On the Complexity of Training Neural Networks with Continuous Activation Functions[‡]

Bhaskar DasGupta[*]
Department of Computer Science
University of Minnesota
Minneapolis, MN 55455–0159
Email: dasgupta@cs.umn.edu

Hava T. Siegelmann[†]
Department of Computer Science
Bar-Ilan University
Ramat-Gan 52900, Israel
hava@bimacs.cs.biu.ac.il

Eduardo Sontag[†]
Department of Mathematics
Rutgers University
New Brunswick, NJ 08903
Email: sontag@control.rutgers.edu

[*] Research supported in part by NSF Grant CCR-92-08913

[†] Research supported in part by US Air Force Grant AFOSR-91-0343

[‡] A preliminary version of this result will appear in 7th ACM Conference on Learning Theory, 1994

Abstract

We deal with computational issues of *loading* a fixed-architecture neural network with a set of positive and negative examples. This is the first result on the hardness of loading a simple 3-node architecture which do not consist of the binary-threshold neurons, but rather utilize a particular continuous activation function, commonly used in the neural network literature.

We observe that the loading problem is polynomial-time if the input dimension is constant. Otherwise, however, *any possible* learning algorithm based on particular fixed architectures faces severe computational barriers. Similar theorems have already been proved by Megiddo and by Blum and Rivest, to the case of binary-threshold networks only. Our theoretical results lend further suggestion to the use of incremental (architecture-changing) techniques for training networks rather than fixed architectures. Furthermore, they imply hardness of learnability in the *probably-approximately-correct* sense as well.

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 paper, 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

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

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

A possible line of work deals with *sample complexity* questions, that is, the quantification of 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 [3, 20, 11, 19]; 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 paper, and the last one provides results for networks employing the standard sigmoid activation function.

A different perspective to learnability questions takes a numerical analysis or approximation theoretic point of view. There 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, 13, 7], which deal with single hidden layer nets, and [8], which dealt with multiple hidden layers.

Yet another direction in which to approach theoretical questions regarding learning by neural networks, and the one that concerns us here, originates with the work of Judd (see for instance [14, 15], as well as the related work [4, 17, 27]). 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 paper, 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. On the other-hand, loading recurrent networks (i.e. networks with feedback loops) is a hard problem. Bruck and Goodman[6] 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[26] 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 paper, 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 [4]. 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 (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. (For the far more complicated architectures considered in Judd's work, in contrast, enough measurements of internal variables are provided that there is essentially no difference between results for varying activations, and the issue does not arise there. However, it is not clear what the consequences are for practical algorithms when the obstructions to learning are due to considering such architectures. In any case, we address here the open problem exactly as posed by Blum and Rivest.)

It turns out that a definite answer to the question posed by Blum and Rivest is not possible. It is shown in [25] 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 in NP (just answer "yes" to the question "do there exist weights that learn the given data?"!). The functions used in the construction in [25] 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, the purely mathematical construct in [25]. 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. (Höffgen[12] proves the hardness of the interpolation problem by sigmoidal nets with two hidden units when the weights are just binary values; however this is different from the problem we are considering). It is the purpose of this paper 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 problem in general.

The rest of the paper is organized as follows:

- In section 2 we introduce the model (in particular, the 2 π-node architecture) and summarize some previous results. We also distinguish the case of fixed versus varying input dimension (and analog versus binary inputs), and observe that the problem is solvable in polynomial time for fixed input dimension using standard linear-programming techniques (see [20] for further positive results on *PAC*-learnability when the input dimension is a fixed constant and the activation functions are piecewise polynomials). In the rest of the paper we concentrate on *binary inputs* only, where 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 binary inputs under the assumption of $RP \neq NP$.

- In section 4 we generalize the hardness of the loading problem to include another similar architectures with more nodes in the hidden layer.

- In section 5 we conclude with some open problems.

Before turning to the next section, we provide a short overview on complexity classes and probabilistic learnability.

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, 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 if and only if any problem Y in NP can be transformed by a polynomial time transformation f to X, such that given an instance I of Y, I has a solution if and only if $f(I)$ has a solution. A problem is NP-complete if and only if 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 $(0\epsilon \leq 1)$ 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 ϵ, 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

A concept is a function $f : \{0,1\}^n \rightarrow \{0,1\}$ where n is an integer. 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 of 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 a single 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:

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

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

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

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

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

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

A class C of representations of an architecture $\mathcal{A}$ is said to be *learnable*[16] if and only if it is (ϵ, δ)-learnable for all ϵ and δ (where $0 < \epsilon, \delta < 1$).

Remark 1.1 *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 the non-learnability of the corresponding concept under the assumption of $RP \neq NP$.

2 Preliminaries and previous works

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 z.

The network C computes a function $f_C : [0,1]^n \to \mathbb{R}$, where n is the input dimension, as follows. The components of the input vector $\vec{x} = (x_1, \ldots, x_n) \in [0,1]^n$ 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 $s_v(x) = \sum_{i=1}^{k} a_i y_i - t_v$, where y_i is the value assigned to v_i and a_i and t_v are weights and threshold 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 $M = \{(\vec{x}, y) \mid x \in [0,1]^n, y \in \{0,1\}\}$, so that $|M| = O(n)$; find weights $\vec{w}$ so that for all pairs $(\vec{x}, y) \in M$:

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

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

We henceforth assume that sink z is restricted to be a threshold gate. This is indeed true for the purpose of analysing the complexity of the decision version of the loading problem for the activation functions that we consider.

For the purpose of this paper 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:

$$\begin{aligned} N_1(\vec{a}, \vec{x}) &= \phi(\sum_{i=1}^{n} a_i x_i), \\ N_2(\vec{b}, \vec{x}) &= \phi(\sum_{i=1}^{n} b_i x_i), \end{aligned}$$

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

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

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

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

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

and the piecewise linear or "saturating" activation functions π (which appears quite frequently in neural networks literature[2, 5, 18, 27]) defined as

$$\pi(x) = \begin{cases} 0 & \text{if } x < 0 \\ x & \text{if } 0 \leq x \leq 1 \\ 1 & \text{if } x > 1 \ . \end{cases} \tag{2}$$

Another model, called the 2-*cascade architecture*, was investigated by Lin and Vitter[17]. 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.

2.3 Loading the k $\mathcal{H}$-Node Architecture

We consider two kinds of inputs: *analog* (with fixed input dimension) and *binary* (with varying input dimension). An analog input is in $[0,1]^n$, where n is a fixed constant. In the binary case, the input is in $\{0,1\}^n$ where n is an input parameter.

Blum and Rivest[4] 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 with binary inputs is NP-complete.

However, when the input is analog (and the 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 [22].

Theorem 2.1 *Let $k > 0$ be an integer. It is possible to load any k $\mathcal{H}$-node architecture in polynomial time if the input dimension is constant.*

Before proving Theorem 2.1, we summarize the related result of Megiddo in [22] regarding polyhedral separability in fixed dimension.

The following definition is due to Megiddo [22]:

Definition 2.1 k-Polyhedral Separability: Given two sets of points A and B in $\mathbb{R}^d$, and an integer $k > 0$, decide whether there exist k hyperplanes

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

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 $\vec{p} \in \mathbb{R}^d$ if $(\vec{x^j})^T\vec{p} > x_0^j$, false if $(\vec{x^j})^T\vec{p} < x_0^j$, 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 $\vec{a} \in A$ and false for each point $\vec{b} \in B$.

The following Lemma is from [22].

Lemma 2.2 [22] 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 of Theorem 2.1. The computational view of the loading problem of analog input is very similar to the model of Lemma 2.2. 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 2.2 holds for the analog inputs 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.□

Remark 2.1 *A k $\mathcal{H}$-node network (where k is a constant) with fixed input dimension is also learnable; this follows as a consequence of a result proven in [20].*

3 The Loading Problem For the 2 π-node Architecture

One can generalize Theorem 2.1 and show that it is possible to load the 2 π-node architecture with analog inputs in polynomial time. In this section we show that the loading problem for the 2 π-node architecture is NP-complete when binary inputs are considered. The main theorem of this section is as follows.

Theorem 3.1 *The loading problem for the 2 π-node architecture (LπAP) with binary inputs is NP-complete.*

A corollary of the above theorem is as follows.

Corollary 3.1 *The class of Boolean functions computable by the 2 π-node architecture with binary inputs is not learnable, unless $RP = NP$.*

To prove theorem 3.1 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

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 $P_1 : \sum_{i=1}^{n} a_i x_i = 0$, $P_2 : \sum_{i=1}^{n} a_i x_i = 1$, $Q_1 : \sum_{i=1}^{n} b_i x_i = 0$, and $Q_2 : \sum_{i=1}^{n} b_i x_i = 1$ (refer to Fig. 2). Let F_{c_1,c_2} denote the set of points which lie on the intersection of two n-dimensional hyperplanes $\sum_{i=1}^{n} a_i x_i = c_1$ and $\sum_{i=1}^{n} b_i x_i = c_2$. Consider the set of points $W = \{F_{0,0}, F_{0,1}, F_{1,0}, F_{1,1}\}$. As all points belonging to the same set $F_{i,j}$ are labeled the same, we consider "labeling sets $F_{i,j}$ in W" rather than the individual points in $\{0,1\}^n$.

Type 1. Either all the sets in W are labeled $'+'$ or all the sets in W are labeled $'-'$. In that case, all the examples are labeled $'+'$ or $'-'$, respectively.

Type 2. Exactly one set in W is labeled $'+'$. Assume that this set is $F_{0,0}$. Then, two different types of separations exist:

(a) There exist two halfspaces

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

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

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 halfspaces of the following form (Fig. 2(b)):

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

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

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

where $0 > \gamma$, $\alpha, \beta \leq \gamma < 0$ (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 set is marked $'+'$, a similar separation is produced.

Type 3. Two sets in W are marked $'+'$ and the remaining two are labeled $'-'$. Because the labeling must be linearly separable, only the following types of classifications are possible:

(a) $F_{0,1}$ and $F_{0,0}$ are $'+'$ (Fig. 2(d)). Then, the input space is partitioned via the three halfspaces:

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

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

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

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

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) $F_{0,0}$ and $F_{1,0}$ are $'+'$ (Fig. 2(c)). Then, the input space is partitioned via the three halfspaces:

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

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

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

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

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) $F_{1,0}$ and $F_{1,1}$ are $'+'$ (similar to Fig. 2(d) with the labeling of $'+'$ and $'-'$ points interchanged). This is the symmetrically opposite case of type 3(a).

(d) $F_{0,1}$ and $F_{1,1}$ are $'+'$ (similar to Fig. 2(c) with the labeling of $'+'$ and $'-'$ points interchanged). This is the symmetrically opposite case of type 3(b).

Type 4. Three sets in W 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: Does there exist a satisfying assignment?

Note that the clause $(v_i \vee v_j)$ (resp. $((\neg v_i) \vee (\neg v_j))$) is equivalent to both the implications $(\neg v_i \rightarrow v_j)$ and $(\neg v_j \rightarrow v_i)$ (resp. $(v_i \rightarrow \neg v_j)$ and $(v_j \rightarrow \neg v_i)$, while the clause v_i (resp. $\neg v_i$) is equivalent to the implication $(\neg v_i \rightarrow v_i)$ (resp. $(v_i \rightarrow \neg v_i)$) 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 $V = \{d_i, \overline{d_i} \mid v_i$ is a variable$\}$, and $E = \{(l_i, l_j) \mid (i,j \in \{1,\ldots,n\}), (l_i \in \{d_i, \overline{d_i}\}), (l_j \in \{d_j, \overline{d_j}\}), (g_i \rightarrow g_j) \in D$ where g_i (resp. g_j) is v_i (resp. v_j) if l_i (resp. l_j) is d_i (resp. d_j) and $\neg v_i$ (resp. $\neg v_j$) otherwise$\}$. Note that an edge (l_i, l_j) in E is directed from l_i to l_j. 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 it exists or, stops if there is no one (see, for example, [24, pp. 377-378]):

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 $v_r \rightarrow \neg v_s$ (resp. $\neg v_r \rightarrow v_s$) 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 $v_r \rightarrow \neg v_s$ (resp. $\neg v_r \rightarrow v_s$) 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:

> *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 $O(\mid V \mid) = O(n)$ 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, named (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 it exists, where, for all $i,j \in \{1,2,\ldots,k-1\}$ $i \neq j$:

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

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 3.2 *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 $d_1 \rightarrow \overline{d_2} \rightarrow d_3 \rightarrow \cdots \rightarrow d_r \rightarrow \overline{d_1} \rightarrow d_{1'} \rightarrow \overline{d_{2'}} \rightarrow d_{3'} \rightarrow \cdots \rightarrow \overline{d_{s'}} \rightarrow d_1$, where r and s' are two positive integers and $x \rightarrow y$ 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 $d_{t'} \rightarrow \overline{d_{t'}}$ or $\overline{d_{t'}} \rightarrow d_{t'}$ 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 $d_{t'} \rightarrow \overline{d_{t'}}$, (that is, the associated clause is $\neg x_{t'}$). Consider the path $P' : \overline{d_1} \rightsquigarrow d_{t'}$. P' is of the form $\overline{d_1} \rightarrow d_{1'} \rightarrow \overline{d_{2'}} \rightarrow \cdots \rightarrow \overline{d_{t'-1}} \rightarrow d_{t'}$ and t' is odd ($t' = 1$ is possible). Now, due to coloring condition (a) and (b), $b_{t'}$ is colored red (see below).

	$i = 1$	$i = 1'$	$i = 2'$	$\ldots$	$i = t' - 1$	$i = t'$
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 $d_{t'} \rightarrow \overline{d_{t'}}$. 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 $H_3 : (a \vec{+} b)\vec{x} > c_0$ 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 3.3 *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:

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

we create the instance I' of the 3-hyperplane problem (like in [4]):

> $\star$ 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.

$\Rightarrow$

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 of type (a) of the 3-hyperplane problem.

$\Leftarrow$

(A) If there is a separation of type (a), the solution of the set-splitting is analogous to [4]: 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 3.2. 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 separations of type 3, and then we exclude in separations of type 4 in Lemma 3.2.

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 proof of Lemmas 3.1 and 3.2 involve a long case-by-case analysis and is provided in the appendix.

Consider the following 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 [4]:

(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 3.1. 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 [23] one can restrict all the weights to have at most $O(n \log n)$ bits. Thus, 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 $(|S| + 5)$-dimensional 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 if and only if there exists a solution to I'. Given a solution to the (2,3)-SSP, by lemma 3.1(part(b)) and the result in [4] the two solution halfspaces to I' are as follows (assume the last 5 dimensions are x_{n+1} to x_{n+5}):

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

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

where

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

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

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

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

Conversely, given a solution to I', by Lemma 3.1(part (a)), Lemma 3.2 and the result in [4] (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 3.3. □

Remark 3.1 *From the above proof of theorem 3.1 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 3.1 which states that the functions computable by the 2 π-node architecture with binary inputs 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 such that it is not (ϵ, δ)-learnable).

Proof of Corollary 3.1. The proof uses a similar technique to the one applied in the proof of theorem 9 of [16]. 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 3.1):

> 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 3.1 (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 the choice of ϵ and δ (and, by Theorem 3.1), 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 3.1, 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 $RP = NP$. □

Remark 3.2 *In a similar manner, the subsequent NP-completeness result of the loading problem proven in the next section can be used to provide a proof of the impossibility of learnability of the associated concept under the assumption of $RP \neq NP$.*

4 Another Architecture Which is Hard to Load

In this section we discuss an extension of the NP-completeness result. Inspired by Blum and Rivest[4] who considered loading a few variations of the k $\mathcal{H}$-node network, in which all activations functions were discrete; we consider a variations of the k Φ-node architecture in which two nodes compute continuous activation functions. The result of this section has theoretical importance only, as binary threshold units are not popular in applications.

Consider a unit G that computes $\mathcal{H}(\sum_{i=1}^{n} \alpha_i x_i - \eta)$, where α_i's are real constants and x_1 to x_n are input variables which assume any real value in $[0,1]$. We say that this unit G computes a Boolean NAND (i.e., *negated* AND) function of its inputs provided its weights and threshold satisfy the following requirements:

$$\alpha_i < \eta < 0 \quad 1 \leq i \leq n \tag{3}$$

For justification, assume that the inputs to node G are binary. Then, the output of G is one if and only if *all* its inputs are zeroes.

Our model consists of $r+2$ hidden nodes $N_1, N_2, \ldots, N_{r+2}$ (where r is a fixed polynomial in n, the number of inputs) and one output node. The nodes $N_1, N_2, \ldots, N_r$ in the hidden layer compute the binary threshold functions $\mathcal{H}$, and the two remaining hidden nodes N_{r+1} and N_{r+2} compute the "saturating activation" functions π (equation 2). The output node N_{r+3} computes a Boolean NAND function (Fig. 3). We term this as the *"Restricted"* $(2,r)$ $(\pi, \mathcal{H})$*-node* architecture.

One can generalise Theorem 2.1 and show that the "Restricted" $(2,r)$ $(\pi, \mathcal{H})$-node architecture can be loaded in polynomial time in the case when the input dimension is fixed. However, the loading problem becomes NP-complete when (binary) inputs of varying dimensions are considered. The main theorem of this section is as follows.

Theorem 4.1 *The loading problem for the "Restricted"* $(2,r)$ $(\pi, \mathcal{H})$*-node architecture with binary inputs of varying dimension is NP-complete.*

Before proving Theorem 4.1 we show, given an instance I of the $(2,3)$-SSP, how to construct an instance I' of the $(r+2,3)$-SSP such that I has a solution if and only if I' has one.

Let $I = (S, C)$ be a given instance of the $(2,3)$-SSP. We construct I' by adding $2r+2$ new elements $Y = \{y_i \mid 1 \leq i \leq 2r+2\}$ and creating the following new sets:

- Create the sets $\{s_i, y_j, y_k\}$ for all $1 \leq i \leq n$, $1 \leq j,k \leq 2r+2$, $j \neq k$. This ensures that if a set in a solution of the set-splitting problem contains an element of S, it may contain at most one more element of Y.
- Create the sets $\{y_i, y_j, y_k\}$ for all $1 \leq i,j,k \leq 2r+2$, $i \neq j \neq k$. This ensures that no set in a solution of the set-splitting problem may contain more than two elements of Y.

Let $I' = (S'.C')$ be the new instance of the $(r+2,3)$-SSP, where $S' = S \cup Y$, and C' contains all the sets of C and the additional sets as described above.

Lemma 4.1 *The instance I' of the $(r+2,3)$-SSP has a solution if and only if the instance I of the $(2,3)$-SSP has a solution.*

Proof.

$\Leftarrow$

Let (S_1, S_2) be a solution of I. Then, a solution $(T_1, T_2, \cdots, T_{r+2})$ of the instance I' is as follows:

$$T_i = \{y_{2i-1}, y_{2i}\} \quad \text{for} \quad 1 \leq i \leq r$$

$$T_{r+1} = S_1 \cup \{y_{2r+1}\}$$

$$T_{r+2} = S_2 \cup \{y_{2r+2}\}$$

$\Rightarrow$

Let $(T_1, T_2, \cdots, T_{r+2})$ be a solution of I'.

Case 1. There are at most two sets of $T_1, T_2, \cdots, T_{r+2}$ which contain all the elements of S. Then these two sets constitute a solution of I.

Example: Let $n = 5$. If $T_1 = \{x_1, x_2, x_3, y_2\}$ and $T_2 = \{x_4, x_5, y_4\}$ are the two sets that contain all the elements of $S = \{x_1, x_2, x_3, x_4, x_5\}$, then the two solution sets S_1 and S_2 are:

$$\begin{aligned} S_1 &= \{x_1, x_2, x_3\} \\ S_2 &= \{x_4, x_5\} \end{aligned}$$

Case 2. Otherwise, there are m ($m \geq 3$) sets, $T_1, \ldots, T_m$, each containing a distinct element of S. At most one element of Y occurs in each T_i (since two elements of Y cannot be in the same set with an element of S without violating the set-splitting constraint), hence $m < r+2$. So, there are $r+2-m$ remaining sets in the solution of the instance I' and at least $2r+2-m$ elements of Y to be placed in those sets. By the pigeonhole principle, one of these remaining $r+2-m$ sets must contain at least three elements of Y (since $m \geq 3$), thus violating the set-splitting constraint. So, case 2 is not possible. □

Proof of Theorem 4.1. The $'+'$ and $'-'$ points are $(r+3)$-polyhedrally separated by the output of the network in which the Boolean formula for the polyhedral separation is the formula for the NAND function. Hence, from the result of [23] we can restrict all the weights to have at most $p(n+r)$ bits (where $p(x)$ is some polynomial in x). Since r is a polynomial in n, any "guessed" solution may be verified in polynomial time. So, the problem is in NP.

We next show that the problem is NP-complete. Given an instance I of the $(2,3)$-SSP, we construct an instance I' of the "Restricted" $(2,r)$ $(\pi,\mathcal{H})$-node architecture as follows: We create first an instance I'' of the $(r+2,3)$-SSP (see Lemma 4.1). We then add the following labeled points, thus constructing the associated instance I'.

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

$\Leftarrow$

Given a solution (S_1, S_2) of I, we construct a solution $(T_1, T_2, \cdots, T_{r+2})$ of I'' as described in the proof of Lemma 4.1. Consider the following $r+2$ halfspaces:

$$H_i : \sum_{j=1}^{n} \delta_{i,j} x_j > -\frac{1}{2} \quad (1 \le i \le r+2)$$

where,

$$\delta_{i,j} = \begin{cases} -1 & \text{if } s_j \in T_i \\ 2 & \text{otherwise} \end{cases}$$

All labeled points of I' are separated by these halfspaces: the $'+'$ points lie in $\wedge_{i=1}^{r+2} H_i$ and the $'-'$ points lie in $\vee_{i=1}^{r+2} \overline{H_i}$.

We map the $r+2$ halfspaces to the "Restricted" $(2,r)$ $(\pi,\mathcal{H})$-node architecture as follows. The hidden nodes compute:

$$\begin{aligned} N_i &= \mathcal{H}[-(\sum_{j=1}^{n} \delta_{i,j} x_j)] \qquad i = 1 \ldots r \\ N_{r+1} &= \pi[-(\sum_{j=1}^{n} \delta_{r+1,j} x_j)] \\ N_{r+2} &= \pi[-(\sum_{j=1}^{n} \delta_{r+2,j} x_j)] \ , \end{aligned}$$

and the output node N_{r+3} computes:

$$N_{r+3} = \begin{cases} 1 & \text{if } \ -(\sum_{i=1}^{r+2} N_i) > -\frac{1}{2} \\ 0 & \text{if } \ -(\sum_{i=1}^{r+2} N_i) \le -\frac{1}{2} \end{cases}$$

$\Rightarrow$

Conversely, given a solution to the instance I', we construct a solution to I. The classification produced by the "Restricted" $(2,r)$ $(\pi,\mathcal{H})$-node architecture is as follows. Each hidden threshold node N_i $(1 \le i \le r)$ defines a halfspace H_i:

$$H_i : \sum_{j=1}^{n} \delta_{i,j} x_j > \eta_i$$

for some real numbers $\delta_{i,1}, \cdots, \delta_{i,n}$ and η_i. From the classifications produced by the 2 π-node architecture as described in section 3.1 and since the output node N_{r+3} computes a Boolean NAND function, there are at most j halfspaces (for $2 \leq j \leq 3$) corresponding to the nodes N_{r+1} and N_{r+2}:

$$H_{r+1} : \sum_{i=1}^{n} a_i x_i > a_0$$

$$H_{r+2} : \sum_{i=1}^{n} b_i x_i > b_0$$

$$H_t : \sum_{i=1}^{n} (a_i + b_i) x_i > c_0 \quad (r+3 \leq t \leq r+j)$$

All the $'+'$ points lie in $\wedge_{i=1}^{r+j} H_i$, and all the $'-'$ points lie in $\vee_{i=1}^{r+j} \overline{H_i}$.

Let T_i be the $'-'$ points separated from the origin by the halfspace H_i of the output of the network (for $1 \leq i \leq r+j$, $2 \leq j \leq 3$). No T_i contains three elements of the same set of the instance I', otherwise the set itself will be in T_i as well, contradicting its positive labeling. Consider the sets T_i as the solution of the instance I''. By Theorem 3.3 the sets T_{r+1} to T_{r+j} can be combined to 2 sets, say T'_{r+1} and T'_{r+2}, without violating the set-splitting constraints. Hence, we have $r+2$ sets, $T_1, T_2, \cdots, T_r, T'_{r+1}, T'_{r+2}$, as the $r+2$ solution sets for the instance I'' which satisfy the set-splitting constraint. Hence, by Lemma 4.1 we can construct the two solution sets (S_1, S_2) of the instance I. $\Box$

5 Conclusion and Open Problems

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 ([4]) and a net with a specific (somehow artificial) analog activation function has a fast loading ([25]). Unfortunately, our proof does not seem to generalize to other activation functions. The following open problems may be worth investigating further:

- Does the NP-completeness result hold for the 2 σ-node architecture, where σ is a more complicated activation function (e.g. when σ is the quadratic spline activation function or the standard sigmoid)?
- 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, see [21].
- Is there a characterization of the activation functions for which the loading problem is intractable?

6 Acknowledgments.

The first author wishes to thank Piotr Berman and Vwani P. Roychowdhury for helpful discussions.

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] Batruni, R., “A multilayer neural network with piecewise-linear structure and back-propagation learning,” *IEEE Transactions on Neural Networks* **2**(1991): 395-403.

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

[4] Blum, A., and Rivest, R. L., “Training a 3-node neural network is NP-complete,” *Neural Networks,* **5**(1992): 117-127.

[5] Brown, J., M. Garber, and S. Vanable, “Artificial neural network on a SIMD architecture,” in *Proc. 2nd Symposium on the Frontier of Massively Parallel Computation*, Fairfax, VA, 1988, pp. 43-47.

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

[7] 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.

[8] 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.

[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] Höffgen K-U., “Computational limitations on training sigmoidal neural networks,” *Information Processing Letters*, 46(1993), pp. 269-274.

[13] 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.

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

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

[16] 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.

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

[18] Lippmann, R., "An introduction to computing with neural nets," *IEEE Acoustics, Speech, and Signal Processing Magazine*, 1987, pp. 4-22.

[19] 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.

[20] 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 .

[21] 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.

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

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

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

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

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

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

APPENDIX

Proof of Lemma 3.1. **(a)** Since the origin is labeled $'+'$ it is not possible for 3 halfspaces of type 3(c) or type 3(d) to correctly classify the above set of points in the 2-dimensional hypercube.

Note that the origin is labeled $'+'$, and hence it must lie in at least one of H_1 or H_2 for the three halfspaces of type 3(a) or type 3(b).

Consider the halfspaces as in type 3(a). There are two cases:

Case 1. $\beta > 0$. Since $\overline{H_1} \subseteq \overline{H_2}$ and $H_2 \subseteq H_1$, all the $'+'$ points must belong to H_1 and all the $'-'$ points must belong to $\overline{H_2}$. Hence, (0,1) and (1,0) must belong to $\overline{H_2}$ and we have

$$\begin{aligned} \alpha a_1 &\leq \gamma && (4) \\ \alpha a_2 &\leq \gamma && (5) \end{aligned}$$

Adding inequalities 4 and 5, and since $\gamma < 0$, we get

$$\alpha(a_1 + a_2) \leq 2\gamma < \gamma < 0 \qquad (6)$$

Inequality 6 implies that the $'+'$ point (1,1) belongs to $\overline{H_2}$ and hence in $H_1 \wedge H_3$. Hence, we must have

$$\begin{aligned} \alpha(a_1 + a_2) &> \gamma - \beta && (7) \\ \alpha(a_1 + a_2) + \beta(b_1 + b_2) &> \gamma && (8) \end{aligned}$$

We find the following three cases:

Case 1.1. Both the $'-'$ points (0,1) and (1,0) belong to $\overline{H_2} \wedge \overline{H_3}$. Then, we must have

$$\begin{aligned} \alpha a_1 + \beta b_1 &\leq \gamma && (9) \\ \alpha a_2 + \beta b_2 &\leq \gamma && (10) \\ & && (11) \end{aligned}$$

Adding inequalities 9 and 10 and since $\gamma < 0$ we get

$$\alpha(a_1 + a_2) + \beta(b_1 + b_2) \leq 2\gamma < \gamma \qquad (12)$$

Inequality 12 contradicts inequality 8.

Case 1.2. Both the $'-'$ points belong to $\overline{H_1}$. Then, we must have

$$\begin{aligned} \alpha a_1 &\leq \gamma - \beta && (13) \\ \alpha a_2 &\leq \gamma - \beta && (14) \\ & && (15) \end{aligned}$$

Adding inequalities 13 and 14 and since $\gamma < \beta$, we get

$$\alpha(a_1 + a_2) \leq 2(\gamma - \beta) < \gamma - \beta \qquad (16)$$

Inequality 16 contradicts inequality 7.

Case 1.3. One of the negative points belongs to $\overline{H_1}$ but not to $\overline{H_2} \wedge \overline{H_3}$ and the other negative point belongs to $\overline{H_2} \wedge \overline{H_3}$ but not to $\overline{H_1}$.

Case 1.3.1. (0,1) belongs to $\overline{H_2} \wedge \overline{H_3}$ and (1,0) belongs to $\overline{H_1}$. Since (1,0) belongs to $\overline{H_1}$, we must have

$$\alpha a_1 \leq \gamma - \beta \tag{17}$$

From inequalities 7 and 17 we must have $\alpha a_2 > 0$. On the other hand, inequality 5 claims that $\alpha a_2 \leq \gamma$. We get $0 < \gamma$, which is impossible since $\gamma < 0$.

Case 1.3.2. (1,0) belongs to $\overline{H_2} \wedge \overline{H_3}$ and (0,1) belongs to $\overline{H_1}$. Similar to case 3.1.

Case 2. $\beta < 0$. Again, since H_1 and H_2 are parallel, all the $'-'$ points must lie in $\overline{H_1}$, and all the $'+'$ points must belong to H_2. Hence (1,0) and (0,1) must belong to $\overline{H_1}$ which gives

$$\begin{align} \alpha a_1 &\leq \gamma - \beta \tag{18}\\ \alpha a_2 &\leq \gamma - \beta \tag{19}\\ & \tag{20} \end{align}$$

By adding inequalities 18 and 19, we get

$$\alpha(a_1 + a_2) \leq 2\gamma - 2\beta \tag{21}$$

Since $\beta > \gamma$, we have $2\gamma - 2\beta < \gamma - \beta$. Hence, from inequality 21 we get $\alpha(a_1 + a_2) < \gamma - \beta$. Hence, the point (1,1) lies in $\overline{H_1}$, and hence in $H_2 \wedge H_3$. Then, we have

$$\begin{align} \alpha(a_1 + b_1) + \beta(b_1 + b_2) &> \gamma \tag{22}\\ \alpha(a_1 + a_2) &> \gamma \tag{23}\\ & \tag{24} \end{align}$$

Now, we have the following 3 cases:

Case 2.1. Both the negative points (1,0),(0,1) lie in $\overline{H_3} \wedge \overline{H_1}$. Then, we have

$$\begin{align} \alpha a_1 + \beta b_1 &\leq \gamma \tag{25}\\ \alpha a_2 + \beta b_2 &\leq \gamma \tag{26}\\ & \tag{27} \end{align}$$

Adding inequalities 25 and 26 we get

$$\alpha(a_1 + a_2) + \beta(b_1 + b_2) \leq 2\gamma \tag{28}$$

From inequalities 22 and 28 we get $\gamma < \alpha(a_1 + a_2) + \beta(b_1 + b_2) \leq 2\gamma$, which is impossible since $\gamma < 0$.

Case 2.2. Both the negative points (0,1),(1,0) lie in $\overline{H_2}$. Then, we have

$$\begin{align} \alpha a_1 &\leq \gamma \tag{29}\\ \alpha a_2 &\leq \gamma \tag{30}\\ & \tag{31} \end{align}$$

Adding inequalities 29 and 30 we get $\alpha(a_1 + a_2) \leq 2\gamma < \gamma$ (since $\gamma < 0$) which contradicts inequality 23.

Case 2.3. One of the two negative points lie in $\overline{H_2}$ and the other one lie in $\overline{H_1} \wedge \overline{H_3}$ but not $\overline{H_2}$.

Case 2.3.1. (1,0) lies in $\overline{H_2}$ and (0,1) lies in $\overline{H_1} \wedge \overline{H_3}$ but not in $\overline{H_2}$. Hence,

$$\begin{aligned} \alpha a_1 &\leq \gamma && (32)\\ \alpha a_2 + \beta b_2 &\leq \gamma && (33)\\ \alpha a_2 &> \gamma && (34)\\ & && (35) \end{aligned}$$

From inequalities 23 and 32 we have $\alpha a_2 > 0$. On the other hand, from inequality 19 we have $\alpha a_2 \leq \gamma - \beta$. It implies that $0 < \gamma - \beta$ which contradicts $\beta > \gamma$.

Case 2.3.2. (0,1) lies in $\overline{H_2}$ and (1,0) lies in $\overline{H_1} \wedge \overline{H_3}$ but not in $\overline{H_2}$. Similar to case 2.3.1.

The proof for the type 3(b) halfspaces is similar to that of type 3(a) (by interchanging the roles of the parameters α and β).

(b). The following are a set of 2 possible halfspaces:

$$x_1 - x_2 > -\frac{1}{2}$$

$$-x_1 + x_2 > -\frac{1}{2}\square$$

Proof of Lemma 3.2. The case of two halfspaces of type 4(a) follows from the result of [4]. We prove the case for halfspaces of type 4(b):

Let $H_1 : \sum_{i=1}^{3} a_i x_i > a_0$, $H_2 : \sum_{i=1}^{3} b_i x_i > b_0$ and $H_3 : \sum_{i=1}^{3} c_i x_i > c_0$ be the three halfspaces of type 4(b), where $c_i = a_i + b_i$ for $1 \leq i \leq 3$ (assuming (x_1, x_2, x_3) is the input). The following observations are true:

(i) If $a_0 < 0$ (resp. $b_0 < 0$, $c_0 < 0$) then all the examples in A except for the origin lie in $\overline{H_1}$ (resp. $\overline{H_2}$, $\overline{H_3}$). The reason is as follows. If $a_0 < 0$, then since (0,0,1), (0,1,0) and (0,0,1) are $'-'$, we must have $a_1, a_2, a_3 < a_0$. However, then since $a_0 < 0$, $a_1 + a_3 < 2a_0 < a_0$, $a_2 + a_3 < 2a_0 < a_0$, and $a_1 + a_2 + a_3 < 3a_0 < a_0$, hence the claim follows.

(ii) Consider the same set of examples as in A except that now the origin is not labeled. Then, there does not exist a single hyperplane that separates the $'+'$ and $'-'$ points in this set. Assume it does, and let $H : ax_1 + bx_2 + cx_3 > d$ be the hyperplane. Since (1,1,1) is $'-'$, we must have

$$a + b + c \leq d \tag{36}$$

Since (1,0,1) and (0,1,1) are $'+'$, we must have $a + c > d$, $b + c > d$. Adding the last two inequalities we get

$$a + b + 2c > 2d \tag{37}$$

From inequalities 36 and 37 we get

$$2d - c < a + b + c \leq d \Leftrightarrow d < c$$

which implies that the $'-'$ point (0,0,1) belongs to H, a contradiction!

Since the origin is classified as $'+'$ by at least one of the hyperplanes, at least one of a_0, b_0 and c_0 must be negative. We consider the following cases:

Case 1. $a_0, b_0, c_0 < 0$. By observation (i) above all the $'+'$ points except the origin lie in $\wedge_{i=1}^{3} \overline{H_i}$, a contradiction!

Case 2. Two of a_0, b_0, c_0, say a_0 and b_0, are negative.

By observation (i) above all the points (except the origin) are classified as $'-'$ by two of the three halfspaces, namely halfspaces H_1 and H_2, and by observation (ii) above the remaining halfspace H_3 cannot correctly classify all of them.

Case 3. One of a_0, b_0 and c_0 is negative.

Case 3.1. $a_0 < 0$, $b_0, c_0 \geq 0$.

By observation (i) above, all the points except for the origin, lie in $\overline{H_1}$. By observation (ii) above both the $'+'$ points (other than the origin) cannot be correctly classified by H_2 alone or H_3 alone.

Case 3.1.1. (1,0,1) lies in H_2 and (0,1,1) lies in H_3.

Considering the $'+'$ and $'-'$ points (other than the origin) and the corresponding classifications by the hyperplanes H_1, H_2 and H_3, we have the following set of inequalities:

$$\begin{aligned} a_2 &\leq a_0 < 0 \\ b_1 + b_2 + b_3 &\leq b_0 \\ c_3 &\leq c_0 \\ b_1 + b_3 &> b_0 > 0 \\ c_2 + c_3 &> c_0 > 0 \end{aligned}$$

Since $b_1 + b_2 + b_3 \leq b_0$ and $b_1 + b_3 > b_0$, we have $b_2 < 0$. Since $c_2 + c_3 > c_0 \geq 0$, but $c_3 < c_0$, we must have $c_2 > 0$.

However, since $c_2 = a_2 + b_2$, and $a_2 < 0$, $b_2 < 0$, so $c_2 < 0$, hence a contradiction!

Case 3.1.2. (1,0,1) lies in H_3 and (0,1,1) lies in H_2. Similar to case 3.1.1.

Case 3.2. $b_0 < 0$, $a_0, c_0 \geq 0$. Similar to case 3.1.

Case 3.3. $a_0, b_0 \geq 0$, $c_0 < 0$.

By observation (i) above all the points except for the origin lies in $\overline{H_3}$. By observation (ii) above both the $'+'$ points (other than the origin) cannot be correctly classified by H_1 alone or H_2 alone.

Case 3.3.1. (1,0,1) lies in H_1 and (0,1,1) lies in H_2.

Considering the $'+'$ and $'-'$ points (other than the origin) and the corresponding classifications by the hyperplanes H_1, H_2 and H_3, we have the following set of inequalities:

$$\begin{aligned} a_1 &\leq a_0 \\ a_1 + a_3 &> a_0 \geq 0 \\ b_2 &\leq b_0 \\ b_2 + b_3 &> b_0 \geq 0 \\ c_3 &\leq c_0 < 0 \end{aligned}$$

Since $b_2 \leq b_0$, and $b_2 + b_3 > b_0 \geq 0$, we must have $b_3 > 0$. Since $a_1 \leq a_0$, and $a_1 + a_3 > a_0 \geq 0$, we must have $a_3 > 0$. So, $c_3 = a_3 + b_3 > 0$, which contradicts the inequality $c_3 < 0$ above.

Case 3.3.2. (1,0,1) lies in H_2 and (0,1,1) lies in H_1. Similar to case 3.3.1. □

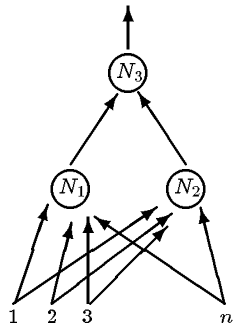

Figure 1:

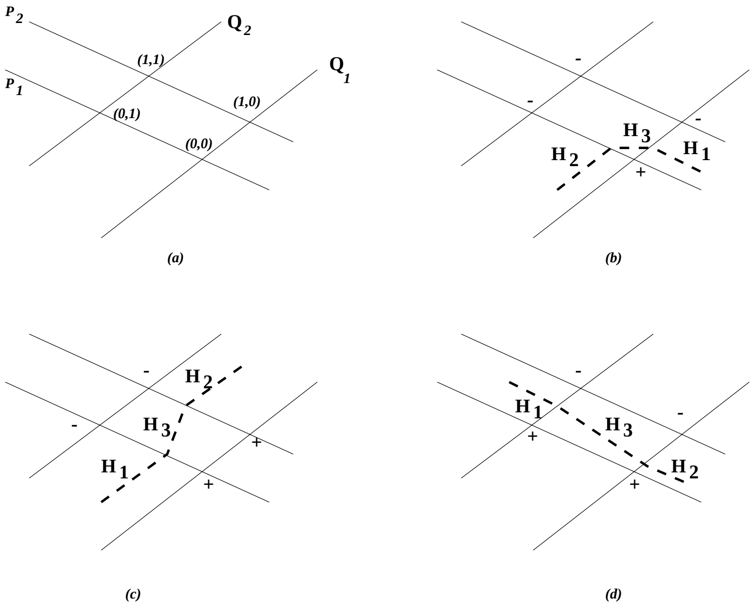

Figure 2:

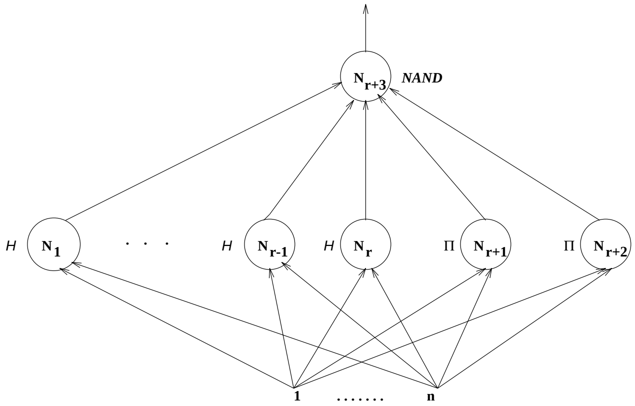

Figure 3:

Figure 1: A 2 Φ-node architecture

Figure 2: Different classifications produced by the 3-node network corresponding to different labeling of the points in the intersection of the hyperplanes.

Figure 3: The "Restricted" $(2, r)$ (π, H)-node network.