Complexity of Neural Network Learning in the Real Number Model

Xiao-Dong(Nick) Zhang
zhang@cerc.wvu.wvnet.edu

Abstract

This paper presents several results concerning complexity of neural network learning in the Blum-Shub-Smale real number model. Two different types of loading problems are defined in this model. The relationships between loading problems and some problems in the real number model are studied. First, we prove the polynomial equivalence between the second loading problem and linear programming problem in the real number model. Second, we show that the TSP problem can be polynomially reduced to the first loading problem. At last, we prove loading a neural network consisting of linear sum units and linear threshold units is NP-complete.

1 Introduction

Judd [6] proved that neural network learning in the conventional computation model is *NP*-complete. He defined a *loading problem* for connectionist learning and then proved that the *3SAT* problem can be polynomially reduced to the *loading problem*. Judd's loading problem is defined using conventional computation models, i.e. based on the Turing machine. Each node in the network implements a binary Boolean function. The complexity measure is, therefore, based on bits.

As pointed out by Blum, Shub and Smale in [1] [2], many areas, e.g., robotics, computer vision, dynamic system theory and large scale scientific computations require a real number model where the complexity measure is based on the number of basic algebraic operations. They defined a real number computation model. This model can be generalized to any algebraic ring. The importance of the BSS model lies in the fact that the *4-Feasibility* Problem over $\mathcal{R}$ is defined and proved to be *NP*-complete. In words, the *4-Feasibility Problem* is: Given a multivariable polynomial f of degree 4 with coefficients from $\mathcal{R}$, is $f(x) = 0$ feasible over $\mathcal{R}$, i.e., does $f(x) = 0$ have a solution over $\mathcal{R}$? Canny has found exponential complexity bounds for *4-Feasibility* problem [3]. The problems of *2-Feasibility* and *3-Feasibility* are proved to be in P [11]. There are a couple of problems whose integer or rational number versions are known to be in P or NP, but whose real number versions are still open, e.g., the linear programming problem (*LPP*) and the traveling salesman problem (*TSP*).

Hong [5] showed that non-uniform aggregate of Boolean gates can polynomially simulate neural networks with real valued weights and threshold. His work is along the line of *infinite precision discrete* models [7] since real numbers are not considered as independent, unit entities.

This paper presents several results concerning complexity of neural network learning in the Blum-Shub-Smale real number model.

2 The Loading Problem in the Real Number Model

A neural net which only consists of linear threshold units can perform a mapping $\mathcal{R}^s \to \{0,1\}^t$. In order to perform the mapping $\mathcal{R}^s \to \mathcal{R}^t$, we should allow linear sum units, which can be characterized by $f(x) = w \cdot x$. In [9], a more powerful unit type called *sigma-pi* unit is described. In this research this unit type will not be involved. In some cases we allow some of the weights of a neuron to be fixed. The fixed weights can not be adjusted in a learning process. The advantage of fixing weights of a neuron is that the threshold value can be varied in some range by changing input values on edges with fixed weights.

An architecture specifies the interconnection structure of a network. The architecture of a network should specify whether a node computes linear sum or linear threshold functions, and which edges have fixed weights. In this research we just consider *feed-*

0-8186-3420-0/93 $03.00 © 1993 IEEE

forward neural net. The architecture of a *feedforward* neural net is a directed acyclic graph G with several designated input nodes and output nodes. The input nodes have no incoming edges. The nodes of G that are neither input nor output nodes are called *hidden units*.

The type of learning considered here is supervised learning. In this paradigm input patterns (called *stimuli*) are presented to a machine paired with their desired output patterns (called *responses*). The object of the learning machine is to remember all the associations presented during a training phase so that in future tests the machine will be able to emit the associated response for any given stimulus. Both stimulus σ and response ρ are vectors of reals, that is, $\sigma \in \mathcal{R}^s$ and $\rho \in \mathcal{R}^t$. The output from a net is also a real vector, denoted by $\theta \in \mathcal{R}^t$. An output vector, θ, is said to agree with a response vector, ρ, if $\theta = \rho$. The notation for such agreement is $\theta \models \rho$. Each stimulus/response pair, (σ, ρ), is called an SR *item*. A *task* is a set of SR items that the machine is required to learn. Computationally, a task can be viewed as a collection of constraints on the mapping that a network is allowed to perform.

Loading is the process of assigning appropriate weight vectors W to each node in the architecture so that the output of the network can perform the given task. So the architecture A defines a mapping from the space of stimuli to the space of responses:

$$M_W^A : \mathcal{R}^s \to \mathcal{R}^t.$$

The values of A and W fully define the behavior of a network.

More formally, the problem can be stated as follows:

Instance: An architecture A and a task T.

Question: Does there exist a W for A such that

$$T \subseteq \{(\sigma, \rho) : M_W^A(\sigma) \text{ agrees with } \rho\}?$$

The above loading problem is of the same form of Judd's original loading problem. Realistic learning algorithms, however, are not just to decide whether or not there exist weight vectors to make given architectures to perform given tasks. Learning algorithms actually can always find weight vectors if given tasks are learnable.

We can define another loading problem as follows:

Instance: An architecture A and a task T.

Question: Find a W for A such that

$$T \subseteq \{(\sigma, \rho) : M_W^A(\sigma) \text{ agrees with } \rho\}.$$

To distinguish we call the first problem the loading problem 1, and the above problem the loading problem 2. The loading problem 1 is a decision problem. The loading problem 2 is a search problem. Obviously, the loading problem 2 is harder than the loading problem 1.

3 Complexity of Loading Problem

3.1 Single Neuron and Linear Programming

First we consider the single neuron problem and the Linear Programming problem.

In this problem we only consider the case of loading problem 2, i.e., trying to find a weight vector for a single neuron so that it can learn a given task.

The linear programming problem is to

$$\begin{aligned} \textit{maximize}: \quad & z = \sum_{j=1}^{n} c_j \cdot x_j \\ \textit{subject to}: \quad & \sum_{j=1}^{n} a_{ij} \cdot x_j \le b_i \\ & \textit{for all } 1 \le i \le m. \end{aligned} \qquad (1)$$

The dual problem of the above is to

$$\begin{aligned} \textit{minimize}: \quad & w = \sum_{i=1}^{m} b_i \cdot y_i \\ \textit{subject to}: \quad & \sum_{i=1}^{m} a_{ij} \cdot y_i \ge c_j \\ & \textit{for all } 1 \le j \le n. \end{aligned} \qquad (2)$$

All the coefficients a, b and c are real numbers. So are the variables x and y.

According to duality theorem of linear programming theory, the given primal problem (1) gets optimal value iff $z \ge w$. In that case the dual problem (2) also reaches its optimal solution.

LEMMA 1. Loading a single neuron can be polynomially reduced to LP.

Proof: This direction is relatively easy. For each instance in the given task, we can always construct a constraint which is of the form of linear inequality. In their early book [4], Duda and Hart have described how to transform a loading problem to a linear programming problem. The transformation procedure takes only $O(m \cdot n)$ algebraic operations. □

LEMMA 2. LP can be polynomially reduced to loading a single neuron.

Proof: The key point here is to construct a learning task and make a neuron memorize the given task. If the neuron can learn the given task, we prove the linear programming problem can be solved. The task consists of $2m + 2n + 1$ instances as shown in Figure 1. It should be noted the neuron in Figure 1 has a special incoming edge with a fixed weight 1. The last element of each instance vector is the ***response***. The other elements are ***stimuli***.

The first set of instances $I = \{I_1, ..., I_{n+m}\}$ can make all x_i and y_j be no less than 0.

The second set of instances $J = \{J_1, ..., J_m\}$ can force the neuron satisfy the constraints of the primal problem (1). The third set of instances $K = \{K_1, ..., K_n\}$ can force the neuron satisfy the constraints of dual problem (2)

The last instance L_1 can assure the optimal condition $z \geq w$.

So if the neuron learns the given task, the weight vector of the neuron constitutes the solution to problems (1) and (2). □

THEOREM 3. Loading a single neuron is polynomially equivalent to LP.

Proof: Immediately follows Lemma 1 and Lemma 2. □

3.2 Layered Threshold Net and TSP

We consider the decision version of *TSP* where the minimization of tour length z becomes deciding whether or not there exist a tour such that its length $z \leq K$, K is a real number.

THEOREM 4. *TSP* can be polynomially transformed to loading problem 1 for a threshold network with 3 layers.

Proof: See [12] for details.

For a given weight vector, the verification of whether the given architecture can perform the given task always takes polynomial time. Theorem 4 states that if *TSP* is *NP*-complete or *NP*-hard, so is the problem of loading a threshold net with 3 layers.

3.3 Layered Net and 4-Feasibility

Both Theorem 3 and Theorem 4 concern networks consisting of linear threshold units. In this section we consider layered nets where linear sum type of units are allowed. We will prove that loading a layered net is *NP*-complete.

In *BSS* model, the only *NP*-complete problem currently known is the ***4-Feasibility*** problem. A multivariable polynomial f of degree 4 is defined as follows:

$$f(x) = \sum_{i=1}^{n} c_i x_{i1} x_{i2} x_{i3} x_{i4}, \tag{3}$$

where $c_i, x_{ij} \in \mathcal{R}$, for $1 \leq i \leq n$, $1 \leq j \leq 4$. The ***4-Feasibility*** problem can be stated as: Is $f(x) = 0$ feasible over $\mathcal{R}$, i.e., does $f(x) = 0$ have a solution over $\mathcal{R}$?

Let's first consider the loading problem 1.

THEOREM 5. Loading a layered net consisting of linear sum and linear threshold units is *NP* complete.

Proof: Obviously, the loading problem in this case can be polynomially verified. So it is in *NP*. In the following we will focus on reducing the ***4-Feasibility*** to the loading problem.

The Figure 2 gives a clear presentation of the proof. The boxes denote the linear sum units whereas the circles denote the linear threshold units. All the threshold units are output nodes. All the unlabeled units are input nodes. In this case all the expected output values are 1s.

Each term in $f(x)$ corresponds to a column of linear sum units in the Figure 2. Each linear sum unit has a fixed weight 1 directly from the input layer. If there are some terms in $f(x)$ with degree less than 4, we can assume some $x_i = 1$. That means the corresponding units has fixed weight 1. Each column i has five inputs, $(i_0, ..., i_4)$. If each column has the input vector (1,0,0,0,0) in one instance and (-1,0,0,0,0) in another instance, $f(x) = 0$ will hold true.

There exist possibilities that $x_{ik} = x_{il}$ or $x_{kj} = x_{lj}$ Some additional threshold units are needed to guarantee this. In the Figure 2 the threshold unit R_{i23} can force $x_{i2} = r_{i23}$ and $x_{i3} = r_{i23}$ so that $x_{i2} = x_{i3}$ with inputs to column i are (0,1,0,0,0), (0,-1,0,0,0), (0,0,1,0,0) and (0,0,-1,0,0), respectively.

If the given layered network in the Figure 2 can be loaded, $f(x) = 0$ will be true, and ***vice versa***. □

If we advance the ***4-Feasibility*** problem to the problem of finding a solution to the $f(x) = 0$ if any, we will get a new problem, call it ***4-Finding***. Obviously, using the same transformation, we can polynomially reduce the ***4-Finding*** problem to loading problem 2. That means loading a layered network is at least as hard as ***4-Finding***. Currently, we do dot yet know relations between ***4-Feasibility*** and ***4-Finding***. Of course, ***4-Finding*** is at least as hard as ***4-Feasibility***.

4 Conclusions

Since the *BSS* model was proposed, there has been no real world problems studied in this model. The *4-Feasibility* problem has been the only one which is known to be *NP*-complete in the *BSS* model. In this work the first loading problem of a neural net consisting of linear sum units and linear threshold units is proved to be *NP*-complete. This result provides evidence that the definition of *NP*-completeness in the *BSS* model is meaningful for some real world problems.

The theorems in last section show that the loading problem is at least as hard as some computationally difficult problems. If the linear programming problem were proved in *NP*-complete, even training a single neuron would be *NP*-complete.

It is, however, hard to say that these theorems are telling bad news to connectionist community. Connectionist research is an engineering discipline where errors and noises would occur. The loading problem studied here is to find *exact* weights for a given task. In practical simulation, it is almost impossible to find *exact* solutions. Approximation should be allowed in many cases. The ***information-based complexity*** model which is developed by Traub and his colleagues (cf. [10]) seems very promising in this context.

Acknowledgements

The author would like to thank Prof. Dave Barrington and Prof. Andy Barto for many helpful discussions, and Dr. Art Gaylord for encouragement and support.

References

[1] Blum, L., "Lectures on a Theory of Computation and Complexity over the Reals (or an Arbitrary Ring)", *1989 Lectures in Complex Systems*, SFI Studies in the Sciences of Complexity, Addison Wesley, 1989.

[2] Blum, L., M. Shub, S. Smale, "On a Theory of Computation and Complexity over the Real Numbers: *NP*-Completeness, Recursive Functions and Universal Machines", *Bulletin of AMS*, Vol. 21, No. 1, July, 1989.

[3] Canny, J., "Some Algebraic and Geometric Computations in PSPACE", *Proceedings of the 20th Annual ACM Symposium on the Theory of Computing*, 1988.

[4] Duda, R.O. and P.E. Hart, *Pattern Classification and Scene Analysis*, John Wiley & Sons, 1973.

[5] Hong, JW, "On Connectionist Models", *Comm. on Pure and Applied Mathematics*, Vol. *XLI*, pp.1039-1050., 1988.

[6] Judd, J. S. *Neural Network Design and Complexity of Learning*, MIT Press, 1990.

[7] Ko, K., "Applying techniques of discrete complexity theory to numerical computation", ed. R. Book, *Studies in Complexity Theory*, 1985.

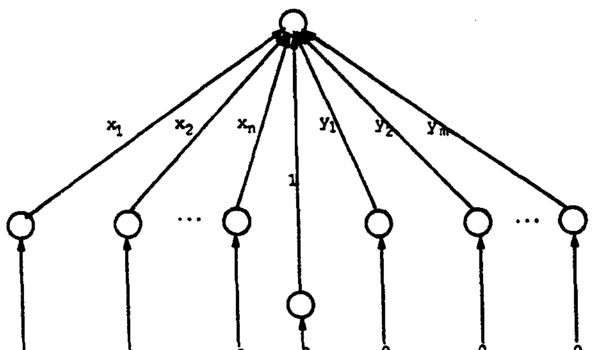
[8] Kohonen, T., "An Introduction to Neural Computing", *Neural Networks*, Vol. 1, pp.3-16, 1988.

[9] Rumelhart, D. and J. McClelland, *Parallel Distributed Processing*, Vol. 1, Cambridge, The MIT Press, 1986.

[10] Traub, J. F. and H. Wozoniakowski, "Information-Based Complexity: New Questions for Mathematicians", *The Mathematical Intelligencer*, Vol. 13, No. 2, 1991.

[11] Triesch, E., "A Note on a Theorem of Blum, Shub, and Smale", *Journal of Complexity*, Vol. 6, pp166-169, 1990.

[12] Zhang, X-D "Complexity of Neural Network Learning in the Real Number Model", Master's Thesis, Department of Computer Science, University of Massachusetts, Amherst, Jan. 1992.



$$
\begin{array}{llllllllll}
I_1 & = (1, & 0, & \cdots 0, & 0, & 0, & 0, & \cdots 0, & 1) \\
I_2 & = (0, & 1, & \cdots 0, & 0, & 0, & 0, & \cdots 0, & 1) \\
 & \cdots & & & & & & & \\
I_n & = (0, & 0, & \cdots 1, & 0, & 0, & 0, & \cdots 0, & 1) \\
I_{n+1} & = (0, & 0, & \cdots 0, & 0, & 1, & 0, & \cdots 0, & 1) \\
I_{n+2} & = (0, & 0, & \cdots 0, & 0, & 0, & 1, & \cdots 0, & 1) \\
 & \cdots & & & & & & & \\
I_{n+m} & = (0, & 0, & \cdots 0, & 0, & 0, & 0, & \cdots 1, & 1) \\
J_1 & = (-a_{11}, & -a_{12}, & -a_{1n}, & b_1, & 0, & 0, & \cdots 0, & 1) \\
J_2 & = (-a_{21}, & -a_{22}, & -a_{2n}, & b_2, & 0, & 0, & \cdots 0, & 1) \\
 & \cdots & & & & & & & \\
J_m & = (-a_{m1}, & -a_{m2}, & -a_{mn}, & b_m, & 0, & 0, & \cdots 0, & 1) \\
K_1 & = (0, & 0, & \cdots 0, & -c_1, & a_{11}, & a_{21}, & \cdots a_{m1}, & 1) \\
K_2 & = (0, & 0, & \cdots 0, & -c_2, & a_{12}, & a_{22}, & \cdots a_{m2}, & 1) \\
 & \cdots & & & & & & & \\
K_n & = (0, & 0, & \cdots 0, & -c_n, & a_{1n}, & a_{2n}, & \cdots a_{mn}, & 1) \\
L_1 & = (c_1, & c_2, & \cdots c_n, & 0, & -b_1, & -b_2, & \cdots -b_m, & 1)
\end{array}
$$

Figure 1. Loading a Single Neuron

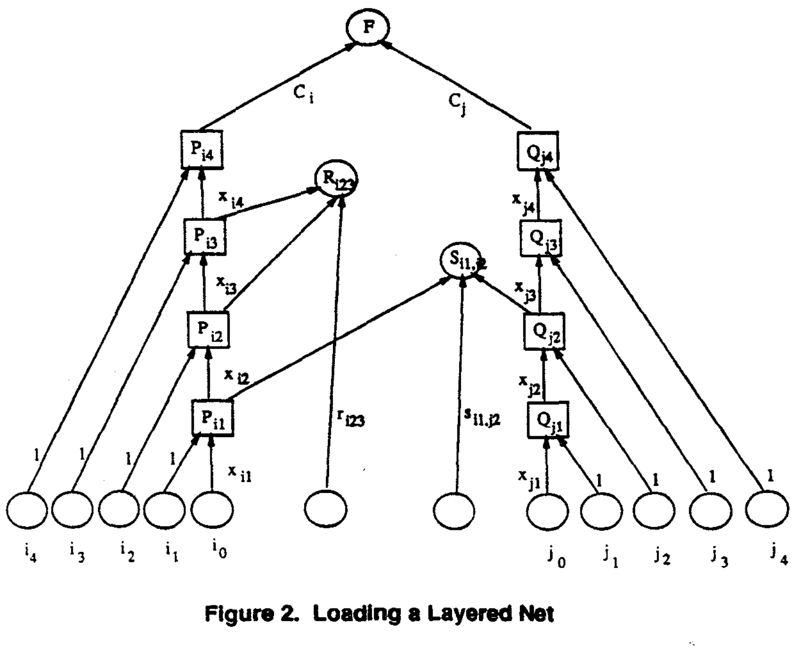


Figure 2. Loading a Layered Net