, 1987, San Diego, California.

Abstract

We formalize a notion of learning in connectionist networks that characterizes the training of feed-forward networks. Considering different families of node functions, we prove the learning problem *NP*-complete and thus demonstrate that it has no efficient general solution. One family of node functions studied is the set of logistic-linear functions, as used by the popular back-propogation algorithm. Several implications of the theorem are discussed, including why this result is actually helpful for connectionist learning research.

This research was supported by the Air Force Office of Scientific Research, Bolling AFB, through grant AFOSR-87-0030. The author was also supported by an NSERC Canada Post-Graduate Scholarship.

1 Introduction

One goal of researchers studying connectionist models is to develop an efficient learning algorithm for networks. The published successes in connectionist learning have been for very small networks, typically much less than 100 nodes. To fully exploit the expressive power of networks, we need to be able to scale them up to much bigger sizes. But it is widely acknowledged that as the networks get larger and deeper, the amount of time required for them to absorb the training data grows prohibitively [7,18,2,12]. Some researchers view this scale-up issue as the most important problem for current connectionist research.

Many researchers have developed algorithms for learning in connectionist networks. Some notable approaches are the Perceptron [15,11], back-propagation [17,13,5], Boltzmann [1,8], and associative reward-penalty (A_{R-P}) [4,3] schemes. The implicit goal of this research is to find a method of learning associated pairs of given values that will work in an arbitrary network. Descriptions of these methods have each been published along with demonstrations of their ability on *selected* associative learning problems. However, no proof of their effectiveness has been offered (as it was for the linearly separable case of the Simple Perceptron [11]), despite the importance of such a result to connectionist research. Are there feasible algorithms for learning in large connectionist networks? Or is there some deep reason why there cannot be? This paper addresses that question.

Here, 'connectionist learning' is treated as mere memorization of some given data by a given network. This problem is formalized in Section 2 and a theorem in Section 3 indicates that in fact there can *not* exist an efficient algorithm for the general case of connectionist learning. The more involved problem of finding regularities in data and generalizing from them is not considered because the easier problem of simply remembering the data is shown to be too difficult to reliably achieve.

Section 4 explores the nature of the strictures revealed by this result and reports that the use of different types of node functions does not make the problem any easier. In Section 5 we explain why the results are *not* a death knell for connectionist learning and argue rather that they are a step forward toward large networks.

2 A Formalization of Network Learning

2.1 The Learning Protocol

The type of learning investigated here is known as supervised learning. In this paradigm input patterns (called *stimuli*) are presented to a machine paired with their desired output patterns (called *responses*). The object of the learning machine is to remember all the associations presented during a training phase so that in future tests the machine will be able to emit the associated response for any given stimulus.

In what follows, every stimulus σ is a fixed-length string of s bits, and every response ρ is a string of r bits with "don't cares", that is $\sigma \in \{0,1\}^s$ and $\rho \in \{0,1,*\}^r$. The output from a net is an element of $\{0,1\}^r$. The purpose of a response string is to specify constraints on what a particular output can be: we say that an output string, θ, agrees with a response string, ρ, if each bit, θ_i, of the output equals the corresponding bit, ρ_i, of the response whenever $\rho_i \in \{0,1\}$. The notation for such agreement is $\theta \models \rho$. Each stimulus/response pair is called an SR *item*. A *task* is a set of SR items that the machine is required to learn. To be reasonable, every stimulus in a task should be associated with no more than one response. Equivalently, a task T should be extendable to some function $f : \{0,1\}^s \rightarrow \{0,1\}^r$. We view functions as sets of ordered pairs and use the notation $T \subseteq f$ to mean $T \subseteq \{(\sigma,\rho) : f(\sigma) \models \rho\}$.

2.2 Network Architecture

The model of connectionist machines considered here is that of non-recurrent, or feed-forward, networks of computing elements. This is a generalized combinational circuit; the connections between nodes form a directed acyclic graph, and the nodes perform some function of their inputs as calculated by previous nodes in the graph.

We define an *architecture* as a 5-tuple $A = (P,V,S,R,E)$ where
P is a set of *posts*,
V is a set of n *nodes:* $V = \{v_1, v_2, \ldots, v_n\} \subseteq P$,
S is a set of s *input posts:* $S = P - V$,
R is a set of r *output posts:* $R \subseteq P$, and
E is a set of directed *edges:* $E \subseteq \{(v_i, v_j) : v_i \in P,\ v_j \in V,\ i < j\}$
The constraints on the edges ensure that no cycles occur in the graph. Denote the set of *input posts to node* v_k as $p(v_k) = \{v_j : (v_j, v_k) \in E\}$.

2.3 Node Functions

Each node in a network contributes to the overall computation by taking signals from its input edges and computing an output signal. In our first analysis, we consider only binary-valued functions.

$$f_i : \{0,1\}^{|p(v_i)|} \rightarrow \{0,1\}$$

The function f_i is a member of a given set $\mathcal{F}$ of functions. Typically, connectionists have used the set of linearly separable functions (LSFns) for $\mathcal{F}$. These functions are characterized by a threshold value and a weight associated with each input to a node. An 'activation level' is calculated from the weighted sum of the inputs, and the output is one of two values depending on whether the activation is above or below the threshold.

We consider LSFns as well as a variety of other node function sets. Two variants that are considered are called SAFns and LUFns. SAFns is the set of all those functions computable by a single multiple-input AND gate augmented with any number of inverters placed on the inputs or output (SA is from Solitary Address, or Single And). LUFns is the set of all Boolean functions (LU is from Look-Up table). Note LUFns $\supseteq$ LSFns $\supseteq$ SAFns.

We also consider two sets of node functions that have real values. Quasi-linear functions (QLFns) are functions composed of any bounded, monotonic function, E, applied to a linear combination of the inputs. (This definition is essentially the same as that used in [16] and [20].) A special case of QLFns is the logistic-linear functions (LLFns), for which $E(x) = 1/(1+e^{-x})$. The back-propagation algorithm of [17] is designed to work with LLFns.

A *configuration* of a network is a set of n functions $F = \{f_1, f_2, \ldots, f_n\}$ corresponding to the set of nodes, V, meaning that f_i is the function that v_i computes.

2.4 The Computational Problem

In a configured network, every node performs a particular function and the network as a whole performs a particular composite function. An architecture, A, and a configuration, F, together define a mapping from the space of stimuli to the space of responses:

$$M_F^A : \{0,1\}^s \rightarrow \{0,1\}^r.$$

A task, as defined above, can be viewed as a collection of constraints on the mapping that a network is allowed to perform. Recall that an SR item in a

task is a pair of strings (σ, ρ). When the posts in S are assigned the values of respective elements of σ, the network mapping defines values for each post in R. It is required that these values agree with respective elements of ρ.

The process of *loading* can now be defined. In the learning problem we are considering, an architecture and a task are given, and loading is the process of assigning an appropriate response function to every node in the architecture so that the derived mapping includes the task. It is a search procedure that accepts a pair (A, T) and returns a *solution*, which is a configuration F such that $T \subseteq M_F^A$. In the case where no such configuration exists, the procedure announces that fact.

3 The Intractability of Network Learning

Our major question regards the intrinsic nature of the learning problem we have posed: How difficult is it to load a given task into a given architecture?

The loading problem is a search problem, but it is usual to frame complexity questions in terms of decision problems. In the space of all possible (A, T) pairs, some pairs will have solution configurations and some will not; that is, for some pairs the architecture can *perform* the task, and for some it cannot. The *performability* decision problem is simply to tell whether a configuration exists: "Can A perform T?". Thus the complexity question posed above is re-phrased as: How difficult is it to decide performability? Re-phrasing again: How much computation is required to recognize the following (paramaterized) language?

$$Perf_{\mathcal{F}} = \{(A, T) : \exists F \in \mathcal{F}^n \ni T \subseteq M_F^A\}$$

(Terminology used here and the related complexity-theoretical concepts of *NP*-completeness are explained well in Garey and Johnson [6].)

The measure of how 'difficult' a decision problem is must be relative to the 'size' of a particular instance of the problem. The size of an instance of the performability problem is taken to be the number of bits that it takes to represent the instance i.e. the architecture and the task. This number is roughly proportional to $n + |T|$. As the architecture gets bigger or as the task gets bigger, one would expect any learning algorithm to take longer to solve it, but the question we would like to answer is "How much longer?" What is the minimum expression $g(s)$ for the worst-case amount of time required to solve an instance of size s? The following theorem is proved in the appendix:

Theorem 1 *$Perf_{SAFns}$ is NP-complete.*

Decades of experience have shown that the g function for any *NP*-complete problem is an exponential expression that becomes unmanageably large at quite small values of s. (See [6] chapter 1.) Hence it is not practical to try to decide large instances of the performability question.

Furthermore, because this decision problem is no harder than the search problem from which it is distilled, the loading problem per se is also intractable. No single general-purpose algorithm can be developed for use in arbitrary architectures that is guaranteed to load any given performable task in polynomial time. (This is true whether 'the algorithm' is conceived as a nodal entity working in a distributed fashion with other nodes, or as a global entity working in a centralized fashion on the network as a whole.) Hence it might appear that we cannot hope to build large connectionist networks that will reliably learn simple supervised learning tasks.

4 Why this is Bad News

We suggested above that a critical current issue for connectionism is to find a learning algorithm that is useful for large-scale networks. Questions of scale-up are precisely what complexity theory is intended to answer, and in the theorem above, complexity theory has declared our learning problem to be intractable in large networks. There are many (architecture,task) pairs for which there is no feasible way to find a solution even though solutions do exist.

There are ways of accommodating this result, but there are several reactions that are inappropriate. We consider individually and dispel each of the following invalid responses:

- *"We will use different learning rules."* The result above is not a measure of running time for one particular learning algorithm—it is a result about the intrinsic difficulty of the *problem*. There is *no* learning rule that can always solve this problem in polynomial time.
- *"The architecture is not part of the input to our algorithms."* It is true throughout the current connectionist literature that no description of the whole network is given to the nodes in the net, but this only implies that what connectionists are attempting to do is even *harder* than the formal problem phrased in this paper—they hope that their distributed algorithms will operate effectively with very limited local knowledge about the network surrounding each node.

- *"We will exploit massive parallelism."* This response is an attempt to contain an exponential expression (c^n) by dividing it by a linear expression (cn), but the arithmetic defeats it soundly.

In many connectionist approaches to learning, there is a strong reason why large numbers of computing elements will not accomplish the loading problem in feasible time: By doubling the number of nodes available, you are doubling the computational resources but you may also be doubling (or squaring!) the amount of computing that has to be done. Naive attempts to exploit parallelism can actually be counterproductive.

- *"We use only very shallow nets."* It has been an empirical observation that although some algorithms (notably back-propagation) work well in nets that have only a few levels intervening between input posts and output posts, they work much slower in deep nets. But one should not believe that the problem of scaling up derives solely from some phenomenon of depth. The proof in the appendix shows that *NP*-completeness appears even in nets of depth two!
- *"We only use the linearly separable functions in our nodes."* The theorem deals only with SAFns; LSFns includes all of SAFns and, when the number of inputs to a node is large, it is considerably more powerful. It might seem, therefore, that this extra power would make loading easier. Unfortunately, this case (and even LUFns) is just as hard (Corollary 3 and 4 in appendix).
- *"We use quasi-linear node functions."* LSFns is a binary-valued special case of the quasi-linear functions (QLFns). Theorem 1 pertains only to discrete, binary-valued signals and does not apply to real-valued quasi-linear functions. However, Theorem 5 in the appendix is an extension of Theorem 1 that pertains specifically to the logistic-linear functions (LLFns) used in back-propagation. It shows that $Perf_{LLFns}$ is *NP*-complete and, as a corollary, that $Perf_{QLFns}$ is *NP*-hard. (*NP*-hard problems are at least as difficult as *NP*-complete problems.)
- *"We want our machines to generalize, not just parrot their training task."* Induction ultimately requires the ability to remember facts. We have not considered induction in this paper because we have shown that the easier job of accurately remembering given facts is too difficult to generally achieve.
- *"We will relax our standards and accept a certain percentage of accuracy."* Our mathematical question has a very exacting criterion of success in training: either the machine performs perfectly or it doesn't. One could speculate that if the criterion was more lenient then the problem might be much easier. But the result can be proved again for cases where more than 80% of the items are required to be correct (proof beyond the scope

of this paper), or where only 80% (say) of the defined bits in each response are required to be correct (proof easy). Lowering standards of performance slightly does not immediately make the problem easy.

• *"We have no "don't-cares" in the responses."* The proof of the theorem does use the "don't-care" symbol but there is an easy repair of the theorem that avoids the "don't-care" by using an extra signal for input to each of the nodes. This detail does not strongly alter the nature of the problem.

None of the retorts in this list will extricate us from the tarpit of *NP*-completeness. The difficulties in the problem are deep and to avoid them the goals of connectionist learning research will have to be more narrowly defined.

5 Why this is Not Bad News

Essentially, the goal we have formulated is to find *one* algorithm that is *guaranteed* to load *any* performable task in *any* conceivable net. Since this is unachievable, we consider several ways to weaken the formulation so as to possibly yield an achievable goal.

First, the theorem is a statement about networks and tasks *in general*, but there may be large useful classes of networks (defined by some design restrictions) where loading a task would always be achievable in polynomial time. One can imagine several ways to constrain the class of networks and/or tasks and/or other aspects in such a way that the new loading problem would have some special regularity in it that might facilitate its solution. For most such sub-cases, our theorem says nothing.

It might be possible that architectural constraints alone may lead to tractable loading, but one approach that will not work is merely to restrict the maximum fan-in to each node. Theorem 1 holds even when fan-in is restricted to be no more than 3. More likely to help is a restriction that sets a *minimum* fan-in, because this forces a minimum number of degrees of freedom everywhere. By a similar line of reasoning, it may also help to set a *minimum* on the number of layers that must exist in the net (say perhaps as a function of the width of the net). This suggestion may contradict experimental evidence, but it follows from understanding the source of *NP*-completeness we have discovered.

The difficulties inherent in connectionist learning may be manageable through some notion of network modularization. Some researchers have experimented with such an approach, although it is not yet clear exactly

how best to break up the overall problem.

Our result might be overcome by using probabilistic algorithms. Perhaps for some class of architectures there is a randomized procedure that will run in polynomial time and report a solution configuration with a certain minimum probability. Repeated invocations of the procedure would give asymptotic certainty regarding performability.

One possible technique that has not been exploited by any connectionist learning schemes to date is to 'prepare' the network prior to seeing the task. It is conceivable that some amount of preparatory computation regarding the architecture alone could greatly assist the subsequent loading of the task. The formulation of *Perf* has excluded this possibility, (as have all the other published approaches) but new formulations might include it.

Another avenue of freedom usually not exploited by connectionist learning schemes is to alter the architecture as learning proceeds. When carried to extremes, this would amount to an exercise in arbitrary circuit design, rather than in connectionist learning, but adhering rigidly to the starting architecture may be just too constrictive; somewhere between these two extremes there may be a balance that combines the best of both worlds. (Valiant and others [19,10] have initiated the study of what can be feasibly learned under the free-design extreme; perhaps the two approaches will eventually find a middle ground.)

It is conceivable that the difficulties in loading stem specifically from the non-recurrence of the nets and the fact that all their 'knowledge' about a stimulus must be elicited in one single evaluation of each node function. If so, then a more reasonable model of network memory might involve storing data as cycles in state-space where the power of attractor dynamics could be exploited to make loading easier (albeit at the cost of more expensive retrieval). Such would be a large departure from our model but there are plenty of pitfalls there too; Porat [14] proves that the problem of deciding just if a configured network stabilizes or cycles is *NP*-hard.

Finally, our formulation of the learning problem may be inappropriate in that it requires a network to be able to load too large a class of tasks. By using performability as the decision problem, we are in effect defining the task class in terms of the architecture itself and asking that any architecture A be able to load any task in the set $P^A = \{T : \exists F \ni \mathcal{M}_F^A \supseteq T\}$. But it is not necessary to expect an architecture to be able to load all of these tasks. From a practical point of view, all that is necessary is that it be able to perform and load some useful class, $\mathcal{T}$, of tasks. Obviously, it is necessary that $\mathcal{T} \subseteq P^A$, and the results herein show that it is too ambitious to have

$\mathcal{T} = P^A$ for arbitrary A. However, there are many ways to define $\mathcal{T}$ so as to exclude some tasks in P^A, thus possibly leading to a loadable class. For instance, are monotonic tasks always loadable? Can networks always load tasks with a 'small' number of items? Can a network, A, load all tasks performable by nets 'half the size' of A? It would be useful to be able to characterize just what class of tasks a network could learn, or conversely, to be able to determine what type of architectures could learn a given class of tasks.

6 Conclusions

The job of simply remembering associated pairs of strings requires only linear time in a von Neumann machine, but we have shown that this trivial problem can become very difficult if it must be achieved in a given non-recurrent network. Hence there is reason for connectionist research to find out how to avoid the problem in its full generality. We have also given good evidence that the difficulty of the problem is independent of the choice of type of functions that each node can perform.

In their book *Perceptrons* [11], Minsky and Papert lament the lack of an effective procedure for loading networks and express a hope that "some profound reason for the failure to produce an interesting learning theorem for the multilayered machine will be found." Our result supplies one such reason, and the proof of the theorem stands as an opening insight into the reasons why the loading problem is so difficult.

We have outlined a wide range of questions regarding narrowed or altered models of the connectionist learning goal. The tool of *NP*-completeness can direct the search for achievable goals and often reveals how to solve the problems as well. Additional answers to some of our questions will assist connectionist learning research by further narrowing its focus to just those cases that hold the promise of scaling up.

Acknowledgements

This work derives from the author's Ph.D. thesis in progress under the direction of A. G. Barto who inspired the dual topic of networks and learning. The author thanks D.A.M. Barrington for guidance on complexity issues, and S. Porat for help in proof-reading.

Appendix

This section is a proof of 2 theorems with corollaries.

First, we introduce some general purpose notation for manipulating strings. If α and β are strings then $\alpha \cdot \beta$ is the concatenation of α and β, and α^n is the concatenation of n copies of α. We use $\bigodot_{i=1}^{n} \alpha_i$ to denote $\alpha_1 \cdot \alpha_2 \cdot \alpha_3 \cdot \ldots \cdot \alpha_n$.

If α is a string, A and B are sets (with distinct elements), $B \subseteq A$, and the length of α is $|A|$, then the notation $\alpha[^A_B]$ denotes the string of length $|B|$ that is formed by associating successive elements of α with successive members of A, and then selecting from α only those elements that are associated with members of B. For example,

if	$\alpha = 2 \cdot 7 \cdot 4 \cdot 1 \cdot 9 \cdot 8$
and	$A = \{d_{10}, d_{14}, d_{15}, d_{16}, d_{17}, d_{19}\}$
and	$B = \{d_{10}, d_{17}, d_{19}\}$
then	$\alpha[^A_B] = 2 \cdot 9 \cdot 8$

Another notational device is used to select single elements from a string; $\alpha\langle k \rangle$ represents the k^{th} element of α. Formally, $\alpha\langle k \rangle = \alpha[^{\{1,2,3,\ldots,a\}}_{\{k\}}]$ where a is the length of α.

For precision, we define the semantics of computation in a network as the unique string that satisfies the inductive expression

$$Comp_F^A(\sigma) = \sigma \cdot \bigodot_{i=1}^{n} f_i(Comp_F^A(\sigma)[^P_{p(v_i)}])$$

Such a string is unique because A is acyclic and the output of each node is dependent only on the output of previous nodes. The network mapping can now be stated as

$$\mathcal{M}_F^A(\sigma) = Comp_F^A(\sigma)[^P_R]$$

Theorem 1 *$Perf_{SAFns}$ is NP-complete.*

Proof: We reduce the classic satisfiability problem (SAT) to $Perf_{SAFns}$. (See [6] for an explanation of this process.) Let (U, Γ) be an arbitrary instance of SAT, where U is a set of variables and Γ is a set of clauses; $U = \{u_1, u_2, \ldots u_w\}$, $\Gamma = \{(\gamma_i, G_i) : 1 \leq i \leq m\}$. We use a novel representation of Γ, the set of clauses: for each $i \leq m$, $\gamma_i \in \{0,1\}^w$, and $G_i \subseteq U$. A string Ω is said to *satisfy* the instance (U, Γ) iff $\Omega[^U_{G_i}] \neq \gamma_i[^U_{G_i}]$ for all $i \leq m$. (This

representation of a clause can be obtained from the traditional disjunctive form by applying de Morgan's Law once and padding for variables that are not in the clause.)

We must construct an architecture A and a task T such that T is performable by A iff (U, Γ) is satisfiable. The set of nodes V will be composed of a set V_1 of "first-layer nodes" and a set V_2 of "second-layer nodes".

$$S = \{v_{0,i} : 0 \leq i \leq w\}$$

$$V_1 = \{v_{1,j} : 1 \leq j \leq w\}$$

$$V_2 = \{v_{2,i} : 1 \leq i \leq m\}$$

$$P = S \cup V_1 \cup V_2$$

$$R = V = V_1 \cup V_2$$

$$E = \{(v_{0,0}, v_{1,j}), (v_{0,j}, v_{1,j}) : 1 \leq j \leq w\} \cup \{(v_{1,j}, v_{2,i}) : u_j \in G_i\}$$

$$A = (P, V, S, R, E)$$

The task is composed of 3 kinds of items. The first kind is called the "truth-value items" and associates a binary value with 'true' and 'false':

$$T_1 = \{(0 \cdot 0^w, 0^w \cdot *^m), (0 \cdot 1^w, 1^w \cdot *^m)\}$$

The second kind of item is called the "disjunct semantics items":

$$T_2 = \{(0 \cdot \gamma_i, *^w \cdot *^{i-1} \cdot 0 \cdot *^{m-i}) : (\gamma_i, G_i) \in \Gamma\}$$

The third kind of item is called the "conjunct semantics item":

$$T_3 = \{(1 \cdot 0^w, *^w \cdot 1^m)\}$$

$$T = T_1 \cup T_2 \cup T_3$$

Figure 1 gives a construction for an example instance of SAT.

Claim: A solution configuration, F, for (A, T) exists iff a satisfying assignment Ω, exists for (U, Γ).

proof ($\exists F \Leftarrow \exists \Omega$): Assume $(U, \Gamma) \in$ SAT by virtue of the assignment string Ω. With each node $v_{k,i} \in V$, associate the node function $f_{k,i}$ and let $F = \{f_{1,1}, f_{1,2}, \ldots, f_{1,w}, f_{2,1}, f_{2,2}, \ldots, f_{2,m}\}$ where

$$f_{1,j}(a \cdot b) = \begin{cases} b & \text{if } a = 0 \\ \Omega\langle j \rangle & \text{if } a = 1 \end{cases}$$

Take as an example the following SAT problem expressed in traditional CNF form: $(\bar{u_1} \vee u_2 \vee u_3)(u_2 \vee \bar{u_3} \vee \bar{u_4})$. In the required form, this is equivalent to

$$\gamma_1 = 1 \cdot 0 \cdot 0 \cdot 0 \quad G_1 = \{u_1, u_2, u_3\}$$
$$\gamma_2 = 0 \cdot 0 \cdot 1 \cdot 1 \quad G_2 = \{u_2, u_3, u_4\}$$

The task for this problem is

$$\begin{aligned} T_1 = \ & (0 \cdot 0 \cdot 0 \cdot 0 \cdot 0, \quad 0 \cdot 0 \cdot 0 \cdot 0 \cdot * \cdot *) \\ & (0 \cdot 1 \cdot 1 \cdot 1 \cdot 1, \quad 1 \cdot 1 \cdot 1 \cdot 1 \cdot * \cdot *) \\ T_2 = \ & (0 \cdot 1 \cdot 0 \cdot 0 \cdot 0, \quad * \cdot * \cdot * \cdot * \cdot 0 \cdot *) \\ & (0 \cdot 0 \cdot 0 \cdot 1 \cdot 1, \quad * \cdot * \cdot * \cdot * \cdot * \cdot 0) \\ T_3 = \ & (1 \cdot 0 \cdot 0 \cdot 0 \cdot 0, \quad * \cdot * \cdot * \cdot * \cdot 1 \cdot 1) \end{aligned}$$

The architecture is as follows:

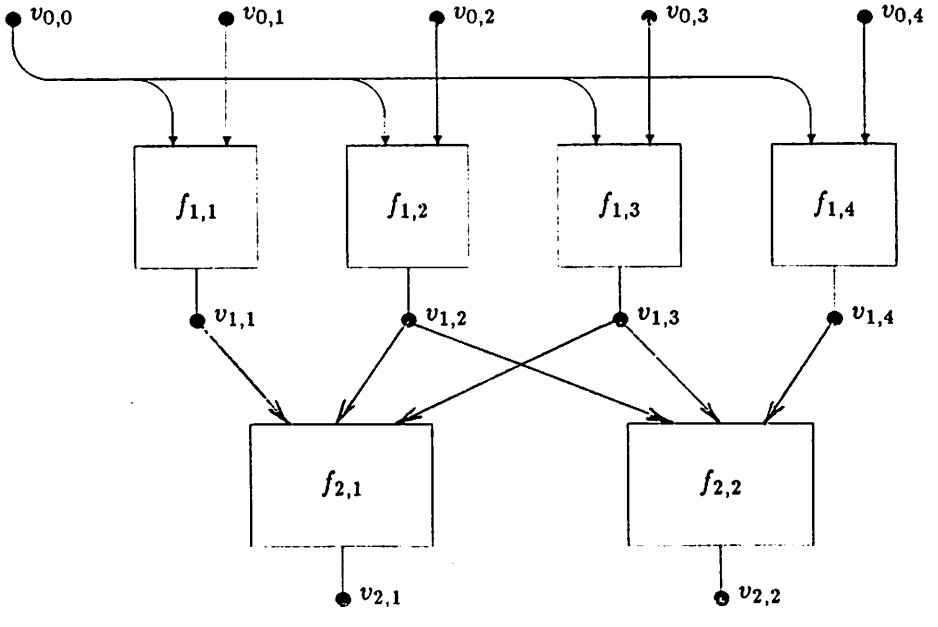

Figure 1: Example Construction for theorem 1

$$f_{2,i}(\alpha) = \begin{cases} 0 & \text{if } \alpha = \gamma_i[^{U}_{G_i}] \\ 1 & \text{otherwise} \end{cases}$$

We must show that $M^A_F \supseteq T$, which we do by showing $M^A_F \supseteq T_1$, $M^A_F \supseteq T_2$, and $M^A_F \supseteq T_3$ individually. First, note that since $f_{1,j}(0 \cdot b) = b$ for all $j \leq w$, we have for any α,

$$Comp^A_F(0 \cdot \alpha)[^{P}_{V_1}] = \bigodot_{j=1}^{w} f_{1,j}((0 \cdot \alpha)[^{S}_{p(v_{1,j})}]) = \bigodot_{j=1}^{w} f_{1,j}(0 \cdot \alpha\langle j \rangle) = \bigodot_{j=1}^{w} \alpha\langle j \rangle = \alpha \tag{1}$$

Equation 1 proves $Comp^A_F(\sigma)[^{P}_{V_1}] = \rho[^{R}_{V_1}]$ for both items $(\sigma, \rho) \in T_1$. Since responses for V_2 are undefined, $M^A_F \supseteq T_1$.

For each $v_{2,i} \in V_2$ there is only one item in T_2 which is defined, and to agree with that response, we must show that $M^A_F(0 \cdot \gamma_i)[^{R}_{V_2}]\langle i \rangle = 0$.

$$\begin{aligned} M^A_F(0 \cdot \gamma_i)[^{R}_{V_2}]\langle i \rangle &= f_{2,i}(Comp^A_F(0 \cdot \gamma_i)[^{P}_{p(v_{2,i})}]) \\ &= f_{2,i}(Comp^A_F(0 \cdot \gamma_i)[^{P}_{V_1}][^{V_1}_{p(v_{2,i})}]) \qquad \text{since } p(v_{2,i}) \subseteq V_1 \\ &= f_{2,i}(\gamma_i[^{V_1}_{p(v_{2,i})}]) \qquad \text{by (1) above} \\ &= f_{2,i}(\gamma_i[^{U}_{G_i}]) \qquad \text{by definition of } E \\ &= 0 \qquad \text{by definition of } f_{2,i}, \text{ as required.} \end{aligned}$$

Since this argument holds for every node in V_2, and responses for V_1 are not defined, $M^A_F \supseteq T_2$.

The only stimulus in T_3 is $1 \cdot 0^w$.

$$Comp^A_F(1 \cdot 0^w)[^{P}_{V_1}] = \bigodot_{j=1}^{w} f_{1,j}((1 \cdot 0^w)[^{S}_{p(v_{1,j})}]) = \bigodot_{j=1}^{w} f_{1,j}(1 \cdot 0) = \bigodot_{j=1}^{w} \Omega\langle j \rangle = \Omega$$

$$\begin{aligned} Comp^A_F(1 \cdot 0^w)[^{P}_{V_2}] &= \bigodot_{i=1}^{m} f_{2,i}((Comp^A_F(1 \cdot 0^w))[^{P}_{V_1}][^{V_1}_{p(v_{2,i})}]) \qquad \text{since } \bigcup_{v \in V_2} p(v) \subseteq V_1 \\ &= \bigodot_{i=1}^{m} f_{2,i}(\Omega[^{V_1}_{p(v_{2,i})}]) \qquad \text{by the previous equation} \\ &= \bigodot_{i=1}^{m} f_{2,i}(\Omega[^{U}_{G_i}]) \qquad \text{by definition of } E \\ &= 1^m \qquad \text{by definition of } f_{2,i} \text{ and } \Omega[^{U}_{G_i}] \neq \gamma_i[^{U}_{G_i}] \end{aligned}$$

So $M_F^A(1 \cdot 0^w) = \Omega \cdot 1^m \models *^w \cdot 1^m$ which is to say $M_F^A \supseteq T_3$. This completes the first half of the claim.

proof ($\exists F \Rightarrow \exists \Omega$): Assume $F = \{f_{1,1}, f_{1,2}, \ldots, f_{1,w}, f_{2,1}, f_{2,2}, \ldots, f_{2,m}\}$ is a configuration such that $M_F^A \supseteq T$. What do we know about F? By inspecting T_1, we know

$$Comp_F^A(0 \cdot 0^w)[_{V_1}^P] = \bigodot_{j=1}^{w} f_{1,j}((0 \cdot 0^w)[_{p(v_{1,j})}^S]) = \bigodot_{j=1}^{w} f_{1,j}(0 \cdot 0) = 0^w$$

by the first item. Hence $f_{1,j}(0 \cdot 0) = 0$. By the second item, we can similarly show $f_{1,j}(0 \cdot 1) = 1$, which leads us to conclude what was shown in equation (1).

By inspecting T_2 and T_3, we have for every i, $1 \leq i \leq m$

$$f_{2,i}(Comp_F^A(0 \cdot \gamma_i)[_{p(v_{2,i})}^P]) = 0 \neq 1 = f_{2,i}(Comp_F^A(1 \cdot 0^w)[_{p(v_{2,i})}^P])$$

$$Comp_F^A(0 \cdot \gamma_i)[_{p(v_{2,i})}^P] \neq Comp_F^A(1 \cdot 0^w)[_{p(v_{2,i})}^P]$$

Applying equation (1) and the definition of E on the l.h.s.,

$$\text{l.h.s.} \; = Comp_F^A(0 \cdot \gamma_i)[_{V_1}^P][_{p(v_{2,i})}^{V_1}] = \gamma_i[_{p(v_{2,i})}^{V_1}] = \gamma_i[_{G_i}^U]$$

Simplifying the r.h.s. by letting $\Omega = Comp_F^A(1 \cdot 0^w)[_{V_1}^P]$,

$$\text{r.h.s.} \; = Comp_F^A(1 \cdot 0^w)[_{V_1}^P][_{p(v_{2,i})}^{V_1}] = \Omega[_{p(v_{2,i})}^{V_1}] = \Omega[_{G_i}^U]$$

Reassembling, we have $\gamma_i[_{G_i}^U] \neq \Omega[_{G_i}^U]$ for all $i, 1 \leq i \leq m$ which is to say that Ω satisfies (U, Γ) and the claim is proved.

Thus we have SAT $\propto Perf_{SAFns}$ and it is easy to see that the algorithm for the transformation runs in polynomial time (in fact linear time and log space).

Finally, it must be demonstrated that there is a non-deterministic machine that can decide $Perf_{SAFns}$ in time polynomial in the length of (A, T). That is, there must be a poly-time method of writing down a valid SAFns configuration and checking that it is correct. Writing down a function from SAFns requires one bit for every nodal input (to specify whether it should be inverted before entering the AND gate), and one bit for the output (to specify whether the whole function should be inverted). For the complete configuration, this takes one bit for each edge in A and one bit for each node in A. That the configuration is correct can be checked by evaluating each

node function once for each item in T. This takes time $O(|V| \times |T|)$ under the assumption that it takes constant time to evaluate any single f_i.

This, and SAT $\propto Perf_{SAFns}$ implies $Perf_{SAFns}$ is *NP*-complete. □

Corollary 2 *For any node function set $\mathcal{F}$ such that all members of F are binary-valued functions, and $\mathcal{F} \supseteq$ SAFns, $Perf_{\mathcal{F}}$ is NP-hard.*

Proof: Both directions of the proof of the claim in theorem 1 require nodes able, at least, to perform functions from SAFns. The reduction thus follows for any node function set that includes them. □

Corollary 3 *$Perf_{LSFns}$ is NP-complete.*

Proof: *NP*-hardness follows from corollary 2, but to be in *NP*, there must be some poly-time way of guessing a function from LSFns and being sure that indeed it *is* from LSFns. If fan-in were bounded in our model, then this would be easy since we could get the non-deterministic selection to be from a fixed table of all LSFns up to that input size. Without bounds on fan-in, this technique will not work. One might attempt to achieve a selection from LSFns by simply writing down the weights that are used in the linear sum, but since the weights are assumed to be *real* (i.e. of a potentially infinite number of decimal places), this technique also is troubled. However, Hong [9] has recently proved that approximations to the weights are sufficient to encode any and all members of LSFns. Specifically, only a polynomial number of digits are required (polynomial in the fan-in), and hence $Perf_{LSFns}$ is *NP*-complete. □

Corollary 4 *$Perf_{LUFns}$ is NP-complete.*

Proof: Again, *NP*-hardness follows from corollary 2, but to be in *NP*, there must be some poly-time way of writing down an arbitrary member of LUFns. (Making sure that it *is* a member of LUFns is trivial since *all* binary-valued functions are members.) Without a bound on fan-in, writing down an arbitrary member of LUFns takes exponential time since it requires $2^{|p(v_i)|}$ bits to fully specify it. However, each node function will be invoked exactly $t = |T|$ times in the performance of the task; hence we can specify a function $F \in$LUFns by asserting a default value (1, say) to cover most inputs, and

then listing the exceptional inputs α for which $F(\alpha) = 0$ (of which there are at most t). Since T has a unary encoding of t, there is a representation of F that is polynomial in the length of (A, T). Hence $Perf_{\mathcal{F}} \in NP$ even when $\mathcal{F}$ = LUFns, and $Perf_{LUFns}$ is NP-complete. □

The next theorem extends the previous one to the case of certain real-valued node functions. We consider a function set used in [17] wherein every member of the set is a function composed of two parts. The first part is the logistic function and the second is a linear weighted sum of its inputs.

$$f(\alpha) = E(e(\alpha))$$

$$\text{where } e(\alpha) = w_0 + \sum w_i \times \alpha\langle i \rangle,$$

$$\text{and } E(x) = \frac{1}{1 + e^{-x}}.$$

We call these functions LLFns (for Logistic Linear Functions). The E function is fixed for all nodes, so to specify a member of LLFns it is enough to specify the weights $w_0, w_1, \ldots$ used in e.

Following [17] again, we say that a value agrees with 1 if it is no smaller than 0.9, and it agrees with 0 if it is no larger than 0.1. Note that $E(x)$ asymptotically approaches 1 as x approaches $+\infty$, and that $E(x)$ asymptotically approaches 0 as x approaches $-\infty$. Let d be some scalar value. We say that α *agrees for high* d *with* β (written $\alpha \models^{d} \beta$) if there is some value for d beyond which α always agrees with β. This implies that the value of α or β is a function of d.

$$\alpha(d) \models^{d} \beta(d) \Leftrightarrow \exists d_0 \text{ such that } \alpha(d) \models \beta(d) \text{ for all } d \geq d_0$$

Such agreement is easy to prove if α is monotonic in d and β is constant.

Note that if two such agreement statements hold for the same high parameter then they hold simultaneously for that parameter.

$$(\alpha \models^{d} \beta \text{ and } \delta \models^{d} \xi) \Leftrightarrow \alpha \cdot \delta \models^{d} \beta \cdot \xi$$

A new notational device is used to select single elements from a string in the case where the element's position in a string is not known except through its relative position in one of the clause sets, G_i, in Γ. For that situation, we use $\frac{i}{k}$ to mean the index in U of the k^{th} element of clause i. Formally, $\frac{i}{k} = (\bigodot_{j=1}^{w} j)[{}^{U}_{G_i}]\langle k \rangle$. Consequently, this identity holds: $\alpha\langle \frac{i}{k} \rangle = \alpha[{}^{U}_{G_i}]\langle k \rangle$.

Theorem 5 *$Perf_{LLFns}$ is NP-complete.*

Proof: We construct a performability problem (A, T) where the architecture, A, is the same as it was in the proof of theorem 1 except that $R = V_2$ instead of $R = V$, and the task, T, is as follows:

$$T = T_1 \cup T_2 \cup T_3$$

$$T_1 = \{(0 \cdot \gamma_i, \quad *^{i-1} \cdot 0 \cdot *^{m-i}) : \ 1 \leq i \leq m\}$$

$$T_2 = \{\{(0 \cdot \gamma_i^{(k)}, \quad *^{i-1} \cdot 1 \cdot *^{m-i}) : \ 1 \leq k \leq |G_i|\} : \ 1 \leq i \leq m\}$$

$$T_3 = \{(1 \cdot \gamma_i, \quad *^{i-1} \cdot 1 \cdot *^{m-i}) : \ 1 \leq i \leq m\}$$

where $\gamma_i^{(k)}$ is γ_i with the k^{th} relevant bit inverted:

$$\gamma_i^{(k)}\langle j \rangle = \begin{cases} \gamma_i\langle j \rangle & \text{if } j \neq \frac{i}{k} \\ 1 - \gamma_i\langle j \rangle & \text{otherwise} \end{cases}$$

Claim: There exists a solution configuration F to (A, T) iff there exists a solution assignment Ω to (U, Γ). For both directions of the proof we shall use the following definitions (they each stand for the computation performed by the first layer of nodes when the net is given some stimulus in the task):

$$\varsigma_i = Comp_F^A(0 \cdot \gamma_i)[_{V_1}^P] \quad (\text{from } T_1)$$

$$\beta_i^{(j)} = Comp_F^A(0 \cdot \gamma_i^{(j)})[_{V_1}^P] \quad (\text{from } T_2)$$

$$\eta_i = Comp_F^A(1 \cdot \gamma_i)[_{V_1}^P] \quad (\text{from } T_3)$$

proof $(\exists F \Leftarrow \exists \Omega)$: Specify the node functions as follows:

$$f_{1,j}(a \cdot b) = \begin{cases} E(-d + 2da + 2db) & \text{if } \Omega\langle j \rangle = 1 \\ E(-d - 2da + 2db) & \text{if } \Omega\langle j \rangle = 0 \end{cases}$$

$$f_{2,i}(\alpha) = E(e_{2,i}(\alpha))$$

where

$$e_{2,i}(\alpha) = -d + 2d \sum_{k=1}^{|G_i|} W_{i,k} \times (\varsigma_i\langle \tfrac{i}{k} \rangle - \alpha\langle k \rangle)$$

$$W_{i,k} = \begin{cases} +1 & \text{if } \gamma_i\langle \frac{i}{k} \rangle = 1 \\ -1 & \text{if } \gamma_i\langle \frac{i}{k} \rangle = 0 \end{cases}$$

The above expression for $e_{2,i}$ is not in standard form but it is straightforward to rearrange it so that it is.

We shall check that each subtask is performed correctly by this configuration. Observe

$$f_{1,j}(0 \cdot 0) = E(-d + 0 + 0) \models^d 0$$

$$f_{1,j}(0 \cdot 1) = E(-d + 0 + 2d) \models^d 1$$

Hence $Comp_F^A(0 \cdot \alpha)[{}^P_{V_1}] \models^d \alpha$. Consequently $\varsigma_i \models^d \gamma_i$. Also

$$f_{2,i}(\varsigma_i[{}^{V_1}_{p(v_{2,i})}]) = E(-d + 2d \sum_k W_{i,k}(\varsigma_i\langle \tfrac{i}{k} \rangle - \varsigma_i\langle \tfrac{i}{k} \rangle)) \models^d 0$$

The agreement holds because the total value of the summation is 0. This argument applies to each value of i, and hence for high d, $M_F^A \supseteq T_1$.

Consider a typical item in T_2. Note that $\beta_i^{(k)} \models^d \gamma_i^{(k)}$, and that $\beta_i^{(k)}\langle {}^i_k \rangle$ therefore differs from $\gamma_i\langle \frac{i}{k} \rangle$ as d increases. The absolute difference converges monotonically to 1, so we have

$$f_{2,i}(\beta_i^{(k)}[{}^{V_1}_{p(v_{2,i})}]) = E(-d + 2d \sum W_{i,k}(\varsigma_i\langle \tfrac{i}{k} \rangle - \beta_i^{(k)}\langle {}^i_k \rangle)) \models^d 1$$

Here we know the agreement for high d holds because the total value of the summation tends to 1 as d increases. Since the equation is valid for all $1 \leq k \leq |G_i|$ for each γ_i, $M_F^A \supseteq T_2$ for high d.

Next we consider a typical item in T_3. For all nodes in layer 1, observe

$$\text{if } \Omega\langle j \rangle = 1, \quad f_{1,j}(1 \cdot x) = E(-d + 2d + 2dx) \models^d 1 \quad \text{for } x \in \{0, 1\}$$

$$\text{if } \Omega\langle j \rangle = 0, \quad f_{1,j}(1 \cdot x) = E(-d - 2d + 2dx) \models^d 0 \quad \text{for } x \in \{0, 1\}$$

Hence $f_{1,j}(1 \cdot x) \models^d \Omega\langle j \rangle$ and consequently $\eta_i \models^d \Omega$ for all i. Examining the second layer, we know

$$f_{2,i}(\eta_i[{}^{V_1}_{p(v_{2,i})}]) = E(-d + 2d \sum W_{i,k}(\varsigma_i\langle \tfrac{i}{k} \rangle - \eta_i\langle \tfrac{i}{k} \rangle)) \models^d 1$$

because as d increases the summation converges to some integer representing the number of places where $\varsigma_i[{}^U_{G_i}]$ is not equal to $\eta_i[{}^U_{G_i}]$, that is, the number of places where $\gamma_i[{}^U_{G_i}]$ is not equal to $\Omega[{}^U_{G_i}]$. By the initial assumption about Ω, this integer is at least 1, so the agreement holds (for high d). This demonstrates that $M_F^A \supseteq T_3$ for high d.

By selecting some value for d which satisfies all the above agreements, $M_F^A \supseteq T$ and this completes the proof of one direction of the claim.

proof ($\exists F \Rightarrow \exists \Omega$): Let $y_{j,k}$ and $z_{i,k}$ be the weights employed in the node functions as follows: for all $i, j,\ 1 \leq i \leq m,\ 1 \leq j \leq w$, let

$$f_{1,j}(a \cdot b) = E(y_{j,0} + y_{j,1}a + y_{j,2}b)$$

$$f_{2,i}(\alpha) = E(z_{i,0} + \sum_{k=1}^{|G_i|} z_{i,k}\alpha\langle k\rangle)$$

Define the satisfying assignment:

$$\Omega\langle j\rangle = \begin{cases} 1 & \text{if } y_{j,1}y_{j,2} > 0 \\ 0 & \text{otherwise} \end{cases}$$

We must show Ω satisfies (U, Γ).

By assumption, the configuration F performs T_1 and T_2, so we know for each i, $1 \leq i \leq m$ and for any k, $1 \leq k \leq |G_i|$

$$f_{2,i}(\varsigma_i|^{V_1}_{p(v_{2,i})}]) \models 0$$

$$f_{2,i}(\beta_i^{(k)}|^{V_1}_{p(v_{2,i})}]) \models 1$$

so

$$f_{2,i}(\varsigma_i|^{V_1}_{p(v_{2,i})}]) < f_{2,i}(\beta_i^{(k)}|^{V_1}_{p(v_{2,i})}])$$

$$E(z_{i,0} + \sum_c z_{i,c}\,\varsigma_i\langle\tfrac{i}{c}\rangle) < E(z_{i,0} + \sum_c z_{i,c}\,\beta_i^{(k)}\langle\tfrac{i}{c}\rangle)$$

but $\varsigma_i\langle j\rangle = \beta_i^{(k)}\langle j\rangle$ for all $j \neq \frac{i}{k}$, or more specifically $\varsigma_i\langle\frac{i}{c}\rangle \neq \beta_i^{(k)}\langle\frac{i}{c}\rangle$ only when $c = k$. Therefore

$$z_{i,k}\,\varsigma_i\langle\tfrac{i}{k}\rangle < z_{i,k}\,\beta_i^{(k)}\langle\tfrac{i}{k}\rangle$$

Let $j = \frac{i}{k}$ and expand both sides in terms of $f_{1,j}$.

$$z_{i,k}E(y_{j,0} + y_{j,2}\,\gamma_i\langle j\rangle) < z_{i,k}E(y_{j,0} + y_{j,2}(1 - \gamma_i\langle j\rangle))$$

$$z_{i,k}\,y_{j,2}\,\gamma_i\langle\tfrac{i}{k}\rangle < z_{i,k}\,y_{j,2}\,(1 - \gamma_i\langle\tfrac{i}{k}\rangle)$$

$$\text{if } \gamma_i\langle j\rangle = 0 \text{ then } 0 < z_{i,k}\,y_{j,2} \tag{2}$$

$$\text{if } \gamma_i\langle j\rangle = 1 \text{ then } z_{i,k}\,y_{j,2} < 0 \tag{3}$$

Again, by assumption that F configures the net for T_1 and T_2,

$$f_{2,i}(\varsigma_i[^{V_1}_{p(v_{2,i})}] \models 0$$

$$f_{2,i}(\eta_i[^{V_1}_{p(v_{2,i})}] \models 1$$

$$E(z_{i,0} + \sum_k z_{i,k}\, \varsigma_i\langle \tfrac{i}{k}\rangle) < E(z_{i,0} + \sum_k z_{i,k}\, \eta_i\langle \tfrac{i}{k}\rangle)$$

$$\sum_k z_{i,k}\, \varsigma_i\langle \tfrac{i}{k}\rangle < \sum_k z_{i,k}\, \eta_i\langle \tfrac{i}{k}\rangle$$

For this to be true for a given i, there must be at least one k such that

$$z_{i,k}\, \varsigma_i\langle \tfrac{i}{k}\rangle < z_{i,k}\, \eta_i\langle \tfrac{i}{k}\rangle$$

Letting $j = \frac{i}{k}$ and expanding both sides as $f_{1,j}$,

$$z_{i,k}\, E(y_{j,0} + y_{j,2}\, \gamma_i\langle j\rangle) < z_{i,k}\, E(y_{j,0} + y_{j,1} + y_{j,2}\, \gamma_i\langle j\rangle)$$

$$0 < z_{i,k}\, y_{j,1}$$

From this and (2), we find that

$$\gamma_i\langle j\rangle = 0 \Rightarrow 0 < z_{i,k}\, z_{i,k}\, y_{j,1}\, y_{j,2} \Rightarrow 0 < y_{j,1}\, y_{j,2} \Rightarrow \Omega\langle j\rangle = 1$$

Similarly, $\gamma_i\langle j\rangle = 1 \Rightarrow \Omega\langle j\rangle = 0$. Summarizing, for all γ_i there exists a k such that $\gamma_i\langle \frac{i}{k}\rangle \neq \Omega\langle \frac{i}{k}\rangle$, or rather $\gamma_i[^U_{G_i}] \neq \Omega[^U_{G_i}]$. That is, Ω satisfies (U, Γ) and the claim is proved.

The claim establishes that the reduction from SAT is valid. Since the transformation can be performed in polynomial time, $Perf_{LLFns}$ is *NP*-hard.

$Perf_{LLFns}$ is in *NP* if there is a polynomial-time procedure to write down values for all the weights. For the case where the weights are truly real-valued (meaning that a weight would have a potentially infinite number of digits), it has not yet been proven that there is a finite approximation that is effectively equivalent to the real numbers (as Hong has done for LSFns). However, for the more realistic case of fixed resolution in each 'real' weight, specifying the configuration is easily performed in polynomial time. With that minor caveat, we have proved $Perf_{LLFns}$ is *NP*-complete. □

Three aspects of LLFns are crucial to the preceding proof: E is monotonic, E is bounded, and e is linear. Other aspects were convenient but not necessary; for example, every node had a fixed E function, every node had the *same* E function, and that E was onto the unit interval $[0, 1]$. We proved the theorem for LLFns only in order to avoid excessive abstraction, but the theorem is extendable to other node function sets.

If we define the quasi-linear functions (QLFns) as all those functions of the form $E(e(\alpha))$ where e is linear and E is a bounded and monotonic, then for some appropriate definition of agreement we have

Corollary 6 *$Perf_{QLFns}$ is NP-complete.*

The theorem is probably extendable to different manifestations of non-linearity, but we note that something about E should be non-linear, for if E (as well as e) is linear, then the net as a whole can implement only linear mappings. From the point of view of connectionists, this is uninteresting.

References

[1] D. H. Ackley, G. E. Hinton, and T. J. Sejnowski. A learning algorithm for Boltzmann machines. *Cognitive Science*, 9:147–169, 1985.

[2] F. Barahona. On the computational complexity of Ising spin glass models. *Journal of Physics A: Math. Gen.*, 3241–3253, 1982.

[3] A. G. Barto. Learning by statistical cooperation of self-interested neuron-like computing elements. *Human Neurobiology*, 4:229–256, 1985.

[4] A. G. Barto and P. Anandan. Pattern recognizing stochastic learning automata. *IEEE Transactions on Systems, Man, and Cybernetics*, 15:360–375, 1985.

[5] Y. L. Cun. Une procedure d'apprentissage pour reseau a sequil assymetrique. *Proceedings of Cognitiva*, 85:599–604, 1985.

[6] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W. H. Freeman, San Fransisco, 1979.

[7] S. E. Hampson and D. J. Volper. Linear function neurons: Structure and training. *Biological Cybernetics*, 53:203–217, 1986.

[8] G. E. Hinton and T. J. Sejnowski. Learning and relearning in Boltzmann machines. In D. E. Rumelhart and J. L. McClelland, editors, *Parallel Distributed Processing: Explorations in the Microstructure of Cognition, vol.1: Foundations*, chapter 7, Bradford Books/MIT Press, Cambridge, MA., 1986.

[9] J. Hong. *On Connectionist Models.* Technical Report, Dept. Computer Science, University of Chicago, Chicago, Ill., U.S.A., May 1987.

[10] M. Kearns, M. Li, L. Pitt, and L. Valiant. On the learnability of boolean formulae. In *Proceedings of the Symposium on Theory of Computing*, 1987.

[11] M. Minsky and S. Papert. *Perceptrons: An Introduction to Computational Geometry.* MIT Press, Cambridge, Mass., 1972.

[12] S. M. Omohundro. *Efficient Algorithms with Neural Network Behaviour.* Technical Report UIUCDCS-R-87-1331, Dept. Computer Science, University of Illinois at Urbana-Champaign, 1304 W. Springfield Ave., Urbana, Il 61801, U.S.A., April 1987.

[13] D. B. Parker. *Learning Logic.* Technical Report TR-47, Massachusetts Institute of Technology, 1985.

[14] S. Porat. *Stability and Looping in Connectionist Models with Asymmetric Weights.* Technical Report TR 210, Computer Science Dept., University of Rochester, Rochester, N.Y. 14627, March 1987.

[15] F. Rosenblatt. *Principles of Neurodynamics: Perceptrons and the Theory of Brain Mechanisms.* Spartan Books, 6411 Chillum Place n.w., Washington, D.C., 1961.

[16] D. E. Rumelhart, G. E. Hinton, and J. L. McClelland. A general framework for parallel distributed processing. In D. E. Rumelhart and J. L. McClelland, editors, *Parallel Distributed Processing: Explorations in the Microstructure of Cognition, vol.2: Psychological and Biological Models*, chapter 2, Bradford Books/MIT Press, Cambridge, MA., 1986.

[17] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning internal representations by error propagation. In D. E. Rumelhart and J. L. McClelland, editors, *Parallel Distributed Processing: Explorations in the Microstructure of Cognition, vol.1: Foundations*, Bradford Books/MIT Press, Cambridge, MA., 1986.

[18] G. Tesauro. Scaling relationships in back-propogation learning: Dependence on training set size. *Complex Systems*, 1:367–372, 1987.

[19] L. G. Valiant. A theory of the learnable. *Communications of the ACM*, 27(11):1134–1142, November 1984.

[20] R. J. Williams. The logic of activation functions. In D. E. Rumelhart and J. L. McClelland, editors, *Parallel Distributed Processing: Explorations in the Microstructure of Cognition, vol.1: Foundations*, Bradford Books/MIT Press, Cambridge, MA., 1986.