10

ON THE INTRACTABILITY OF LOADING NEURAL NETWORKS

Bhaskar DasGupta, Hava T. Siegelmann*, and Eduardo Sontag**

Dept. of Computer Science, University of Minnesota,
Minneapolis, MN 55455-0159. Email: dasgupta@cs.umn.edu

** Dept. of Computer Science, Rutgers University, New Brunswick, NJ 08903. Email: siegelma@paul.rutgers.edu*

*** Dept. of Mathematics, Rutgers University, New Brunswick, NJ 08903. Email: sontag@control.rutgers.edu*

1 INTRODUCTION

Neural networks have been proposed as a tool for machine learning. In this role, a network is trained to recognize complex associations between inputs and outputs that were presented during a supervised training cycle. These associations are incorporated into the weights of the network, which encode a distributed representation of the information that was contained in the patterns. Once trained, the network will compute an input/output mapping which, if the training data was representative enough, will closely match the unknown rule which produced the original data. Massive parallelism of computation, as well as noise and fault tolerance, are often offered as justifications for the use of neural nets as learning paradigms.

By "neural network" we always mean, in this chapter, feedforward ones of the type routinely employed in artificial neural nets applications. That is, a net consists of a number of processors ("nodes" or "neurons") each of which computes a function of the type

$$y = \sigma\left(\sum_{i=1}^{k} a_i u_i + b\right) \tag{10.1}$$

of its inputs $u_1, \ldots, u_k$. These inputs are either external (input data is fed through them) or they represent the outputs of other nodes. No cycle is allowed in the connection graph (i.e., feedforward nets rather than "recurrent" nets are

V. Roychowdhury et al. (eds.), *Theoretical Advances in Neural Computation and Learning*

considered) and the output of one designated node is understood to provide the output value produced by the entire network for a given vector of input values. The possible coefficients a_i and b appearing in the different nodes are the *weights* of the network, and the functions σ appearing in the various nodes are the *node* or *activation* functions. An *architecture* specifies the interconnection structure and the σ's, but not the actual numerical values of the weights themselves.

This chapter deals with basic *theoretical* questions regarding learning by neural networks. There are three types of such questions that one may ask, all closely related and complementary to each other. We next describe all three, keeping for the end the one that is the focus of this chapter.

One line of work deals with *sample complexity* questions, which attempt at quantifying the amount of information (number of samples) needed in order to characterize a given unknown mapping. Some recent references to such work, establishing sample complexity results, and hence "weak learnability" in the Valiant model, for neural nets, are the papers [2, 19, 11, 18]; the first of these references deals with networks that employ hard threshold activations, the second and third cover continuous activation functions of a type (piecewise polynomial) close to those used in this chapter, and the last one provides results for networks employing the standard sigmoid activation function.

Another line of work adopts an approximation theoretic point of view. In particular, one asks questions such as *how many* hidden units are necessary in order to approximate well, that is to say, with a small overall error, an unknown function. This type of research ignores the training question itself, asking instead what is the best one could do, in this sense of overall error, if the best possible network with a given architecture were to be eventually found. Some recent papers along these lines are [1, 12, 5], which deal with single hidden layer nets, and [6], which dealt with multiple hidden layers.

Yet another direction of research on learning by neural networks, and the one that concerns us here, originates with the work of Judd (see for instance [13, 14], as well as the related work [3, 17, 30]). Judd, like us, was motivated by the observation that the "backpropagation" algorithm often runs very slowly, especially for high-dimensional data. Recall that this algorithm is used in order to find a network (that is, find the weights, assuming a fixed architecture) that reproduces the observed data. Of course, many modifications of the vanilla "backprop" approach are possible, using more sophisticated techniques such as high-order (Newton), conjugate gradient, or sequential quadratic programming methods. However, the "curse of dimensionality" seems to arise as a computational obstruction to all these training techniques as well, when attempting to learn arbitrary data using a standard feedforward network. For the simpler case of linearly separable data, the perceptron algorithm and linear programming techniques help to find a network -with no "hidden units"- relatively fast. Thus one may ask if there exists a *fundamental barrier* to training by general feedforward networks, a barrier that is insurmountable no matter which particular algorithm one uses. (Those techniques which *adapt* the architecture to the data, such as cascade correlation or incremental techniques, would not be subject to such a barrier.)

In this chapter, we consider the tractability of the training problem, that is, of the question (essentially quoting Judd): "Given a network architecture (interconnection graph as well as choice of activation function) and a set of training examples, does there exist a set of weights so that the network produces the correct output for all examples?"

The simplest neural network, i.e., the perceptron, consists of one threshold neuron only. It is easily verified that the computational time of the loading problem in this case is polynomial in the size of the training set irrespective of whether the input takes continuous or discrete values. This can be achieved via a linear programming technique (see [25] for further results in this direction). On the other-hand, loading recurrent networks (i.e. networks with feedback loops) is a hard problem. In [26], Siegelmann and Sontag showed the existence of a finite size recurrent network made of a specific saturated linear neurons which is Turing universal. Thus, the loading problem is undecidable for such nets. Furthermore, in [27], they showed that if real numbers are allowed in the weights of these specific networks (rather than rational ones) the network is equivalent to a non-uniform version of Turing machines (i.e. Turing machine with advice) which is stronger than the common model. Kilian and Siegelmann [16] proved universality for the sigmoidal network and a large class of sigmoidal-type nets. They concluded that Turing-universality is a common property among recurrent nets (and not only for the specific case of the saturated linear function). A different power is demonstrated by the recurrent threshold nets. It was proved in [23] that the problem of determining whether a recurrent network with threshold units (that is, the number of states in the network is finite) has a stable configuration is NP-hard. Bruck and Goodman [4] showed that a recurrent threshold network of polynomial size cannot solve NP-complete problems unless NP=*co*-NP. The result was further extended by Yao [29] who showed that a polynomial size threshold recurrent network cannot solve NP-complete problems even approximately within a guaranteed performance ratio unless NP=*co*-NP.

In the rest of this chapter, we focus on feedforward nets only. We show that, for networks employing a simple, saturated piecewise linear activation function, and two hidden units only, the loading problem is NP-complete. Recall that if one establishes that a problem is NP-complete then one has shown, in the standard way done in computer science, that the problem is at least as hard as most problems widely believed to be hard (the "traveling salesman" problem, Boolean satisfiability problem, and so forth). This shows that, indeed, *any possible* neural net learning algorithm (for this activation function) based on fixed architectures faces severe computational barriers. Furthermore, our result implies non-learnability in the PAC sense under the complexity-theoretic assumption of $RP \neq NP$. We generalize our result to another similar architecture.

The work most closely related to ours is that due to Blum and Rivest; see [3]. They showed a similar NP-completeness result for networks having the same architecture but where the activation functions are all of a hard threshold type, that is, they provide a binary output y equal to 1 if the sum in equation (10.1) is positive, and 0 otherwise. In their papers, Blum and Rivest explicitly pose as an open problem the question of establishing NP-completeness, for this architecture, when the activation function is "sigmoidal" and they conjecture that this is indeed the case.

It turns out that a definite answer to the question posed by Blum and Rivest is not possible. It is shown in [28] that for *certain* activation functions σ, the problem can be solved in constant time, independently of the input size, and hence the question is *not* NP-complete. In fact, there exist "sigmoidal" functions, innocent-looking qualitatively (bounded, infinite differentiable and even analytic, and so forth) for which any set of data can be loaded, and hence for which the loading problem is *not* NP-complete. The functions used in the construction in [28] are however extremely artificial and in no way likely to appear in practical implementations. Nonetheless, the mere *existence* of such examples means that the mathematical question is far from trivial.

The main open question, then, is to understand if "reasonable" activation functions lead to NP-completeness results similar to the ones in the work by Blum and Rivest or if they are closer to the other extreme, i.e., the purely mathematical construct in [28]. The most puzzling case is that of the standard sigmoid function, $1/(1+e^{-x})$. For that case we do not know the answer yet, but we conjecture that NP-completeness will indeed hold. It is the purpose of this chapter to show an NP-completeness result for piecewise linear or "saturating" activation function that has appeared in the neural networks literature, especially in the context of hardware implementations, and which is relatively simpler to analyze than the standard sigmoid.

We view our result as a first step in dealing with the general case of arbitrary piecewise linear functions, and as a further step towards elucidating the complexity of the learning problem in general.

The rest of the chapter is organized as follows:

- In section 2 we introduce the model and summarize some previous results. We also distinguish the case of fixed versus varying input dimensions (and analog versus binary inputs), and observe that the problem is solvable in polynomial time when the input dimension is fixed using standard linear-programming techniques (see [19] for further positive results on *PAC*-learnability when the input dimension is a fixed constant and the activation functions are piecewise polynomials). In the remaining part of the chapter we concentrate on the case when the input dimension is *not* constant.
- In section 3 we prove the hardness of the loading problem for the 2 π-node architecture and use this result to show the impossibility of learnability for varying input dimension under the assumption of $RP \neq NP$.
- In section 4 we conclude with some open problems.

Before turning to the next section, we provide a short overview on complexity classes and probabilistic learnability. Most of these concepts have been already introduced in Chapters 7 and 8, and readers familiar with the material are recommended to skip to Section 2. 4

1.1 Some complexity classes

We informally discuss some well known structural-complexity classes (the reader is referred to any standard text on structural complexity classes (e.g. [9, 10]) for more details). Here, whenever we say polynomial time we mean polynomial time in the length of any reasonable encoding of the input (see [9] for a discussion of a "reasonable" encoding of the inputs), and problems referred to here are always *decision* problems.

A problem is in the class P when there is a polynomial time algorithm which solves the problem. A problem is in NP when a "guessed" solution for the problem can be verified in polynomial time. A problem X is NP-hard iff any problem Y in NP can be transformed in polynomial time f to X, such that given an instance I of Y, I has a solution iff $f(I)$ has a solution. A problem is NP-complete iff it is both NP and NP-hard. Examples of NP-complete problems include the traveling salesperson problem, the Boolean satisfiability problem and the set-splitting problem.

A problem X is in the complexity class RP ("random polynomial") with *error parameter* ϵ if and only if there is a polynomial time algorithm A such that for every instance I of X the following holds:

> If I is a "yes" instance of X then A outputs "yes" with probability at least $1 - \epsilon$ for some constant $0 < \epsilon < 1$, and if I is a "no" instance of X then A always outputs "no".

It is well known that $P \subseteq RP \subseteq NP$, but whether any of the inclusions is proper is an important open question in structural complexity theory.

1.2 Probabilistic learnability

Let $n \in \mathcal{N}$. A concept is a function $f : \{0,1\}^n \rightarrow \{0,1\}$. We focus on functions computable by architectures (defined in section 2.2); hence, we use the terms function and architecture interchangeably. The set of inputs $f^{-1}(0) = \{x \mid x \in \{0,1\}^n, f(x) = 0\}$ is the set of negative examples, where the set of inputs $f^{-1}(1) = \{x \mid x \in \{0,1\}^n, f(x) = 1\}$ is the set of positive examples.

Let C_n be the set Boolean functions on n variables defined by a specific architecture $\mathcal{A}$. Then $C = \cup_{i=1}^{\infty} C_n$ is a class of representations achievable by the architecture $\mathcal{A}$ for all binary input strings. For example, C may be the class of Boolean formulae computable by one hidden-layer net with two sigmoidal hidden units and threshold output unit. Given some function $f \in C$, $POS(f)$ (resp. $NEG(f)$) denotes the source of positive (resp. negative) examples for f. Whenever $POS(f)$ (resp. $NEG(f)$) is called, a positive or $'+'$ (resp. negative or $'-'$) example is provided according to some arbitrary probability distribution D^+ (resp. D^-) satisfying the condition:

$$\sum_{x=f^{-1}(1)} D^+(x) = 1,$$

and

$$\sum_{x=f^{-1}(0)} D^-(x) = 1.$$

A *learning algorithm* is an algorithm that may access $POS(f)$ and $NEG(f)$. Each access to $POS(f)$ or $NEG(f)$ is counted as one step. A class C of representations of an architecture $\mathcal{A}$ is said to be (ϵ, δ)*-learnable* iff, for some given fixed constants $0 < \epsilon, \delta < 1$, there is a learning algorithm L such that for all $n \in \mathcal{N}$, all functions $f \in C_n$, and all possible distributions D^+ and D^-:

(a) L halts in number of steps polynomial in n, $\frac{1}{\epsilon}$, $\frac{1}{\delta}$, and $||\mathcal{A}||$ (where $||\mathcal{A}||$ denotes the size of the architecture $\mathcal{A}$), and

(b) L outputs a hypothesis $g \in C_n$ such that with probability at least $1 - \delta$ the following conditions are satisfied:

$$\sum_{x \in g^{-1}(0)} D^+(x) < \epsilon,$$

and

$$\sum_{x \in g^{-1}(1)} D^-(x) < \epsilon.$$

A class C of representations of an architecture $\mathcal{A}$ is said to be *learnable* [15] iff it is (ϵ, δ)-learnable for all ϵ and δ (where $0 < \epsilon, \delta < 1$).

Remark 10.1 *Hence, to prove that a class of representations of an architecture $\mathcal{A}$ is not learnable, it is sufficient to prove that it is not (ϵ, δ)-learnable for some particular values of ϵ and δ, and some particular distributions D^+ and D^-.*

As we will see later, our results on NP-completeness of the loading problem will imply a non-learnability of the corresponding concept under the assumption of $RP \neq NP$.

2 PRELIMINARIES AND PREVIOUS WORK

In this section we define our model of computation precisely and state some previous results for this model.

2.1 Feedforward networks and the loading problem

Let Φ be a class of real-valued functions, where each function is defined on some subset of $\mathbb{R}$. A Φ-net C is an unbounded fan-in directed acyclic graph. To each vertex v, an activation function $\phi_v \in \Phi$ is assigned, and we assume that C has a single sink w.

The network C computes a function $f_C : [0,1]^n \rightarrow \mathbb{R}$ as follows. The components of the input vector $x = (x_1, \ldots, x_n) \in [0,1]^n$ are assigned to the sources of C. Let $v_1, \ldots, v_k$ be the immediate predecessors of a vertex v. The input for v is then $s_v(x) = \sum_{i=1}^{k} a_i y_i - b_v$, where y_i is the value assigned to v_i and a and b are the weights of v. We assign the value $\phi_v(s_v(x))$ to v. Then $f_C = s_z$ is the function computed by C where z is the unique sink of C.

The architecture $\mathcal{A}$ of the Φ-net C is the structure of the underlying directed acyclic graph. Hence each architecture $\mathcal{A}$ defines a behavior function $\beta_{\mathcal{A}}$ that maps from the r real weights (corresponding to all the weights and thresholds of the underlying directed acyclic graph) and the input string into a binary output. We denote such a behavior as the function $\beta_{\mathcal{A}}(\mathbb{R}^r, [0,1]^n) \mapsto \{0,1\}$. The set of inputs which cause the output of the network to be 0 (resp. 1) are termed as the set of *negative* (resp. *positive*) examples. The *size* of the architecture $\mathcal{A}$ is the number of nodes and connections of $\mathcal{A}$ plus the maximum number of bits needed to represent any weight of $\mathcal{A}$.

The *loading problem* is defined as follows: Given an architecture $\mathcal{A}$ and a set of positive and negative examples $M = \{(x,y) \mid x \in [0,1]^n, y \in \{0,1\}\}$, so that $|M| = O(n)$; find weights $\vec{w}$ so that for all pairs $(x,y) \in M$:

$$\beta_{\mathcal{A}}(\vec{w}, x) = y \ .$$

The decision version of the loading problem is to decide (rather than to find the weights) whether such weights exist that load M onto $\mathcal{A}$.

Since the sink z of C is assumed to output only zero or one for the purpose of loading, we may henceforth assume that sink z is a threshold gate without any loss of generality.

For the purpose of this chapter we will be concerned with a very simple architecture as described in the next section.

2.2 The k Φ-node architecture

Here we focus on 1 hidden layer (1HL) architectures. The k Φ-node architecture is a 1HL architecture with k hidden ϕ-units (for some $\phi \in \Phi$), and an output node with the threshold activation $\mathcal{H}$. The 2 Φ-node architecture consists of two hidden nodes N_1 and N_2 that compute:

$$N_1(\vec{a}, \vec{x}) \;=\; \phi(\sum_{i=1}^{n} a_i x_i),$$

and

$$N_2(\vec{b}, \vec{x}) \;=\; \phi(\sum_{i=1}^{n} b_i x_i),$$

respectively.

The output node N_3 computes the threshold function of the inputs received from the two hidden nodes, namely a binary threshold function of the form

$$N_3(N_1, N_2, \alpha, \beta, \gamma) \;=\; \begin{cases} 1 & \text{if } \alpha N_1(\vec{a}, \vec{x}) + \beta N_2(\vec{b}, \vec{x}) > \gamma, \\ 0 & \text{if } \alpha N_1(\vec{a}, \vec{x}) + \beta N_2(\vec{b}, \vec{x}) \leq \gamma, \end{cases}$$

for some parameters α, β, and γ. Figure 1 illustrates a 2 Φ-node architecture.

The two activation-function classes Φ that we consider are the threshold functions $\mathcal{H}$, defined as

$$\mathcal{H}(x) = \begin{cases} 0 & \text{if } x \leq 0, \\ 1 & \text{if } x > 0, \end{cases}$$

and the piecewise linear or "saturating" activation functions π defined as

$$\pi(x) = \begin{cases} 0 & \text{if } x < 0, \\ x & \text{if } 0 \leq x \leq 1, \\ 1 & \text{if } x > 1\,. \end{cases} \tag{10.2}$$

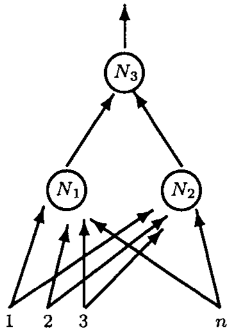

Figure 1 *A 2 Φ-node architecture*

Another model, called the 2-*cascade architecture* , was investigated by Lin and Vitter [17] (see fig. 2). A 2-cascade architecture consists of two processors N_1 and N_2 each of which computes a binary threshold function $\mathcal{H}$. The output of the node N_1 in the hidden layer is provided to the input of the output node N_2. Moreover, all the inputs are connected to both the nodes N_1 and N_2. Obviously, this is equivalent to the 2 Φ-node architecture where the hidden node N_1 and the output node N_3 computes the binary threshold function $\mathcal{H}$, and the remaining hidden node N_2 computes the identity function δ (i.e., $\delta(x) = x$ for all x).

The 2-cascade net is more economical in terms of the size of the architecture than the 2 $\mathcal{H}$-node architecture since they have lesser number of nodes and edges. Also, from the different classifications that can be produced by the 2-cascade network as mentioned in [17], it follows that they can realize all the classifications realizable by the 2 $\mathcal{H}$-node architecture and some additional classifications which cannot be realized by the 2 $\mathcal{H}$-node architecture. Hence, the 2-cascade network is more powerful than the 2 $\mathcal{H}$-node architecture in its classification capabilities as well.

2.3 Loading The k $\mathcal{H}$-Node Network: Previous Work

We summarize the results known for loading a 1-hidden layer threshold network (i.e. a k $\mathcal{H}$-node architecture for some integer $k > 1$). We consider two kinds

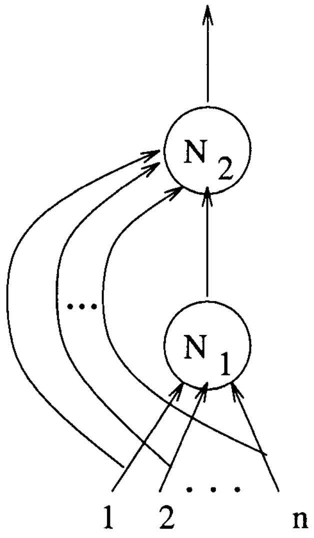

Figure 2 A 2-cascade network. Both the nodes N_1 and N_2 are threshold units.

of inputs: *analog* and *binary*. An analog input is in $[0, 1]^d$, where d is a fixed constant, also called the *input dimension*. In the binary case, the input is in $\{0, 1\}^n$ when n an input parameter.

Consider the geometrical view of the loading problem for such a network. Every threshold neuron defines a hyperplane, and we ask if there exists a set of hyperplanes that separate the points in the d-dimensional space which are labeled $'+'$ from the points there which are separated as $'-'$. The following definition is due to Megiddo [21]:

Definition 2.1 k-Polyhedral Separability: Given two sets of points A and B in $\mathbb{R}^d$, and an integer k, decide whether there exist k hyperplanes

$$H_j = \{p : (x^j)^T p = x_0^j\}, \quad (x^j \in \mathbb{R}^d, \; x_0^j \in \mathbb{R}, \; j = 1, \ldots, k)$$

that separate the sets through a Boolean formula. That is, associate a Boolean variable v_j with each hyperplane H_j. The variable v_j is true at a point p if $(x^j)^T p > x_0^j$, false if $(x^j)^T p < x_0^j$, and undefined at points lying on the hyperplane itself. A Boolean formula $\phi = \phi(v_1, \ldots, v_k)$ that separates the sets A and B is true for each point $a \in A$ and false for each $b \in B$.

One hidden layer net with k hidden units separates the space by k hyperplanes. However, not any Boolean formula of them is permitted, but only those which can be defined by a threshold neuron, i.e., the ones which are linearly separable in the quadrants.

When the input is analog (and the input dimension is hence constant), loading a 1-hidden layer network requires a polynomial time only in the size of the training set. This result is achieved by utilizing a result described by Megiddo [21]. Furthermore, such nets are also learnable as was proven by Maass in [19]. Megiddo [21] proved, in the same paper, that when the inputs are in Z^* (that is, integer values with unbounded dimensions) then problem turns to be NP-complete, even for the simple 2 $\mathcal{H}$-node architecture. Blum and Rivest [3] showed when the inputs are binary and the training set is sparse (i.e. if n is the length of the longest string in the training set M, then $|M|$ is polynomial in n) the loading problem is NP-Complete for the 2 $\mathcal{H}$-node architecture. In another related paper, Lin and Vitter [17] proved a slightly stronger result by showing that the loading problem of 2-cascade threshold net is NP-complete.

Next, we summarize some of the proof techniques of the above stated previous work.

2.3.1 Varying Input Dimensions (and Binary Inputs): Loading The 2 $\mathcal{H}$-node Network is NP-Complete: Blum and Rivest

Theorem 10.1 *[3] Loading the 2 $\mathcal{H}$-node network is NP-Complete for the case of varying input dimensions.*

The problem is in NP since the maximum number of bits required to represent each weight is $O(n \log n)$ (see, for example, [22]). To show that the problem is complete in NP, the following geometrical view of the problem is considered:

Let M be the training set, and n is the length of the longest string in M. Assume that $|M| = O(n)$.

1. A training example (i_j, o_j) can be thought of as a point in n+1-dimensional Boolean space $\{0,1\}^{n+1}$, labeled $+/-$.
2. The zeroes of the functions N_1, N_2 can be thought of as n-dimensional hyperplanes in this space.

3. The two hidden nodes define two hyperplanes which divide the n-dimensional space into four quadrant (may be degenerate) according to the $+/-$ sides of each of them.

4. The output node computes a threshold function on these quadrants.

Hence, the loading problem for the 2 $\mathcal{H}$-node architecture is equivalent to the following problem: Given a collection of labeled points in $\{0,1\}^n$, does there exist either

1. A single hyperplane separating positive and negative points, *or*

2. Two hyperplanes so that one quadrant consists of all positive points and no negative point, *or*

3. Two hyperplanes so that one quadrant consists of all negative points and no positive point.

An outline of their proof is as follows. First they proved that the second case, which they term as the *quadrant of positive Boolean example problem*, is NP-complete by giving a reduction from the set-splitting problem which is known to be NP-complete [9]. To complete the proof for the 2 $\mathcal{H}$-node network, they enlarged the dimension of the problem by a small constant and enlarged the given training set by a constant factor to disallow cases 1 and 3. For more details, see [3].

2.3.2 Fixed Input Dimension: Loading The k $\mathcal{H}$-Node Network is Polynomial Time

Theorem 10.2 *Let $k > 0$ be an integer constant. Then, it is possible to load any k $\mathcal{H}$-node architecture in polynomial time in the analog-input (fixed dimension) case.*

Before proving this result, we summarize the related result of Megiddo in [21] regarding polyhedral separability in fixed dimension and hyperplanes.

Lemma 2.1 *[21] Let d, k be constants, and Z represents the integers numbers. M is a set of points in Z^d which are labeled $+/-$. Then, there exists an algorithm to decide whether a set of classified points M can be separated by k hyperplanes which takes time polynomial in $|M|$.*

Proof. The following two propositions are proved there:

Proposition 2.2 [21, pp. 328] The hyperplanes $H_1, H_2, \ldots, H_k$ separate the sets A and B through a Boolean formula iff for every pair of points $a \in A$ and $b \in B$, there exists a hyperplane H_i such that a and b lie on different sides of it.

Proposition 2.3 [21, pp. 335] Suppose A and B are sets of points in $\mathbb{R}^d$ with integer coordinates, and suppose there exists a hyperplane $H = \{p \in R^d : y^T = y_0\}$ that separates A from B with $y^T a < y_0$ for $a \in A$. Then, there exists a hyperplane $H = \{p \in R^d : y^T p = y_0\}$, a positive rational number r, and integers j_A, j_B $(j_A, j_B \geq 1, j_A + j_B \leq d + 1)$ such that:

1. For every $a \in A$, $x^T a \leq x_0 - r$.
2. For every $b \in B$, $x^T b \geq x_0 + r$.
3. For at least j_A points $a \in A$, and j_B points $b \in B$, $x^T a = x_0 - r$ and $x^T b = x_0 + r$.

(The last proposition is proven to be equivalent to a linear programming problem of maximizing $r \geq 0$ over the bounded area: $x^T a \leq x_0 - r$, $x^T b \geq x_0 + r$, $-1 \leq x_j \leq 1$, which requires polynomial time only.)

Assume the sets $A, B \in Z^d$ are separable with k hyperplanes. Then, there exist k pairs of subsets A_i, B_i $(\bigcup A_i = A, \bigcup B_i = B)$ and k hyperplanes H_i $(i = 1, \ldots k)$ such that H_i separates A_i from B_i. By proposition 2.3, there exist such hyperplanes that satisfy also the equalities stated in the proposition. Furthermore, each candidate hyperplane is determined by some finite set of at most $d + 1$ points and at most d equalities $x_j = +/- 1$. We can enumerate in polynomial time of the sum $|A| + |B|$ all the relevant configurations of the k hyperplanes. By proposition 2.2, it takes polynomial time to check whether the hyperplanes separate A and B. Hence, the algorithm requires only polynomial time in $|M|$. □

Proof of Theorem 10.2. The computational view of the loading problem of analog input is very similar to the model of Lemma 2.1. However, in this case the points are in $[0, 1]^d$ rather than Z^d. The second discrepancy is that the output of the k $\mathcal{H}$-node architecture is a linear threshold function of the hyperplanes rather than an arbitrary Boolean function. The proof of Lemma 2.1 holds for the analog input as well. We add a polynomial algorithm to test each separating configuration of the hyperplanes to assure that the output of the network is indeed a linear threshold function of the hyperplanes. □

3 THE LOADING PROBLEM FOR THE 2 π-NODE ARCHITECTURE WITH VARYING INPUT DIMENSIONS.

One can generalize Theorem 10.2 and show that it is possible to load the 2 π-node architecture with analog inputs (with fixed input dimensions) in polynomial time. In this section we show that the loading problem for the 2 π-node architecture is NP-complete when (binary) inputs of arbitrary dimension are considered. The main theorem of this section is as follows.

Theorem 10.3 *The loading problem for the* 2 *π-node architecture ($L\pi AP$) with (binary) inputs of varying dimension is NP-complete.*

A corollary of the above theorem is as follows.

Corollary 10.1 *The class of Boolean functions computable by the* 2 *π-node architecture with (binary) inputs of varying dimension is not learnable, unless $RP = NP$.*

To prove theorem 10.3 we reduce a restricted version of the set splitting problem, which is known to be NP-complete [9], to this problem in polynomial time. However, due to the continuity of this activation function, many technical difficulties arise. The proof is organized as follows:

1. Providing a geometric view of the problem [subsection 3.1].
2. Introducing the (k, l)-set splitting problem and the symmetric 2-SAT problem [subsection 3.2].
3. Proving the existence of a polynomial algorithm that transforms a solution of the (3,3)-set splitting problem into a solution of its associated (2,3)-set splitting problem (using the symmetric 2-SAT problem) [subsection 3.3].

4. Defining the 3-hyperplane problem and proving it is NP-complete by reducing from the (2,3)-set splitting problem [subsection 3.4].
5. Proving that the LπAP is NP-complete. This is done using all the above items[subsection 3.5].

In subsection 3.6, we prove the corollary.

3.1 A Geometric View Of The Loading Problem

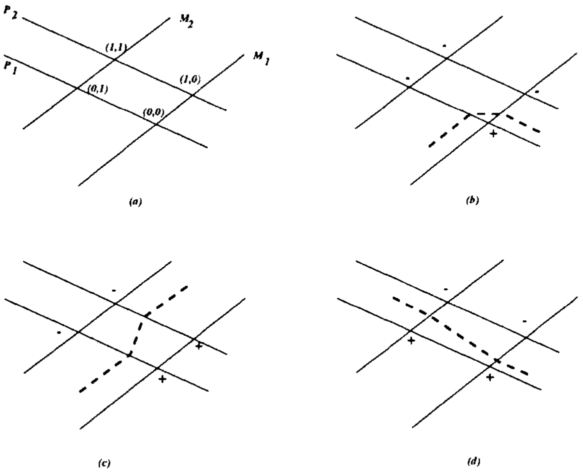

Figure 3 Different classifications produced by the 3-node network.

We start by categorizing the different types of classifications produced by the 2 π-node architecture. Without loss of generality we assume $\alpha, \beta \neq 0$ (if $\alpha = 0$ or $\beta = 0$ the network reduces to a simple perceptron which can be trained in polynomial time). Consider the 4 hyperplanes $M_1 : \sum_{i=1}^{n} a_i x_i = 0$, $M_2 : \sum_{i=1}^{n} a_i x_i = 1$, $P_1 : \sum_{i=1}^{n} b_i x_i = 0$, and $P_2 : \sum_{i=1}^{n} b_i x_i = 1$ (refer to fig. 3). Let (c_1, c_2) denote the set of points in the $(n-1)$-dimensional facet corresponding to $\sum_{i=1}^{n} a_i x_i = c_1$ and $\sum_{i=1}^{n} b_i x_i = c_2$. As all points belonging to one facet are labeled equally, we consider "labeling the facets" rather than the single points.

Type 1. All facets are labeled either $'+'$ or $'-'$. In that case, all the examples are labeled $'+'$ or $'-'$, respectively.

Type 2. Exactly one facet is labeled '+'. Assume that this facet is (0,0). Then, two different types of separations exist:

(a) There exist two halfspaces H_1 and H_2 such that all the '+' points belong to $H_1 \wedge H_2$ and all the '−' points belong to $\overline{H_1} \vee \overline{H_2}$ (H_1 and H_2 may be identical).

(b) There exist three hyperplanes of the following form (fig. 3(b)):

$$H_1 : \alpha(\sum_{i=1}^{n} a_i x_i) > \gamma,$$

$$H_2 : \beta(\sum_{i=1}^{n} b_i x_i) > \gamma,$$

and

$$H_3 : \sum_{i=1}^{n}(\alpha a_i + \beta b_i) x_i > \gamma,$$

where $\alpha, \beta \leq \gamma < 0$ (hence $\gamma > 2\gamma$), and all the '+' and '−' points belong to $H_1 \wedge H_2 \wedge H_3$ and $\overline{H_1} \vee \overline{H_2} \vee \overline{H_3}$, respectively (here, as well, H_1 and H_2 may be identical).

If any other facet is marked '+', a similar separation is produced.

Type 3. Two facets are marked '+' and the remaining two are labeled '−'. Because the labeling must be linearly separable, only the following types of classifications are possible:

(a) $(0,1)$ and $(0,0)$ are '+' (fig. 3(d)). Then, the input space is partitioned via the three halfspaces:

$$H_1 : \alpha(\sum_{i=1}^{n} a_i x_i) > \gamma - \beta,$$

$$H_2 : \alpha(\sum_{i=1}^{n} a_i x_i) > \gamma,$$

and

$$H_3 : \sum_{i=1}^{n}(\alpha a_i + \beta b_i) x_i > \gamma,$$

where $\beta > \gamma, \alpha \leq \gamma < 0, \alpha + \beta \leq \gamma$.

If $\beta < 0$ then all the '+' and '−' points lie in $H_1 \vee (H_2 \wedge H_3)$ and $\overline{H_2} \vee (\overline{H_1} \wedge \overline{H_3})$, respectively.

If $\beta > 0$ then all the $'+'$ and $'-'$ points lie in $H_2 \vee (H_1 \wedge H_3)$ and $\overline{H_1} \vee (\overline{H_2} \wedge \overline{H_3})$, respectively.

(b) $(0,0)$ and $(1,0)$ are $'+'$ (fig. 3(c)). Then, the input space is partitioned via the three halfspaces:

$$H_1 : \beta(\sum_{i=1}^{n} b_i x_i) > \gamma - \alpha,$$

$$H_2 : \beta(\sum_{i=1}^{n} b_i x_i) > \gamma,$$

and

$$H_3 : \sum_{i=1}^{n}(\alpha a_i + \beta b_i)x_i > \gamma,$$

where $\alpha > \gamma, \beta \leq \gamma < 0, \alpha + \beta \leq \gamma$.

If $\alpha < 0$ then all the $'+'$ and $'-'$ points lie in $H_1 \vee (H_2 \wedge H_3)$ and $\overline{H_2} \vee (\overline{H_1} \wedge \overline{H_3})$, respectively.

If $\alpha > 0$ then all the $'+'$ and $'-'$ points lie in $H_2 \vee (H_1 \wedge H_3)$ and $\overline{H_1} \vee (\overline{H_2} \wedge \overline{H_3})$, respectively.

(c) $(1,0)$ and $(1,1)$ are $'+'$ (similar to fig. 3(d) with the labeling of $'+'$ and $'-'$ points interchanged). This is the symmetrically opposite case of type 3(a).

(d) $(0,1)$ and $(1,1)$ are $'+'$ (similar to fig. 3(c) with the labeling of $'+'$ and $'-'$ points interchanged). This is the symmetrically opposite case of type 3(b).

Type 4. Three facets are labeled $'+'$. This case is symmetrically opposite to type 2, and thus details are precluded. Note that two types are possible in type 4, namely type 4(a) and type 4(b), depending upon whether two or three halfspaces are involved, respectively (similar to type 2).

3.2 The Set Splitting and Symmetric 2-SAT Problems

The following problem is referred to as the (k,l)-set splitting problem (SSP) for $k \geq 2$.

INSTANCE: A set $S = \{s_i \mid 1 \leq i \leq n\}$, and a collection $C = \{c_j \mid 1 \leq j \leq m\}$ of subsets of S, all of exactly size l.

QUESTION: Are there k sets $S_1, \ldots, S_k$, such that $S_i \cap S_j = \phi$ for $i \neq j$, $\cup_{i=1}^k S_i = S$, and $c_j \not\subseteq S_i$ for $1 \leq i \leq k$ and $1 \leq j \leq m$?

Note that the (k,l)-SSP is solvable in polynomial time if both $k \leq 2$ and $l \leq 2$, but remains NP-complete if $k \geq 2$ and $l = 3$ (see [9]).

For later purposes we consider the *symmetric* 2-SAT problem:

INSTANCE: Variables $v_1, v_2, \cdots, v_n$ and a collection D of one or two literal disjunctive clauses satisfying the condition:

$$\forall i,j \quad [(v_i \vee (\neg v_j)) \notin D] \ \&\ [((\neg v_i) \vee v_j) \notin D].$$

QUESTION: Decide whether there exists a satisfying assignment, and find one if exists.

Note that the clause $(x_i \vee x_j)$ (resp. $((\neg x_i) \vee (\neg x_j))$) is equivalent to both the implications $(\neg x_i \rightarrow x_j)$ and $(\neg x_j \rightarrow x_i)$ (resp. $(x_i \rightarrow \neg x_j)$ and $(x_j \rightarrow \neg x_i)$, while the clause x_i (resp. $\neg x_i$) is equivalent to the implication $(\neg x_i \rightarrow x_i)$ (resp. $(x_i \rightarrow \neg x_i)$) only. These two forms of disjunction and implication are used interchangeably. In a manner similar to [24], we create a directed graph $G = (V, E)$, where where $V = \{d_i, \overline{d_i} \mid v_i \text{ is a variable}\}$, and $E = \{(l_i, l_j) \mid (i, j \in \{1, \ldots, n\}), (l_i \in \{d_i, \neg d_i\}), (l_j \in \{d_j, \neg d_j\}), (l_i \rightarrow l_j) \in D\}$. Note that an edge (x, y) in E is directed from x to y. In the symmetric 2-SAT problem, the graph G has the following crucial property:

(♣) Complemented and uncomplemented vertices alternate in any path. This is because the edges in G are only of the form $(d_i, \overline{d_j})$ or $(\overline{d_i}, d_j)$ for some two indices i and j ($i = j$ is possible).

The following algorithm finds a satisfiable assignment if exists or, stops if there is no one:

1. Denote by $\Rightarrow$ the transitive closure of $\rightarrow$. For any variable v_i such that $v_i \Rightarrow \neg v_i$ (resp $\neg v_i \Rightarrow v_i$) set v_i to false (resp. true).

2. Repeat until there is no edge directed into a false literal or from a true literal.

   - Pick an edge directed into a false literal, i.e. of the type $d_r \rightarrow \neg d_s$ (resp. $\neg d_r \rightarrow d_s$) so that the variable v_s is set to true (resp. false) and set v_r to false (resp. true).
   - Pick an edge directed from a true literal, i.e. of the type $d_r \rightarrow \neg d_s$ (resp. $\neg d_r \rightarrow d_s$) so that the variable v_r is set to true (resp. false) and set v_s to false (resp. true).

3. If there is still an unassigned variable, set it arbitrarily and return to step 2. Otherwise, halt.

The above algorithm produces a satisfying assignment provided the following condition holds (see, for example, [24, pp. 377-378]):

> *The instance of the 2-SAT problem has a solution if and only if there is no directed cycle in G which contains both the vertices d_i and $\overline{d_i}$ for some i.*

It is easy to check the above condition in $O(| V |) = O(n)$ time by finding the strongly connected components of G. Hence, computing a satisfying assignment (or, reporting that no such assignment exists) can be done in time polynomial in the input size.

3.3 The (k,l)-Reduction Problem

We prove that under certain conditions, a solution of the (k,l)-set splitting instance (S,C) can be transformed into a solution of the associated $(k-1,l)$-set splitting problem. More formally, we define the (k,l)-reduction problem ((k,l)-RP) as follows:

INSTANCE: An instance (S,C) of the (k,l)-SSP, and a solution $(S_1, S_2, \ldots, S_k)$.

QUESTION: Decide whether there exists a solution $(S'_1, S'_2, \ldots, S'_{k-1})$ to the associated $(k-1,l)$-SSP and construct one (if exists), where, for all

$i, j \in \{1, 2, \ldots, k-1\}$ $i \neq j$:

$$\begin{aligned} S'_i &= S_i \cup T_i, \\ T_i &\subseteq S_k, \\ (T_i \cap T_j) &= \phi \quad i \neq j \ , \\ \cup_{p=1}^{k-1} T_p &= S_k \ . \end{aligned}$$

We next state the existence of a polynomial algorithm for the $(3,3)$-reduction problem. Since we are interested in placing elements of S_3 in S_1 or S_2, we focus on sets having at least one element of S_3. Since (S_1, S_2, S_3) is a solution of the $(3,3)$-SSP, no set contains 3 elements of S_3. Let $C' = \{c_j \mid 1 \leq i \leq m\} \subseteq C$ be the collection of sets which contain at least one element of S_3. Obviously, $\forall j (c_j \not\subseteq S_1) \wedge (c_j \not\subseteq S_2) \wedge (c_j \not\subseteq S_3)$.

Let $A = \{a_i \mid 1 \leq i \leq |S|\}$ and $B = \{b_i \mid 1 \leq i \leq |S|\}$ be two disjoint sets. Each element of $A \cup B$ is to be colored *'red'* or *'blue'* so that the overall coloring satisfies the *valid coloring conditions*:

(a) For each set $\{x_i, x_j, x_p\} \in C'$, where $x_i, x_j \in S_3$, at least one of a_i or a_j should be colored red if $x_p \in S_1$ and at least one of b_i or b_j has to be colored red if $x_p \in S_2$.

(b) For each i, $1 \leq i \leq |S|$, at least one of a_i or b_i has to be colored blue.

(c) For each set $\{x_i, x_j, x_p\}$ such that $x_p \in S_3$ and $x_i, x_j \in S_1$ (resp. $x_i, x_j \in S_2$), a_p (resp. b_p) must be colored red.

Theorem 10.4 *The following two statements are true:*

(a) *The* $(3,3)$*-reduction problem is polynomially solvable.*

(b) *If the* $(3,3)$*-RP has no solution, no valid coloring of* $A \bigcup B$ *exists.*

Proof.
(a) We show how to reduce the $(3,3)$-reduction problem in polynomial time to the symmetric 2-SAT. As the later is polynomially solvable, part (a) will be proven. Assume an instance (S, C, S_1, S_2, S_3) is given and (S'_1, S'_2) is to be found. For each element $x_i \in S_3$ assign a variable v_i; $v_i = TRUE$ (resp. $v_i = FALSE$) indicates that the element x_i is placed in S_1 (resp. S_2). For each

set $c_k = \{x_i, x_j, x_p\}$, where $x_i, x_j \in S_3$, if x_p is in S_1, create the clause $\neg v_i \vee \neg v_j$ (indicating both v_i and v_j should not be true, since otherwise $c_k \subseteq S_1'$); if x_p is in S_2 create the clause $v_i \vee v_j$; for each set $c_k = \{x_i, x_j, x_p\}$, where $x_i, x_j \in S_1$ (resp, $\in S_2$), create the clause $\neg v_p$ (resp. v_p). Let D be the collection of all such clauses. This instance of the symmetric 2-SAT problem has a satisfying assignment if and only if the (3, 3)-RP has a solution: for each variable v_j; v_j is true (resp. false) in the satisfying assignment if and only if x_j is assigned into S_1 (resp. S_2).

(b) Construct the graph G from the collection of clauses D as described in section 3.2. If no satisfying assignment exists, the graph G has a directed cycle containing both d_i and $\overline{d_i}$ for some i. We show that in that case no valid coloring of all the elements of $A \cup B$ is possible: rearrange the indices and names of the variable, if necessary, so that the cycle contains d_1 and $\overline{d_1}$, and (due to property ($\clubsuit$) of G of section 3.2) is of the form $d_1 \to \overline{d_2} \to d_3 \to \cdots \to d_r \to \overline{d_1} \to d_{1'} \to \overline{d_{2'}} \to d_{3'} \to \cdots \to \overline{d_{s'}} \to d_1$, where r and s' are two positive integers and $x \to y$ denotes an edge directed from vertex x to vertex y in G (not all of the indices $1, 2, \ldots, r, 1', 2', \ldots, s'$ need to be distinct). Next, we consider the following 2 cases.

Case 1. Assume a_1 is colored red. Hence, b_1 must be colored blue due to coloring condition (b).

Consider the path from P from $\overline{d_1}$ to d_1 (i.e., the path $\overline{d_1} \rightsquigarrow d_1$, where $\rightsquigarrow$ denotes the sequence of one or more edges in G). The following subcases are possible:

Case 1.1. P contains at least one edge of the form $d_{t'} \to \overline{d_{t'}}$ or $\overline{d_{t'}} \to d_{t'}$ for some index t'. Consider the *first* such edge along P as we traverse from $\overline{d_1}$ to d_1.

Case 1.1.1. The edge is of the form $d_{t'} \to \overline{d_{t'}}$, (that is, the associated clause is $\neg x_{t'}$). Consider the path $P' : \overline{d_1} \rightsquigarrow d_{t'}$. P' is of the form $\overline{d_1} \to d_{1'} \to \overline{d_{2'}} \to \cdots \to \overline{d_{t'-1}} \to d_{t'}$ and t' is odd ($t' = 1$ is possible). Now, due to coloring condition (a) and (b), $b_{t'}$ is colored red (see below).

	$i = 1$	$i = 1'$	$i = 2'$	...	$i = t' - 1$	$i = t'$
a_i :		blue	red	$\cdots$	red	
b_i :	blue	red	blue	$\cdots$	blue	red

On the other hand, $a_{t'}$ is colored red due to coloring condition (c) and the edge $d_{t'} \to \overline{d_{t'}}$. But, coloring condition (b) prevents both $a_{t'}$ and $b_{t'}$ to be colored red.

Case 1.1.2. The edge is of the form $\overline{d_{t'}} \rightarrow d_{t'}$ (that is, the associated clause is $x_{t'}$). Consider the path $P' : \overline{d_1} \rightsquigarrow \overline{d_{t'}}$. P' is of the form $\overline{d_1} \rightarrow d_{1'} \rightarrow \overline{d_{2'}} \rightarrow \cdots \rightarrow d_{t'-1} \rightarrow \overline{d_{t'}}$ and t' is even. Now, due to coloring condition (a) and (b), $a_{t'}$ is colored red (see below).

	$i = 1$	$i = 1'$	$i = 2'$	$\ldots$	$i = t' - 1$	$i = t'$
a_i :		blue	red	$\cdots$	blue	red
b_i :	blue	red	blue	$\cdots$	red	

On the other hand, $b_{t'}$ is colored red due to coloring condition (c) and the edge $\overline{d_{t'}} \rightarrow d_{t'}$. But, coloring condition (b) prevents both $a_{t'}$ and $b_{t'}$ to be colored red.

Case 1.2. P contains no edge of the form $d_{t'} \rightarrow \overline{d_{t'}}$ or $\overline{d_{t'}} \rightarrow d_{t'}$ for any index t'.

Then, s' is even, and because of the coloring conditions (a) and (b) we must have $b_{s'}$ colored blue (see below).

	$i = 1$	$i = 1'$	$i = 2'$	$\ldots$	$i = s' - 1$	$i = s'$
a_i :		blue	red	$\cdots$	blue	
b_i :	blue	red	blue	$\cdots$	red	blue

Now, b_1 must be colored red because of the edge $\overline{d_{s'}} \rightarrow d_1$, a contradiction.

Case 2. Assume a_1 is colored blue.

This case is symmetric to Case 1 if we consider the path $d_1 \rightsquigarrow \overline{d_1}$ instead of the path $\overline{d_1} \rightsquigarrow d_1$.

Hence, part (b) is proved. □

3.4 The 3-hyperplane Problem

We prove the following problem, which we term as the 3-hyperplane problem (3HP), to be NP-complete.

INSTANCE: A set of points in an n-dimensional hypercube labeled $'+'$ and $'-'$.

QUESTION: Does there exist a separation of one or more of the following forms:

(a) A set of two halfspaces $\vec{a}\vec{x} > a_0$ and $H_2 : \vec{b}\vec{x} > b_0$ such that all the '+' points are in $H_1 \wedge H_2$, and all the '−' points belong to $\overline{H_1} \vee \overline{H_2}$?

(b) A set of 3 halfspaces $H_1 : \vec{a}\vec{x} > a_0$, $H_2 : \vec{b}\vec{x} > b_0$ and $H_3 : (\vec{a+b})\vec{x} > c_0$ such that all the '+' points belong to $H_1 \wedge H_2 \wedge H_3$ and all the '−' points belong to $\overline{H_1} \vee \overline{H_2} \vee \overline{H_3}$?

Theorem 10.5 *The 3-hyperplane problem is NP-complete.*

Proof. We first notice that this problem is in NP as an affirmative solution can be verified in polynomial time. To prove NP-completeness of the 3HL, we reduce the (2,3)-set splitting problem to it:

Given an instance I of the (2,3)-SSP:

$$I: \quad S = \{s_i\},\ C = \{c_j\},\ c_j \subseteq S,\ |\, S \,| = n,\ |\, c_j \,| = 3 \text{ for all } j$$

we create the instance I' of the 3-hyperplane problem (like in [3]):

> ⋆ The origin (0^n) is labeled '+'; for each element s_j, the point p_j having 1 in the j^{th} coordinate only is labeled '−'; and for each clause $c_l = \{s_i, s_j, s_k\}$, we label with '+' the point p_{ijk} which has 1 in its i^{th}, j^{th}, and k^{th} coordinates.

We next prove that

An instance I' of the 3-hyperplane problem has a solution if and only if instance I of the (2,3)-SSP has a solution.

⇒

Given a solution (S_1, S_2) of the (2,3)-SSP, we create the following two halfspaces: $H_1 : \sum_{i=1}^{n} a_i x_i > -\frac{1}{2}$, where $a_i = -1$ if $s_i \in S_1$ and $a_i = 2$ otherwise, $H_2 : \sum_{i=1}^{n} b_i x_i > -\frac{1}{2}$, where $b_i = -1$ if $s_i \in S_2$ and $b_i = 2$ otherwise. This is a solution type (a) of the 3-hyperplane problem.

⇐

(A) If there is a separation of type (a), the solution of the set-splitting is analogous to [3]: Let S_1 and S_2 be the set of $'-'$ points p_j separated from the origin by H_1 and H_2, respectively (any point separated by both is placed arbitrarily in one of them). To show that this separation is indeed a valid solution, assume a subset $c_d = \{x_i, x_j, x_k\}$ so that p_i, p_j, p_k are separated from the origin by H_1. Then, also c_d is separated from the origin by the same hyperplane, contradicting its positive labeling.

(B) Otherwise, let $H_1 : \sum_{i=1}^{n} a_i x_i > -\frac{1}{2}$, $H_2 : \sum_{i=1}^{n} b_i x_i > -\frac{1}{2}$ and $H_3 : \sum_{i=1}^{n}(a_i + b_i)x_i > c$ be the three solution halfspaces of type (b), where $0 > c$ (since the origin is labeled $'+'$). We show how to construct a solution of the set splitting problem.

Let S_1 and S_2 be the set of $'-'$ points p_j separated from the origin by H_1 and H_2, respectively (any point separated by both is placed arbitrarily in one of the sets), and let S_3 be the set of points p_j separated from the origin by H_3 but by *neither* H_1 *nor* H_2. If $S_3 = \phi$ then S_1 and S_2 imply a solution as in (A) above. Otherwise, the following properties hold:

(I) There cannot be a set $c_j = \{s_x, s_y, s_z\}$ where p_x, p_y and p_z all belong to S_3. Otherwise, $a_x, a_y, a_z < c < 0$, and the $'+'$ point corresponding to c_j is classified $'-'$ by H_3. Similarly, no set c_j exists that is included in either S_1 or S_2.

(II) Consider a set $\{s_x, s_y, s_z\}$, where $p_x, p_y \in S_3, p_z \in S_1$. Since $a_z \leq -\frac{1}{2}$ and $a_z + a_x + a_y > -\frac{1}{2}$, we conclude $a_x + a_y > 0$. Hence, at least one of a_x or a_y must be strictly positive. Similarly, if $p_z \in S_2$, at least one of b_x, b_y is strictly positive.

(III) Consider any element s_x of S_3. Since the associated point p_x is classified as $'-'$ by H_3, $a_x + b_x < c < 0$. Hence, at least one of a_x and b_x is negative for each p_x.

(IV) If there is a set $\{s_x, s_y, s_z\}$ where $s_x \in S_3$, and $s_y, s_z \in S_1$ (resp. $s_y, s_z \in S_2$) then a_x (resp. b_x) is positive. This is because since $s_y, s_z \in S_1$ (resp. $s_y, s_z \in S_2$), $a_y, a_z \leq -\frac{1}{2}$ (resp. $b_y, b_z \leq -\frac{1}{2}$), but $a_x + a_y + a_z > -\frac{1}{2}$ (resp. $b_x + b_y + b_z > -\frac{1}{2}$), and hence $a_x > \frac{1}{2}$ (resp. $b_x > \frac{1}{2}$).

As for condition (I), (S_1, S_2, S_3) can be viewed as a solution of the (3,3)-SSP. We show that this solution can be transformed into a solution of the required (2,3)-SSP.

Let $A = \{a_i \mid 1 \leq i \leq t\}$, $B = \{b_i \mid 1 \leq i \leq t\}$, S_1, S_2 and S_3 be as in theorem 10.4. Each element x of $A \cup B$ is colored *red* (resp. *blue*) if $x > 0$ (resp. $x \leq 0$). Conditions (a), (b) and (c) of valid coloring of $A \cup B$

hold because of conditions (II), (III) and (IV) above. Thus, (S_1, S_2, S_3) is transformed into (S'_1, S'_2)—a solution of the (2,3)-SSP. □

3.5 Loading The 2 π-node Architecture is NP-complete

Next, we prove that loading the 2 π-node architecture is NP-complete. We do so by comparing it to the 3-hyperplane problem. To this end, we construct a gadget that will allow the architecture to produce only separations of type 2 (section 3.1), which are similar to those of the 3HP.

We construct such a gadget with two steps: first, in Lemma 3.1, we exclude separation of type 3, and then in Lemma 3.2 we exclude separations of type 4.

Lemma 3.1 *Consider the* 2*-dimensional hypercube in which (0,0), (1,1) are labeled* $'+'$, *and (1,0), (0,1) are labeled* $'-'$. *Then the following statements are true:*

(a) *There do not exist three halfspaces* H_1, H_2, H_3 *as described in type* 3*(a)-(d) in section 3.1 which correctly classify this set of points.*

(b) *There exist two halfspaces of the form* $H_1 : \vec{a}\vec{x} > a_0$ *and* $H_2 : \vec{b}\vec{x} > b_0$, *where* $a_0, b_0 < 0$, *such that all the* $'+'$ *and* $'-'$ *points belong to* $H_1 \wedge H_2$ *and* $\overline{H_1} \vee \overline{H_2}$, *respectively.*

Lemma 3.2 *Consider the labeled set A: (0,0,0), (1,0,1), (0,1,1) are labeled* $'+'$, *and (0,0,1), (0,1,0), (1,0,0), (1,1,1) are labeled* $'-'$. *Then, there does not exist a separation of these points by type* 4 *halfspaces as described in section 3.1.*

The proofs of Lemmas 3.1 and 3.2 involve a detailed case analysis and hence omitted; they are available in [7].

Consider the same classification again on a 3-dimensional hypercube: (0,0,0), (1,0,1), and (0,1,1) are labeled $'+'$, and (0,0,1), (0,1,0), (1,0,0), and (1,1,1) are labeled $'-'$. Then, the following statements are true due to the result in [3]:

(a) No single hyperplane can correctly classify the $'+'$ and $'-'$ points.

(b) No two halfspaces H_1 and H_2 exist such that all the $'+'$ points belong to $H_1 \vee H_2$ and all the $'-'$ points belong to $\overline{H_1} \wedge \overline{H_2}$.

(c) There exist two halfspaces $H_1 : \sum_{i=1}^{3} \alpha_i x_i > \alpha_0$ and $H_2 : \sum_{i=1}^{3} \beta_i x_i > \beta_0$ such that all the $'+'$ points lie in $H_1 \wedge H_2$, and all the $'-'$ points lie in $\overline{H_1} \vee \overline{H_2}$ (where $X = (x_1, x_2, x_3)$ is the input).

Now, we can show that the loading problem for the 2 π-node architecture is NP-complete.

Proof of theorem 10.3. First we observe that the problem is in NP as follows. The classifications of the labeled points produced by the 2 π-node architecture (as discussed in section 3.1) are 3-polyhedrally separable. Hence, from the result of [22] we can restrict all the weights to have at most $O(n \log n)$ bits. Hence, a "guessed" solution can be verified in polynomial time.

Next, we show that the problem is NP-complete. Consider an instance $I = (S, C)$ of the (2,3)-SSP. We transform it into an instance I' of the problem of loading the 2 π-node architecture as follows: we label points on the $(|S| + 5)$ hypercube similar to as is $\star$ (section 3.4).

> The origin $(0^{|S|+5})$ is labeled $'+'$; for each element s_j, the point p_j having 1 in the j^{th} coordinate only is labeled $'-'$; and for each clause $c_l = \{s_i, s_j, s_k\}$, we label with $'+'$ the point p_{ijk} which has 1 in its i^{th}, j^{th}, and k^{th} coordinates. The points $(0^n,0,0,0,0,0)$, $(0^n,0,0,0,1,1)$, $(0^n,1,0,1,0,0)$ and $(0^n,0,1,1,0,0)$ are marked $'+'$, and the points $(0^n,0,0,0,1,0)$, $(0^n,0,0,0,0,1)$, $(0^n,0,0,1,0,0)$, $(0^n,0,1,0,0,0)$, $(0^n,1,0,0,0,0)$ and $(0^n,1,1,1,0,0)$ are labeled $'-'$.

Next, we show that a solution for I exists iff there exists a solution to I'. Given a solution to the (2,3)-SSP, by lemma 3.1(part(b)) and the result in [3] the two solution halfspaces to I' are as follows (assume the last 5 dimensions are x_{n+1} to x_{n+5}):

$$H_1 : (\sum_{i=1}^{n} a_i x_i) - x_{n+1} - x_{n+2} + x_{n+3} - x_{n+4} + x_{n+5} > -\frac{1}{2},$$

and

$$H_2 : (\sum_{i=1}^{n} b_i x_i) + x_{n+1} + x_{n+2} - x_{n+3} + x_{n+4} - x_{n+5} > -\frac{1}{2},$$

where

$$a_i = \begin{cases} -1 & \text{if } s_i \in S_1, \\ 2 & \text{otherwise,} \end{cases}$$

and

$$b_i = \begin{cases} -1 & \text{if } s_i \in S_2, \\ 2 & \text{otherwise.} \end{cases}$$

We map the two solution halfspaces into the 2 π-node architecture as follows:.

$$\begin{aligned} N_1 &= \pi[-((\sum_{i=1}^{n} a_i x_i) - x_{n+1} - x_{n+2} + x_{n+3} - x_{n+4} + x_{n+5})] , \\ N_2 &= \pi[-((\sum_{i=1}^{n} b_i x_i) + x_{n+1} + x_{n+2} - x_{n+3} + x_{n+4} - x_{n+5})] , \\ N_3 &= \begin{cases} 1 & \text{if } \ -N_1 - N_2 > -1, \\ 0 & \text{if } \ -N_1 - N_2 \leq -1 . \end{cases} \end{aligned}$$

Conversely, given a solution to I', by Lemma 3.1(part (a)), Lemma 3.2 and the result in [3] (as discussed above) the only type of classification produced by the 2 π-node architecture consistent with the classifications on the lower 5 dimensions is of type 2(a) (with $H_1 \neq H_2$) or 2(b) only, which was shown to be NP-complete in theorem 10.5. □

Remark 10.2 *From the above proof of theorem 10.3 it is clear that the NP-completeness result holds even if all the weights are constrained to lie in the set* $\{-2, -1, 1\}$. *Thus the hardness of the loading problem holds even if all the weights are "small" constants.*

3.6 Learning the 2 π-node Architecture

Here, we prove corollary 10.1 which states that the functions computable by the 2 π-node architecture is not learnable unless $RP = NP$. As it is not believed that NP and RP are equal, the corollary implies that most likely the 2 π-node architecture is not learnable (i.e. there are particular values of ϵ and δ it is not (ϵ, δ)-learnable).

Proof of Corollary 10.1. The proof uses a similar technique to the one applied in the proof of theorem 9 of [15]. We assume that the functions computed by the 2 π-node architecture are learnable and show that it implies an RP algorithm for solving a known NP-complete problem, that is, NP=RP.

Given a instance $I = (S, C)$ of the $(2,3)$-SSP, we create an instance I' of the 2 π-node architecture and a set of labeled points M (this was used in the proof of theorem 10.3):

> The origin $(0^{|S|+5})$ is labeled $'+'$; for each element s_j, the point p_j having 1 in the j^{th} coordinate only is labeled $'-'$; and for each clause $c_l = \{s_i, s_j, s_k\}$, we label with $'+'$ the point p_{ijk} which has 1 in its i^{th}, j^{th}, and k^{th} coordinates. The points $(0^n, 0, 0, 0, 0, 0)$, $(0^n, 0, 0, 0, 1, 1)$, $(0^n, 1, 0, 1, 0, 0)$ and $(0^n, 0, 1, 1, 0, 0)$ are marked $'+'$, and the points $(0^n, 0, 0, 0, 1, 0)$, $(0^n, 0, 0, 0, 0, 1)$, $(0^n, 0, 0, 1, 0, 0)$, $(0^n, 0, 1, 0, 0, 0)$, $(0^n, 1, 0, 0, 0, 0)$ and $(0^n, 1, 1, 1, 0, 0)$ are labeled $'-'$.

Let D^+ (resp. D^-) be the uniform distribution over these $'+'$ (resp. $'-'$) points. Choose $\epsilon < \min\{\frac{1}{|S|+5}, \frac{1}{|C|+4}\}$, and $\delta = 1 - \epsilon$. To prove the corollary it is sufficient to show that for the above choice of ϵ, δ, D^+ and D^-, (ϵ, δ)-learnability of the 2 π-node architecture can be used to decide the outcome of the $(2,3)$-SSP in random polynomial time:

- Suppose I is an instance of the $(2,3)$-SSP and let (S_1, S_2) be its solution. Then, from the proof of the "only if" part of Theorem 10.3 (see previous subsection), there exists a solution to I' which is consistent with the labeled points of M. So, if the 2 π-node architecture is (ϵ, δ)-learnable, then due to choice of ϵ and δ (and, by Theorem 10.3), the probabilistic learning algorithm *must* produce a solution which is consistent with M with probability at least $1 - \epsilon$, thereby providing a probabilistic solution of the $(2,3)$-SSP. That is, if the answer to the $(2,3)$-SSP question is "YES", then we answer "YES" with probability at least $1 - \epsilon$.

- Now, suppose that there is no solution possible for the given instance of the $(2,3)$-SSP. Then, by Theorem 10.3, there is no solution of the 2 π-node architecture which is consistent with M. Hence, the learning algorithm must *always* either produce a solution which is *not* consistent with M, or *fail* to halt in time polynomial in n, $\frac{1}{\epsilon}$, and $\frac{1}{\delta}$. In either case we can detect that the learning algorithm was inconsistent with labeled points or did not halt in stipulated time, and answer "NO". In other words, if the answer to the (2, 3)-SSP is "NO", we always answer "NO".

Since the (2, 3)-SSP is NP-complete (i.e., any problem in NP has a polynomial time transformation to (2, 3)-SSP), it follows that any problem in NP has a random polynomial time solution, i.e., $NP \subseteq RP$. But it is well-known that $RP \subseteq NP$, hence we have $RP = NP$. □

4 CONCLUDING REMARKS

We have shown that the loading problem is NP-complete even for a simple feedforward network with a specific "saturated linear " (analog type) activation functions. This adds to the previously known results stating that the loading of a simple net with discrete activations is NP-complete ([3]) and a net with a specific (somehow artificial) analog activation function has a fast loading ([28]). It is possible to extend the NP-completeness result when a fixed polynomial number of threshold units are added in the hidden layer, provided the function computed by the output node is restricted; the reader is referred to [7] for details. Unfortunately, our proof does not seem to generalize for standard sigmoid or other similar activation functions. The following open problems may be worth investigating further:

- Does the NP-completeness result hold for the 2 σ-node architecture, where $\sigma(x) = \frac{1}{1+e^{-x}}$ is the standard sigmoid function?
- What is the complexity of the loading problem for networks with more layers? Note that hardness of the loading problem for networks with one hidden layers does not necessarily imply the same for networks with more hidden layers. In fact, it is already known that there are functions which cannot be computed by threshold networks with one hidden layer and a constant number of nodes, but can be computed by threshold networks with two hidden layers and a constant number of nodes [20].
- Is there a characterization of the activation functions for which the loading problem is intractable?

REFERENCES

[1] Barron, A.R., "Approximation and estimation bounds for artificial neural networks", *Proc. 4th Annual Workshop on Computational Learning Theory*, Morgan Kaufmann, 1991, pp. 243-249.

[2] Baum, E.B., and Haussler, D., "What size net gives valid generalization?," *Neural Computation,* **1**(1989): 151-160

[3] Blum, A., and Rivest, R. L., "Training a 3-node neural network is NP-complete," in *Advances in Neural Information Processing Systems 2* (D.S. Touretzky, ed), Morgan Kaufmann, San Mateo, CA, 1990, pp. 9-18; also as "Training a 3-Node Neural Network is NP-Complete," *Neural Networks,* **5**(1992): 117-127.

[4] Bruck, J., and Goodman, J. W., "On the power of neural networks for solving hard problems", *Journal of Complexity,* **6**(1990): 129-135.

[5] Darken, C., Donahue, M., Gurvits, L., and Sontag, E., "Rate of approximation results motivated by robust neural network learning," *Proc. 6th ACM Workshop on Computational Learning Theory*, Santa Cruz, July 1993, pp. 303-309.

[6] DasGupta, B., and Schnitger, G., "The power of approximating: a comparison of activation functions," in *Advances in Neural Information Processing Systems 5* (Giles, C.L., Hanson, S.J., and Cowan, J.D., eds), Morgan Kaufmann, San Mateo, CA, 1993, pp. 615-622.

[7] DasGupta, B., Siegelmann, H. T., and Sontag, E., "On the Complexity of Training Neural Networks with Continuous Activation Functions", Tech Report # 93-61, Department of Computer Science, University of Minnesota, September, 1993.

[8] Fischer, P. and Simon, H. U., "On Learning Ring-Sum Expansions", *SIAM J. Computing,* **21,** **1**(1992): 181-192.

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

[10] Gill, J., "Computational Complexity of Probabilistic Turing Machines", *SIAM J. Computing,* **7,** **4**(1977): 675-695.

[11] Goldberg, P., and Jerrum, M., "Bounding the Vapnik-Chervonenkis dimension of concept classes parametrized by real numbers," *Proc. 6th ACM*

Workshop on Computational Learning Theory, Santa Cruz, July 1993, pp. 361-369.

[12] Jones, K.L., "A simple lemma on greedy approximation in Hilbert space and convergence rates for projection pursuit regression and neural network training," *Annals of Statistics*, to appear.

[13] Judd, J.S., "On the complexity of learning shallow neural networks," *J. of Complexity,* **4**(1988): 177-192.

[14] Judd, J.S., *Neural Network Design and the Complexity of Learning*, MIT Press, Cambridge, MA, 1990.

[15] Kearns, M., Li, M., Pitt, L., and Valiant, L., "On the learnability of Boolean formulae," *Proc. of the 19th ACM Symp. Theory of Computing*, 1987, pp. 285-295.

[16] Kilian, J. and Siegelmann, H. T., "Computability With The Classical Sigmoid," *Proc. of the 5th ACM Workshop on Computational Learning Theory,* Santa Cruz, July 1993, pp. 137-143.

[17] Lin, J-H., and Vitter, J. S., "Complexity results on learning by neural networks," *Machine Learning,* **6**(1991): 211-230.

[18] Macintyre, A., and Sontag, E. D., "Finiteness results for sigmoidal 'neural' networks," *Proc. 25th Annual Symp. Theory Computing*, San Diego, May 1993, pp. 325-334.

[19] Maass, W., "Bounds for the computational power and learning complexity of analog neural nets," *Proc. of the 25th ACM Symp. Theory of Computing,* May 1993, pp. 335-344 .

[20] Maass, W., Schnitger, G., and Sontag, E. D., "On the computational power of sigmoid versus boolean threshold circuits", *Proc. of the 32nd Annual Symp. on Foundations of Computer Science,*1991, pp. 767-776.

[21] Megiddo, M., "On the complexity of polyhedral separability," *Discrete Computational Geometry,* **3**(1988): 325-337.

[22] Muroga, S., *Threshold Logic and its Applications,* John Wiley & Sons Inc., 1971.

[23] Papadimitriou, C. H., Schäffer, A. A., and Yannakakis M., "On the Complexity of Local Search", *Proc. 22nd Annual Symp. Theory Computing,* 1990, pp. 438-445.

[24] Papadimitriou, C.H., and Steiglitz, K., *Combinatorial Optimization: Algorithms and Complexity*, Prentice-Hall, Englewood Cliffs, 1982.

[25] Roychowdhury V. P., Siu K.-Y., and Kailath T., "Classification of Linearly Non-Separable Patterns by Linear Threshold Elements", to appear in *IEEE Trans. on Neural Networks.*

[26] Siegelmann H. T., and Sontag E. D., "On the computational power of neural nets", *Proc. 5th ACM Workshop on Computational Learning Theory,* Pittsburgh, July, 1992.

[27] Siegelmann H. T. and Sontag, E. D., "Neural networks with Real Weights: Analog Computational Complexity," TCS journal, to appear.

[28] Sontag, E.D., "Feedforward nets for interpolation and classification," *J. Comp. Syst. Sci.,* **45**(1992): 20-48.

[29] Yao, X., "Finding Approximate Solutions to NP-hard Problems by Neural Networks is hard", *Information Processing Letters,* **41**(1992): 93-98.

[30] Zhang, X-D., "Complexity of neural network learning in the real number model," preprint, Comp. Sci. Dept., U. Mass., 1992.

Acknowledgement

The work of Bhaskar DasGupta was supported in part by NSF Grant CCR-92-08913. The work of Hava T. Siegelmann and Eduardo Sontag was supported in part by US Air Force Grant AFOSR-91-0343.