Complexity Results on Learning by Neural Nets[*]

Jyh-Han Lin and *Jeffrey Scott Vitter*

Department of Computer Science
Brown University
Providence, R. I. 02912–1910

Abstract

We consider the computational complexity of learning by neural nets. We are interested in how hard it is to design appropriate neural net architectures and to train neural nets for general and specialized learning tasks. Our main result shows that the training problem for 2-cascade neural nets (which have only two non-input nodes, one of which is hidden) is $\mathcal{NP}$-complete, which implies that finding an optimal net (in terms of the number of non-input units) that is consistent with a set of examples is also $\mathcal{NP}$-complete. This result also demonstrates a surprising gap between the computational complexities of one-node (perceptron) and two-node neural net training problems, since the perceptron training problem can be solved in polynomial time by linear programming techniques. We conjecture that training a k-cascade neural net, which is a classical threshold network training problem, is also $\mathcal{NP}$-complete, for each fixed $k \geq 2$. We also show that the problem of finding an optimal perceptron (in terms of the number of non-zero weights) consistent with a set of training examples is $\mathcal{NP}$-hard.

Our neural net learning model encapsulates the idea of modular neural nets, which is a popular approach to overcoming the scaling problem in training neural nets. We investigate how much easier the training problem becomes if the class of concepts to be learned is known *a priori* and the net architecture is allowed to be sufficiently non-optimal. Finally, we classify several neural net optimization problems within the polynomial-time hierarchy.

1 Introduction

Neural nets are often used to learn functions, in either a supervised or unsupervised mode. They are enticing because in some instances they are *self-programming*, in that they can

[*]Support was provided in part by an NSF Presidential Young Investigator Award CCR–8906419 with matching funds from IBM, by NSF research grant DCR–8403613, and by ONR grant N00014–83–K–0146, ARPA Order No. 6320, Amendment 1. An earlier and shortened version of this research appeared in *Proceedings of the 2nd Annual ACM Workshop on Computational Learning Theory*, Santa Cruz, CA, July–August 1989, published by Morgan Kaufman Publishers, San Mateo, CA. The authors can be reached by electronic mail at jhl@cs.brown.edu and jsv@cs.brown.edu, respectively. Correspondence can be addressed to the second author.

adjust their parameters by using general procedures based solely on examples of input-output pairs. In this paper we consider the computational complexity of learning by neural nets, building upon the work of Judd [1987, 1988], Blum and Rivest [1988], and Baum and Haussler [1989]. We are interested in how hard it is to design appropriate neural net architectures and to train neural nets for general and specialized learning tasks.

In the next section we introduce our neural net model and related definitions. Our main result in Section 3 extends the work of Judd [1987, 1988] and Blum and Rivest [1988] and further demonstrates the intractability of training nonmodular neural nets, as the problem dimension or size gets large. We refer to this phenomenon as the *scaling problem.* For Sections 4 and 5, we define a *modular* (or *hierarchical*) neural net model that encapsulates the idea of incremental design of large nets based on smaller subcomponent nets. Each subcomponent is trained separately and then fixed while higher-level subcomponents are trained (see, for example, [Weibel 1989], [Weibel and Hampshire 1989], and [Hinton 1989]). This modular approach can help alleviate the scaling problem. One of our goals in this paper is to determine to what extent the scaling problem is lessened.

We define the size of a neural net or net architecture to be the number of non-input nodes. Perceptrons, for example, have size 1. Most of our results are independent of this particular definition of size. However, when relevant we also consider other size measures for neural nets, such as the height, the number of edges, the number of nonzero weights, and the number of bits in the representation.

In Section 3 we present our main result that the training problem for a simple two-node completely unspecified net architecture with only one hidden unit, called a 2-cascade neural net, is $\mathcal{NP}$-complete. Since the perceptron (one-node neural net) training problem can be solved in polynomial time by linear programming techniques, this result demonstrates a surprising gap between the computational complexities of one-node and two-node neural net training problems. We conjecture that the training of k-cascade neural nets, which is a well-known threshold network training problem (see, for example, [Dertouzos 1965]), is also $\mathcal{NP}$-complete, for each fixed $k \geq 2$. We also show that the problem of finding an optimal perceptron (in terms of the number of non-zero weights) consistent with a set of training examples is $\mathcal{NP}$-hard.

In Section 4 we investigate how hard it is to train a modular neural net for a set of examples when the neural net is constrained to be in some architecture for learning a particular concept class. For the case of learning isothetic (that is, axis-parallel) rectangles, we show that it is easier to train a neural net that is sufficiently non-optimal in size, so that there is some "play" in setting its parameters. In the proces we introduce a general framework of Occam nets. In Section 5 we state several modular neural net optimization problems. In the appendix we show these problems to be $\mathcal{NP}$-complete or $\mathcal{NP}$-hard, and we classify them more precisely within the polynomial-time hierarchy.

2 The Neural Net Learning Model

In this paper, we restrict ourselves to feedforward neural nets of linear threshold elements. In particular, we are mainly concerned with neural nets for classification tasks. The inputs to the feedforward net will be from X^n, where X is either $\{0, 1\}$ or $\Re$. The nets produce one binary output.

Definition 1 A linear threshold unit $f_v = [\vec{w}; \theta]$ with input $\vec{x}$ is characterized by a weight vector $\vec{w}$ and a threshold θ:

$$f_v(\vec{x}) = \begin{cases} 1 & \text{if } \vec{w} \cdot \vec{x} \geq \theta; \\ 0 & \text{otherwise.} \end{cases}$$

For convenience, we identify the positive region defined by f_v as f_v and the negative region as $\overline{f_v}$.

One of the main issues in neural net design is the problem of *scaling*: Is it feasible, within limited resources and time, to build and train ever larger neural nets? By "train" we refer to determining the weights of the linear threshold elements. The results of Judd [1987] and Blum and Rivest [1988] and our results in Section 3 show for completely unspecified neural nets that scaling is intractable as the dimension and size gets large.

To overcome this problem of scaling, in the particular application of speech recognition, Weibel and Hampshire [1989] adopt the approach of modular and incremental design of large nets based on smaller subcomponent nets. The idea is to exploit the knowledge developed by smaller, independently trained nets by fixing and incorporating these smaller net modules into larger superstructures. It is hoped that this modular approach could not only reduce training time but also lead to a more incremental and distributed approach to the construction of large-scale neural nets. Modular neural nets are gaining popularity in a variety of applications; more information appears in [Weibel 1989], [Weibel and Hampshire 1989], and [Hinton 1989]. We encapsulate these ideas in the following *modular* (or *hierarchical*) neural net learning model, which we use in Sections 4 and 5:

Definition 2 A *modular (feedforward) neural net architecture* F is a directed acyclic graph G with n ordered designated input nodes and one output node. Nodes of G that are not input nor output nodes are called *hidden units*. Each non-input node v in G has *indegree*(v) inputs and is either associated with a linear threshold function f_v with *indegree*(v) inputs or is left undefined (denoted by $\perp$). A neural net f is a neural net architecture with no undefined nodes. We identify f with the function it represents and $Comp(F)$ with the set of functions computable by neural nets where each undefined node v in F is replaced by some linear threshold function f_v. The *complexities* or *sizes* of f and F, which we denote $|f|$ and $|F|$, are the numbers of non-input nodes in f and F, respectively.

Definition 3 *Training* a net architecture with a set of training examples consists of determining the weights of the undetermined linear threshold elements such that the function it computes is consistent with the training examples.

The classical perceptron is a neural net of size 1; it has no hidden units. Our definition allows us to "hardwire" parts of the net architecture, which we use to investigate the computational complexities of modular neural net design and training problems. (Note that we elect not to allow partially defined nodes.) In this paper we shall focus mainly on the definition of size specified above. Other possible size measures include height, number of edges, number of nonzero weights, and number of bits in the representation. Except where noted, our results are independent of the particular size measure used.

Let $D_n = 2^{X^n}$, we define a *concept class* $C_n \subseteq D_n$ to be a nonempty set of concepts. Each individual concept $c \in C_n$ is a subset of domain X^n. For each $c \in C_n$, we let *size*(c) denote the length of the encoding of c in some fixed encoding. We define $C_{n,s}$ to be the concept class of all concepts in C_n that have size at most s; hence, $C_n = \bigcup_{s \geq 1} C_{n,s}$. A *labeled example* for a concept c is a pair $(x, label)$, where $x \in X^n$ and *label* is "+" if $x \in c$ and "−" if $x \notin c$; we call $(x, +)$ a *positive example* and $(x, -)$ a *negative example.*

Definition 4 We call a neural net architecture F optimal if for all F' such that $Comp(F) \subseteq Comp(F')$, we have $|F| \leq |F'|$. We also call a neural net architecture F optimal for a concept class $C_{n,s}$ if $C_{n,s} \subseteq Comp(F)$ and for all F' such that $C_{n,s} \subseteq Comp(F')$ we have $|F| \leq |F'|$.

3 Cascade Neural Nets and Optimal Perceptrons

In this section we present our main result, concerning the difficulty of training k-cascade neural nets. We also investigate the computational complexity of optimizing the number of non-zero weights in a one-node neural net (that is, the well known perceptron) so that it is consistent with a set of training examples.

Judd [1987, 1988] shows that determining whether a neural net architecture can be trained for a set of training examples is $\mathcal{NP}$-complete. This is extended in [Blum and Rivest 1988] to a simple two-layer three-node architecture with two hidden units. In this section we extend their result further to only two nodes by showing that the training problem is also $\mathcal{NP}$-complete for a 2-cascade neural net (with only one hidden unit). The node functions are initially completely unspecified.

First we consider the training problem where the net architecture is allowed to be fully connected, and we want to minimize the number of non-input nodes. The following problem formalizes the problem at hand:

OPTIMAL CONSISTENT NET
Instance: A set S of training examples and a positive integer K.
Question: Is there a neural net f consistent with S such that $|f| \leq K$?

This problem is clearly in $\mathcal{NP}$. We show that this problem is $\mathcal{NP}$-complete by showing $\mathcal{NP}$-completeness for the particular case $K = 2$ (2-CASCADE NEURAL NET TRAINING), which we consider below.

A k-cascade neural net (see Figure 1), where $k \geq 2$, has $k - 1$ hidden units N_1, N_2, …, N_{k-1} and one output node N_k. All n inputs are boolean and are connected to nodes N_1, …, N_k. In addition, each N_i is connected to N_{i+1}; we designate the weight of this edge by g_i. Each node has $n + 1$ inputs except for N_1, which has only n inputs. We adopt the convention that g_i is the last weight to N_{i+1}. Cascade neural nets are more powerful and economical in terms of their size (the number of non-input nodes) than the class of layered neural nets considered in [Blum and Rivest 1988] (see [Dertouzos [1965]).

Let us consider the following problem, for any fixed $k \geq 2$:

k-CASCADE NEURAL NET TRAINING
Instance: A set $S = S^+ \cup S^-$ of training examples of n boolean inputs, where S^+ is the set of positive examples and S^- is the set of negative examples.
Question: Is there a k-cascade neural net f consistent with all training examples?

We shall show that this problem is $\mathcal{NP}$-complete for $k = 2$ by reducing the QUADRANT problem to it. The QUADRANT problem asks if the positive examples S^+ can be

Figure 1: Cascade neural net.

confined to a single quadrant, defined by the intersection of two halfspaces, with the negative examples S^- confined to the other three quadrants.

QUADRANT
Instance: A set $S = S^+ \cup S^-$ of training examples of n boolean inputs, where S^+ is the set of positive examples and S^- is the set of negative examples.
Question: Are there two halfspaces N_1 and N_2 such that $S^+ \subseteq N_1 \cap N_2$ and $S^- \subseteq \overline{N_1} \cup \overline{N_2}$?

Theorem 1 [Blum and Rivest 1988] *QUADRANT is $\mathcal{NP}$-complete.*

We use this to prove our main result:

Theorem 2 *2-CASCADE NEURAL NET TRAINING is $\mathcal{NP}$-complete.*

Proof: Training a 2-cascade neural net is clearly in $\mathcal{NP}$. To prove $\mathcal{NP}$-hardness, we reduce QUADRANT to it. Given a set of training examples $S = S^+ \cup S^-$ for QUADRANT, we add two new dimensions and create the following set of augmented examples $T = T^+ \cup T^-$ for training a 2-cascade neural net:

$$\begin{aligned} T^+ &= \{\vec{x}00 \mid \vec{x} \in S^+\} \cup \{\vec{x}11 \mid \vec{x} \in S\}, \\ T^- &= \{\vec{x}00 \mid \vec{x} \in S^-\} \cup \{\vec{x}01, \vec{x}10 \mid \vec{x} \in S\}. \end{aligned}$$

This is illustrated pictorially in Figure 2. The points $\vec{x}00$ in the n-dimensional hypercube on the first n dimensions retain their former sign.

The positive region induced by a 2-cascade net is bordered by a "zig-zag" of hyperplanes, in which the two outer (semi-infinite) hyperplanes are parallel. The basic idea of the proof is that the extra two dimensions of the examples in T force one of the semi-infinite hyperplanes to "miss" the n-dimensional hypercube, so that there is a 2-cascade neural net f consistent with T if and only if there is a quadrant solution to S.

Figure 2: Examples used to show that 2-CASCADE NEURAL NET TRAINING is $\mathcal{NP}$-complete.

($\Longrightarrow$) Suppose the quadrant solution to S is

$$[\vec{a};\theta_1]\wedge[\vec{b};\theta_2].$$

We construct a 2-cascade neural net f consistent with T as follows:

$$\begin{aligned} N_1 &= [\vec{a},-A-|\theta_1|,-A-|\theta_1|;\theta_1], \\ N_2 &= [\vec{b},2B,2B,3B-\theta_2;3B], \end{aligned}$$

where $A>\sum_{i=1}^{n}|a_i|$ and $B>\sum_{i=1}^{n}|b_i|$. It is interesting to note that there is also a quadrant solution to T:

$$[\vec{a},2A+2|\theta_1|,-A-|\theta_1|;\theta_1]\ \wedge\ [\vec{b},-B-|\theta_2|,2B+2|\theta_2|;\theta_2].$$

This is surprising on first glance, given the second-half of the proof, immediately below.

($\Longleftarrow$) Suppose the 2-cascade neural net f consistent with T is as follows:

$$\begin{aligned} N_1 &= [\vec{a},A_1,A_2;\theta_1], \\ N_2 &= [\vec{b},B_1,B_2,g_1;\theta_2]. \end{aligned}$$

In the following, let

$$\begin{aligned} {N_2}^{N_1=0} &= [\vec{b},B_1,B_2;\theta_2], \\ {N_2}^{N_1=1} &= [\vec{b},B_1,B_2;\theta_2-g_1]. \end{aligned}$$

Case 1. Suppose $g_1 \geq 0$. This implies that all examples in ${N_2}^{N_1=0}$ are positive examples and all examples not in ${N_2}^{N_1=1}$ are negative examples. Also note that all positive examples belong either to ${N_2}^{N_1=0}$ or to $N_1 \cap {N_2}^{N_1=1}$. If for all $\vec{x} \in S^+$ we have $\vec{x}00 \in {N_2}^{N_1=0}$, then clearly S is linearly separable and has a trivial quadrant solution:

$$[\vec{b}; \theta_2] \wedge [\vec{b}; \theta_2].$$

Otherwise, we claim for all $\vec{x} \in S^+$ that

$$\vec{x}00 \in N_1 \cap {N_2}^{N_1=1}.$$

Suppose there exists some $\vec{y} \in S^+$ such that $\vec{y}00 \in {N_2}^{N_1=0}$. There exists at least one example $\vec{x} \in S^+$ such that $\vec{x}00 \in N_1 \cap {N_2}^{N_1=1}$ but $\vec{x}00 \notin {N_2}^{N_1=0}$. Since $\vec{y}00 \in {N_2}^{N_1=0}$ and since $\vec{y}01$ and $\vec{y}10$ are negative examples, we must have $B_1, B_2 < 0$ and $\vec{y}11 \notin {N_2}^{N_1=0}$. But $\vec{y}11$ is a positive example, and so $\vec{y}11 \in N_1 \cap {N_2}^{N_1=1}$. From $B_1, B_2 < 0$, we have $\vec{y}01, \vec{y}10 \in {N_2}^{N_1=1}$. Since $\vec{y}01$ and $\vec{y}10$ are negative examples, it follows that $\vec{y}01, \vec{y}10 \notin N_1$ and $A_1, A_2 > 0$.

From the fact that $\vec{x}00 \notin {N_2}^{N_1=0}$ and $B_1, B_2 < 0$, we have $\vec{x}11 \notin {N_2}^{N_1=0}$. Since $\vec{x}11$ is a positive example, we must have $\vec{x}11 \in N_1 \cap {N_2}^{N_1=1}$. From $B_1, B_2 < 0$, we know that $\vec{x}01, \vec{x}10 \in {N_2}^{N_1=1}$. Since $\vec{x}00 \in N_1 \cap {N_2}^{N_1=1}$ and $A_1, A_2 > 0$, it follows that $\vec{x}01, \vec{x}10 \in N_1$. Thus, we have to conclude that $\vec{x}01, \vec{x}10 \in N_1 \cap {N_2}^{N_1=1}$. But this implies that $\vec{x}01$ and $\vec{x}10$ are positive examples, a contradiction to our supposition that there exists some $\vec{y} \in S^+$ such that $\vec{y}00 \in {N_2}^{N_1=0}$! This proves our claim and shows that the quadrant solution to S is

$$[\vec{a}; \theta_1] \wedge [\vec{b}; \theta_2 - g_1].$$

Case 2. The case when $g_1 < 0$ can be proved similarly, except the trivial quadrant solution is

$$[\vec{b}; \theta_2 - g_1] \wedge [\vec{b}; \theta_2 - g_1].$$

Otherwise we claim for all $\vec{x} \in S^+$ that

$$\vec{x}00 \in \overline{N_1} \cap {N_2}^{N_1=0}.$$

This shows that the quadrant solution to S is

$$[-\vec{a}; -\theta_1 + \eta] \wedge [\vec{b}; \theta_2],$$

where $0 < \eta \leq \min_{\vec{x} \in S^+}\{\theta_1 - \vec{a}\vec{x}\}$. □

Corollary 1 *The OPTIMAL CONSISTENT NET problem is $\mathcal{NP}$-complete, even if the inputs are boolean.*

Proof : This result holds, even for the special case of $K = 2$, as a consequence of Theorem 2. □

Blum and Rivest [1988] have shown that the problem of whether S^+ can be isolated by two parallel planes is also $\mathcal{NP}$-complete. Since our proof can be modified to cover this restricted case, we have also proved the following theorem:

Theorem 3 *It is $\mathcal{NP}$-complete to decide whether there is a restricted 2-cascade neural net f, in which the weight vector $\vec{b}$ of N_2 is the negative of the weight vector $\vec{a}$ of N_1, that is consistent with a set of training examples.*

Proof: The key modification needed is as follows: Suppose the quadrant solution to S is

$$[\vec{a};\theta_1] \wedge [-\vec{a};\theta_2].$$

Since the inputs are binary vectors, we may assume without loss of generality that $|\theta_1|, |\theta_2| \leq \sum_{i=1}^{n} |a_i|$. It is easy to see that the following restricted 2-cascade neural net is consistent with T:

$$\begin{aligned} N_1 &= [\vec{a}, -2A, -2A; \theta_1], \\ N_2 &= [-\vec{a}, 2A, 2A, 3A - \theta_2; 3A], \end{aligned}$$

where $A > \sum_{i=1}^{n} |a_i|$. □

We are hopeful that our reduction for 2-cascade neural nets can be extended to handle k-cascade neural nets, for each fixed $k \geq 3$, by adding new dimensions and creating an augmented training set in a similar manner. We make the following conjecture:

Conjecture 1 *Training a k-cascade neural net is $\mathcal{NP}$-complete for each fixed $k \geq 2$.*

Theorem 2 shows that the OPTIMUM CONSISTENT NET problem is $\mathcal{NP}$-complete, where the size of a net is defined to be the number of non-input nodes. We can show that it remains $\mathcal{NP}$-complete for the case of perceptrons when the size measure is the number of non-zero weights:

OPTIMAL CONSISTENT PERCEPTRON
Instance: A set S of boolean training examples.
Problem: Construct a perceptron $f = [\vec{w};\theta]$ such that f is consistent with S and the number of non-zero components in $\vec{w}$ is minimized.

Theorem 4 *The OPTIMAL CONSISTENT PERCEPTRON problem is $\mathcal{NP}$-hard.*

Proof: Haussler [1988] has shown the problem of finding the optimal monotone monomial consistent with a set of training examples is $\mathcal{NP}$-complete, which by duality implies that the problem of finding the optimal monotone pure disjunction is also $\mathcal{NP}$-complete. We shall abbreviate this latter problem as the OPTIMAL MONOTONE PURE DISJUNCTION problem and reduce it to the OPTIMAL CONSISTENT PERCEPTRON problem via a Turing reduction.

Let $\{v_1, v_2, \ldots, v_n\}$ be the set of n boolean variables and let S be the training set. We want to know if there exists a monotone pure disjunction with at most K unnegated variables that is consistent with S. First we check if there is any monotone pure disjunction consistent with S, regardless of its size. This can be easily done in polynomial time with the standard consistency algorithm (see, for example, [Vitter and Lin 1988]). If the answer is "No," we return "No." Otherwise, let *OptP* be the searching algorithm for the OPTIMAL CONSISTENT PERCEPTRON problem, which takes a set S' of training examples as input and outputs an optimal perceptron f consistent with S'. We run the following iterative decision algorithm *OptMPD*, which calls *OptP* as subroutine, to determine whether $(S, K) \in$ OPTIMAL MONOTONE PURE DISJUNCTION:

```
Algorithm OptMPD
Input: A set $S$ of training examples and a positive integer $K$.
Output: "Yes" or "No."
begin
    $S^* \leftarrow S \cup \{(\vec{0}, -), (\vec{0}_{k_1,\ldots,k_p}, -)\}$;
    $S' \leftarrow S^*$;
    $[\vec{w}; \theta] \leftarrow OptP(S')$;
    while $\vec{w}$ contains any negative component do
        begin
            $S' \leftarrow MarkOff(S', \vec{w})$;
            $[\vec{w}; \theta] \leftarrow OptP(S')$
        end;
    if $NonZero(\vec{w}) \leq K$
        then return "Yes";
        else return "No"
end.
```

In the above algorithm $\vec{0}_{k_1,\ldots,k_p}$ denotes the example all of whose components are 0, except those at indices $k_1, \ldots, k_p$, which are 1. Subroutine *MarkOff* takes a set of examples S' and a weight vector $\vec{w}$ as input and returns a new set of examples by zeroing out the components of each example corresponding to negative components in $\vec{w}$. Given weight vector $\vec{w}$, subroutine *NonZero* counts the number of non-zero weights in $\vec{w}$. We define $I = \{k_1, \ldots, k_p\}$ to be the maximal set of indices such that for each $k_q \in I$ there exists a negative example $\vec{y}$ with $\vec{y}_{k_q} = 1$. We call a perceptron *positive* if all its weights and threshold are nonnegative. Algorithm *OptMPD* forces *OptP* to output an optimal positive perceptron by

1. Including $\vec{0}$ as a negative example to force $\theta > 0$. This can be done since the concept class to be learned is monotone pure disjunctions.

2. Including the binary n-vector $\vec{0}_{k_1,\ldots,k_p}$, as a negative example. This can be done since any monotone pure disjunction consistent with S has to classify $\vec{0}_{k_1,\ldots,k_p}$ as a negative example. The reason for this inclusion will be clear below.

3. Iteratively zeroing out the components of examples corresponding to negative weight components. This procedure preserves consistency, in that at the end of each iteration the monotone pure disjunctions consistent with S remain consistent with the set of new examples, and *vice versa*. This follows because those example components corresponding to negative weight components are useful only for the identification of negative examples and cannot be included in any monotone pure disjunctions that are consistent with S.

We claim that *OptMPD* return "Yes" if and only if there exists a monotone pure disjunction consistent with S^* (and, therefore, consistent with S) with at most K unnegated variables. To see that this is true, we need the following lemma:

Lemma 1 *Let $f = [\vec{w}, \theta]$ be any optimal positive perceptron consistent with S^*. By optimal we mean that the number of nonzero components in $\vec{w}$ is minimum. Then for each $\vec{w}_j > 0$, we have $j \notin I$.*

Proof: (By contradiction.) Let J be the set of indices of non-zero weight components. Since we include $\vec{0}_{k_1,\ldots,k_p}$ as a negative example, for each positive example $\vec{x}$ there must exist $j \in J - I$ such that $\vec{x}_j = 1$. Thus, we may construct another perceptron $f' = [\vec{z}, \theta]$ consistent with S^* as follows: Let $W = \sum_{i \in I} \vec{w}_i$. For all $i \in J - I$, let $\vec{z}_i = \vec{w}_i + W$; all other components of $\vec{z}$ are 0s. Therefore, f is not optimal. Contradiction. $\square$

Continuation of the Proof of Theorem 4. Suppose that $v_{i_1} + \ldots + v_{i_\ell}$ is a monotone pure disjunction consistent with S, where $\ell \leq K$. Then $[\vec{0}_{i_1,\ldots,i_\ell}; 1]$ is a consistent positive perceptron. For the other direction, suppose that $f = [\vec{w}; \theta]$ is an optimal positive perceptron consistent with S with exactly $k \leq K$ non-zero weights, and let the set of indices of non-zero weights be $J = \{j_1, \ldots, j_k\}$. It is clear that $g = v_{j_1} + \ldots + v_{j_k}$ is an optimal monotone pure disjunction consistent with S: First, g has to include all positive examples since $\theta > 0$. Secondly, g also excludes all negative examples by Lemma 1. Finally, g has to be optimal; otherwise, f is not optimal, either. This proves our claim.

Finally, note that *OptMPD* runs in polynomial time if *OptP* is a polynomial-time searching algorithm. Therefore, this is a polynomial-time reduction and the OPTIMAL CONSISTENT PERCEPTRON problem is $\mathcal{NP}$-hard. $\square$

The results in this section show that the training problem is inherently difficult even for simple 2-node neural nets. Furthermore, the training problem for perceptrons is also computationally infeasible if the number of non-zero weights is to be minimized. In the next section we investigate in a theoretical way possible restrictions for making the training problem tractable.

4 Neural Nets and the VC Dimension

In typical real-world neural net design problems, we start with a set of training examples, choose (or guess) an appropriate net architecture, and then use some procedure (such as back propagation) to train the neural net (that is, to set the parameters of the net so that we can correctly classify as many examples as possible). It is shown in [Baum and Haussler 1989] that if a large enough fraction of enough random examples (drawn independently from an unknown distribution) can be loaded onto the neural net, then the net will "generalize" in Valiant's sense [Valiant 1984] and probably answer future queries with low error. (By "loaded," we mean that the example is correctly classified by the fully specified neural net.) This learning framework is known as the *probably approximately correct* (or *PAC*) learning model. In the following we adopt the PAC-learning model of Valiant [1984] and Blumer et al [1989] and investigate how the complexity of modular training is affected by restricting the problem's domain to learning a specific concept class.

A central concept of PAC-learning framework is the *Vapnik-Chervonenkis dimension* (VC dimension) of concept classes. Intuitively, the VC dimension is a combinatorial measure of the expressive power (or richness) of a concept class.

Definition 5 Let $C_{n,s} \subseteq D_n$ be a concept class. Given a set of nonlabeled examples $S \subseteq X^n$, we denote by $\Pi_{C_{n,s}}(S)$ the set of all subsets $P \subseteq S$ such that there is some concept $c \in C_{n,s}$ for which $P \subseteq c$ and $(S - P) \subseteq \overline{c}$. If $\Pi_{C_{n,s}}(S) = 2^S$, we say that *S is shattered by $C_{n,s}$*. The *Vapnik-Chervonenkis dimension* (VC dimension) of $C_{n,s}$ is the cardinality of the largest finite set of examples that is shattered by $C_{n,s}$; it is infinite if arbitrarily large sets can be shattered.

We use log to denote the logarithm base 2 and ln to denote the natural logarithm. The following corollary from [Baum and Haussler 1989] bounds the VC dimension of a net architecture:

Corollary 2 *Let F be a net architecture with $s \geq 2$ non-input nodes and E edges, then*

$$VCdim(F) \leq 2(E+s)\log(es),$$

where e is the base of natural logarithm.

Let $\mathcal{F}_s$ be a net architecture with s non-input nodes and with all possible edges; that is, the s non-input nodes are numbered from 1 to s, and each noninput node has inputs from the n input nodes and from all previous noninput nodes. Clearly, $Comp(\mathcal{F}_s) = \bigcup_{|f| \leq s}\{f\}$. The following lemma bounds the VC dimension of $\mathcal{F}_s$.

Lemma 2 *The VC dimension of $\mathcal{F}_s$ can be bounded as follows:*

1. $VCdim(\mathcal{F}_0) \leq \log n$,
2. $VCdim(\mathcal{F}_1) = n+1$,
3. $VCdim(\mathcal{F}_s) \leq s(2n+s+1)\log(es)$, *for all $s \geq 2$.*

Proof: Bounds 1 and 2 are straightforward. For bound 3, note that the number of edges in $\mathcal{F}_s$ is $ns + s(s-1)/2$. The proof then follows directly from Corollary 2. □

The next lemma gives a general lower bound on the size of a net architecture that contains some concept class:

Lemma 3 *Let $C_{n,s}$ be a concept class, where $VCdim(C_{n,s}) \geq 2n$, and let F be a net architecture such that $C_{n,s} \subseteq Comp(F)$. We have*

$$|F| = \Omega\left(\sqrt{\frac{VCdim(C_{n,s})}{n} \Big/ \log\left(\frac{VCdim(C_{n,s})}{n}\right)}\right).$$

Proof: The proof follows simply from Lemma 2 and the fact that $VCdim(\mathcal{F}_{|F|}) \geq VCdim(F) \geq VCdim(C_{n,s})$. □

It is not surprising that training is hard without any domain knowledge. In the following we investigate how much easier the training problem becomes when the net architecture is constrained for learning a particular concept class. In the problem statements of this section, $C_{n,s}$ is an implicitly known concept class (such as the union of s isothetic (that is, axis-parallel) rectangles or symmetric boolean functions) and is not a part of the input.

NET ARCHITECTURE TRAINING
Instance: A set S of training examples for a concept from $C_{n,s}$ and a modular neural net architecture F for $C_{n,s}$ (that is, $C_{n,s} \subseteq Comp(F)$).

Figure 3: Net architecture for the unions of s isothetic rectangles.

Question: Is there some $f \in Comp(F)$ such that f is consistent with S?

One of the concept classes with wide application in both artificial intelligence and database is the class of the unions of isothetic rectangles (see, for example, [Haussler 1988]). We show the following:

Theorem 5 *The NET ARCHITECTURE TRAINING problem is $\mathcal{NP}$-complete if the concept class $C_{n,s} = R_s$ is the set of unions of s isothetic rectangles.*

Proof: It is well known that it is $\mathcal{NP}$-hard to decide if the minimum number of isothetic rectangles needed to cover all positive training examples in the plane is less than or equal to s (see [Masek 1978]). To solve this problem, we construct a modular three-layer net architecture F as shown in Figure 3. The output node is hardwired to be the OR of the s second-layer hidden units, which are all ANDs. Each AND has inputs from 4 hidden units under it. Among these four hidden units, two have single inputs from x and the other two have single inputs from y. There exists a neural net $f \in Comp(F)$ consistent with all training examples if and only if the minimum number of isothetic rectangles needed is less than or equal to s. □

This theorem also gives a result similar to that in [Judd 1987] for our modular model of net architecture. The reason why this problem is difficult is that some net architectures are harder to train than others. In practice, neural net researchers often design their nets and net architectures to be slightly nonoptimal so as to allow some "play" in constructing the weights during the training. In some cases, this approach makes the training problem tractable. This approach motivates the following notion of Occam nets:

Definition 6 Let F^{opt} be an optimal net architecture for $C_{n,s}$. An (α, j, k)-*Occam net finder* A for $C_{n,s}$, where $0 \leq \alpha < 1$ and $j, k \geq 0$, is a polynomial-time algorithm that maps each set of training examples S to some consistent *Occam net* $f \in Comp(H_{n,s,|S|})$, where $H_{n,s,|S|}$ is a net architecture, such that $VCdim(H_{n,s,|S|}) \leq |S|^{\alpha} n^{j} |F^{\mathrm{opt}}|^{k}$.

By modifying the analysis of [Blumer et al 1989] we obtain the following theorem:

Theorem 6 *If there is an (α, j, k)-Occam net finder A for $C_{n,s}$, where $0 \leq \alpha < 1$ and $j, k \geq 0$, and if the number S of random examples satisfies*

$$|S| \geq \max\left\{ \frac{4}{\epsilon} \log \frac{2}{\delta}, \left(\frac{8n^j |F^{\rm opt}|^k}{\epsilon} \log \frac{13}{\epsilon} \right)^{1/(1-\alpha)} \right\},$$

where $F^{\rm opt}$ is an optimal net architecture for $C_{n,s}$, then A is a PAC-learning algorithm and the neural net f is its output hypothesis. That is, with probability at least $1-\delta$, the neural net f will predict correctly at least a fraction $1-\epsilon$ of future random examples drawn from the same distribution.

Proof: The proof is a simple application of Theorem 3.2.1 in [Blumer et al 89]. □

The following lemma allows us to bound the VC dimension of an Occam net architecture in terms of the size measure of a concept class $C_{n,s}$ instead of the size of the optimal net architecture containing $C_{n,s}$:

Lemma 4 *If $|F^{\rm opt}| = \Omega(s^\beta)$ for some $\beta > 0$, then the upper bound on $VCdim(H_{n,s,|S|})$ in Definition 6 can be replaced by $|S|^\alpha n^j s^k$, and $|F^{\rm opt}|^k$ in Theorem 6 can be replaced by s^k.*

The next theorem shows an example of Occam net finders:

Theorem 7 *There is an (α, j, k)-Occam net finder for the the concept class $C_{n,s} = R_s$ of the set of unions of s isothetic rectangles.*

Proof : There is a well-known simple greedy algorithm for R_s, which is optimal within a relative factor of $\ln |S| + 1$ (see, for example, [Blumer et al 89]). The output of the greedy algorithm can be easily transformed into a neural net $f \in Comp(H_{s,|S|})$, where $H_{s,|S|}$ is a net architecture of size $O(s \log |S|)$ and with $O(s \log |S|)$ edges. From Corollary 2 we have

$$VCdim(H_{s,|S|}) = O((s \log |S|)(\log s + \log\log |S|)).$$

Clearly, $VCdim(R_s) = \Omega(s)$. From Lemma 3, we have $|F^{\rm opt}| = \Omega(\sqrt{s/\log s})$. Thus, from Lemma 4 there is an (α, j, k)-Occam net finder for R_s. □

By Theorem 3.2.4 in [Blumer et al 1989], we may generalize Theorem 7 and prove the following:

Theorem 8 *Let C be a concept class with finite VC dimension d, let $C_s = \{\bigcup_{i=1}^s c_i \mid c_i \in C, 1 \leq i \leq s\}$. If there exists a polynomial-time net finder for C, then there also exists an (α, j, k)-Occam net finder for C_s.*

Proof: Since the VC dimension of C is finite, we may assume that the size of neural nets returned by the polynomial-time net finder is also finite. The union operation can be implemented with a single threshold element. The rest of the proof follows immediately from Theorem 3.2.4 in [Blumer et al 1989]. □

The results in this section suggest that it is sometimes easier to train non-optimal neural nets than optimal ones. This observation agrees with experimental results reported in [Rumelhart et al 1986] that the training time can usually be reduced by increasing the number of hidden units. (In [Rumelhart et al 1986] hidden units compute differentiable functions; in this paper we consider threshold functions.)

5 Neural Net Optimization Problems

We show in this section the infeasibility of comparing the power of different modular neural net architectures or even just answering whether the function performed by one neural net can be realized by another modular neural net architecture. These results are interesting for the following reasons:

1. Learning is impossible unless the function to be learned is realizable by the net architecture. This imposes a lower bound on the size of a net architecture.
2. But as the size of the net architecture gets larger, the training problem gets more complex. The resulting computational constraints put an upper bound on architecture size.

We formalize the related problems as follows. The first problem ask if the given neural net outputs anything other than 0.

NON-ZERO NET
Instance: A neural net f.
Question: Is $f \neq 0$?

The next problem asks if two given nets differ on some input?

NET INEQUIVALENCE
Instance: Neural nets f_1 and f_2.
Question: Is $f_1 \neq f_2$?

OPTIMAL EQUIVALENT NET
Instance: Neural net f and positive integer K.
Question: Is there a neural net f' such that $f' = f$ and $|f'| \leq K$?

The next problem deals with determining if a neural net is optimal.

NET MEMBERSHIP
Instance: Neural net f and neural net architecture F.
Question: Is $f \in \mathit{Comp}(F)$.

The next problem asks if a given net architecture realizes some function that the other does not?

NET ARCHITECTURE NONCONTAINMENT
Instance: Neural net architectures F_1 and F_2.
Question: Is $\mathit{Comp}(F_1) \not\subseteq \mathit{Comp}(F_2)$?

The next problem asks if two given net architectures are not equivalent.

NET ARCHITECTURE INEQUIVALENCE
Instance: Neural net architectures F_1 and F_2.
Question: Is $\mathit{Comp}(F_1) \neq \mathit{Comp}(F_2)$?

The next problem deals with determining if a given net architecture is optimal.

OPTIMAL NET ARCHITECTURE
Instance: Neural net architecture F and positive integer K.
Question: Is there a neural net architecture F' such that $Comp(F') \supseteq Comp(F)$ and $|F'| \leq K$?

In the appendix we show that the above problems are all $\mathcal{NP}$-complete or $\mathcal{NP}$-hard, and we classify their computational complexities more precisely within the polynomial-time hierarchy.

6 Conclusions

Neural nets offer the potential of learning a wide variety of concepts in a simple, uniform way. To fully evaluate their potential, we must determine how difficult it is to construct a neural net that learns a particular class of concepts as a function of the concept complexity, the size of the net architecture, and so on. Our results indicate that, without any domain-specific knowledge, the training problem is in general infeasible, even for concepts representable by a very simple 2-node neural net with only one hidden unit. On the other hand, if the concept class to be learned is known *a priori* and the net architecture is appropriately sized and properly interconnected, sometimes the training problem can be much easier (perhaps by a specialized learning algorithm).

Back propagation [Rumelhart et al 1986] [Hinton 1989] is a method for self-programming neural nets with differentiable node functions. Experiments by Rumelhart et al [1986] show that back propagation works better given non-optimal rather than optimal net architectures. It would be interesting to extend our model and show this property theoretically.

Acknowledgments. We thank the referees for several helpful comments and suggestions.

Appendix

The problems defined in Section 5 are all $\mathcal{NP}$-hard. An interesting theoretical goal is to classify these $\mathcal{NP}$-hard problems in the polynomial-time hierarchy [Stockmeyer 1977] [Garey and Johnson 1979]:

$$\Sigma_0^p = \Pi_0^p = \Delta_0^p = \mathcal{P};$$

and for $k \geq 0$,

$$\begin{aligned} \Sigma_{k+1}^p &= \mathcal{NP}(\Sigma_k^p), \\ \Pi_{k+1}^p &= \text{co-}\mathcal{NP}(\Sigma_k^p), \\ \Delta_{k+1}^p &= \mathcal{P}(\Sigma_k^p). \end{aligned}$$

The class $P(A)$ consists of all problems that can be solved in P with an oracle for A. Problems at each level of the hierarchy are at least as hard as (and are generally believed to be harder than) those at the preceding level. A natural complete set for Σ_k^p is the set B_k of true boolean formulas with k alternating quantifiers.

The computational complexities of the problems in Section 5 are summarized in the following theorem:

Theorem 9 *The problems defined in Section 5 can be classified as follows:*

1. *The NON-ZERO NET problem is $\mathcal{NP}$-complete.*
2. *The NET INEQUIVALENCE problem is $\mathcal{NP}$-complete.*
3. *The OPTIMAL EQUIVALENT NET problem is in Σ_2^p and is $\mathcal{NP}$-hard.*
4. *The NET MEMBERSHIP problem is Σ_2^p-complete.*
5. *NET ARCHITECTURE NONCONTAINMENT is in Σ_3^p and is Σ_2^p-hard.*
6. *The NET ARCHITECTURE INEQUIVALENCE problem is in Σ_3^p and is Σ_2^p-hard.*
7. *The OPTIMAL NET ARCHITECTURE problem is in Π_3^p and is $\mathcal{NP}$-hard.*

We shall use the following theorem from [Stockmeyer and Meyer 1973] and [Wrathall 1977] to establish the upper bounds for the Theorem 9:

Theorem 10 *Let $L \subseteq ,^*$ be a language. For any $k \geq 1$, $L \in \Sigma_k^p$ if and only if there exist polynomials $p_1, \ldots, p_k$ and a polynomial time recognizable relation R of dimension $k+1$ over $,^*$ such that for all $x \in ,^*$ we have $x \in L$ if and only if*

$$(\exists y_1)(\forall y_2)\ldots(\mathrm{Q}y_k)\left[\langle x, y_1, \ldots, y_k\rangle \in R\right],$$

where $|y_i| \leq p_i(|x|)$ and Q is "$\exists$" if k is odd and "$\forall$" if k is even. Dually, for any $k \geq 1$, $L \in \Pi_k^p$ if and only if we have $x \in L$ if and only if

$$(\forall y_1)(\exists y_2)\ldots(\mathrm{Q}y_k)\left[\langle x, y_1, \ldots, y_k\rangle \in R\right],$$

where $|y_i| \leq p_i(|x|)$ and Q is "$\forall$" if k is odd and "$\exists$" if k is even.

Figure 4: Net architecture for the NET MEMBERSHIP problem.

Proof of Theorem 9:

1. The NON-ZERO NET problem is clearly in $\mathcal{NP}$. To prove completeness, we reduce SATISFIABILITY to this problem. Given a boolean formula ϕ, we construct a neural net f_ϕ simulating ϕ. Clearly, ϕ is satisfiable if and only if f_ϕ is a non-zero net.

2. The NET INEQUIVALENCE problem is $\mathcal{NP}$-complete since it contains the NON-ZERO NET problem as a special case.

3. The upper bound for the OPTIMAL EQUIVALENT NET problem follows from the fact that $(f, K) \in$ OPTIMAL EQUIVALENT NET if and only if

$$(\exists f')(\forall \vec{x})\,[|f'| \leq K \text{ and } f'(\vec{x}) = f(\vec{x})]\,.$$

   The $\mathcal{NP}$-hardness is obtained by reducing NON-ZERO NET to this problem. Give an instance f of NON-ZERO NET, we construct a neural net $z \vee f$, where z is a new variable. Now $f \neq 0$ if and only if

$$(z \vee f, 0) \notin \text{OPTIMAL EQUIVALENT NET.}$$

4. The upper bound for the NET MEMBERSHIP problem follows from the fact that $f \in \mathit{Comp}(F)$ if and only if

$$(\exists f' \in \mathit{Comp}(F))(\forall \vec{x})\,[f'(\vec{x}) = f(\vec{x})]\,.$$

Figure 5: Neural net architecture for $F_1 \cup F_2$. Note that the constant 1 can be implemented as $x + \overline{x}$.

To establish the lower bound, we reduce B_2 QBF SATISFIABILITY to this problem. Given an instance of B_2 formula $(\exists \vec{x})(\forall \vec{y}) B(\vec{x}, \vec{y})$, we construct a net architecture F_B as shown in Figure 4. Now the given B_2 QBF formula is satisfiable if and only if $1 \in \mathit{Comp}(F_B)$. This result does not depend on the particular size measure used.

5. The upper bound for the NET ARCHITECTURE NONCONTAINMENT problem follows from the fact that $\mathit{Comp}(F_1) \not\subseteq \mathit{Comp}(F_2)$ if and only if

$$(\exists f_1 \in \mathit{Comp}(F_1))(\forall f_2 \in \mathit{Comp}(F_2))(\exists \vec{x}) \left[f_1(\vec{x}) \neq f_2(\vec{x}) \right].$$

We reduce the NET MEMBERSHIP problem, which is Σ_2^p-complete, to this problem. This is easy to see since $f \in F$ if and only if it is not the case that $f \not\subseteq F$.

6. The upper bound for the NET ARCHITECTURE INEQUIVALENCE problem follows from the fact that $\mathit{Comp}(F_1) \neq \mathit{Comp}(F_2)$ if and only if

$$\mathit{Comp}(F_1) \not\subseteq \mathit{Comp}(F_2) \text{ or } \mathit{Comp}(F_2) \not\subseteq \mathit{Comp}(F_1).$$

We reduce the NET ARCHITECTURE NONCONTAINMENT problem, which is Σ_2^p-hard, to this problem. We can construct a net architecture that computes exactly $\mathit{Comp}(F_1) \cup \mathit{Comp}(F_2)$ by the construction illustrated in Figure 5. The proof follows from the fact that $\mathit{Comp}(F_1) \not\subseteq \mathit{Comp}(F_2)$ if and only if $\mathit{Comp}(F_1) \cup \mathit{Comp}(F_2) \neq \mathit{Comp}(F_2)$.

7. The upper bound for the OPTIMAL NET ARCHITECTURE problem is established by the fact that $(F, K) \in$ OPTIMAL NET ARCHITECTURE if and only if

$$(\forall f \in \mathit{Comp}(F))(\exists f' \in \mathit{Comp}(\mathcal{F}_K))(\forall \vec{x}) \left[f(\vec{x}) = f'(\vec{x}) \right],$$

where $\mathcal{F}_K$ is defined as in Section 4. This problem is $\mathcal{NP}$-hard since it contains the OPTIMAL EQUIVALENT NET problem, which is $\mathcal{NP}$-hard as shown above, as a special case. This result is also independent of the particular size measure used.

□

References

E. Baum and D. Haussler [1989]. "What Size Net Gives Valid Generalization?", *Neural Computation,* **1**(2) (1989), 151–160.

A. Blum and R. L. Rivest [1988]. "Training a 3-Node Neural Network is $\mathcal{NP}$-Complete," *Proceedings of the First ACM Workshop on the Computational Learning Theory,* Cambridge, MA (August 1988), 9–18.

A. Blumer, A. Ehrenfeucht, D. Haussler, and M. K. Warmuth [1989]. "Learnability and the Vapnik-Chervonenkis Dimension," *Journal of the Association for Computing Machinery,* **36**(4) (October 1989), 929–965.

M. L. Dertouzos [1965]. *Threshold Logic: A Synthesis Approach,* MIT Press, Cambridge, MA (1965).

M. R. Garey and D. S. Johnson [1979]. *Computers and intractability: A Guide to the Theory of $\mathcal{NP}$-completeness,* W. H. Freeman and Co., San Francisco, CA (1979).

D. Haussler [1988]. "Quantifying Inductive Bias: AI Learning Algorithms and Valiant's Learning Framework," *Artificial Intelligence,* **36** (1988), 177–221.

G. E. Hinton [1989]. "Connectionist Learning Procedures," *Artificial Intelligence,* **40** (1989), 185–234.

J. S. Judd [1987]. "Complexity of Connectionist Learning with Various Node Functions," COINS Technical Report No. 87–60, University of Massachusetts (July 1987).

J. S. Judd [1988]. "On the Complexity of Loading Shallow Neural Networks," *Journal of Complexity,* **4** (1988), 177–192.

W. J. Masek [1978]. "Some $\mathcal{NP}$-Complete Set Cover Problems," MIT Laboratory for Computer Science, unpublished manuscript.

D. E. Rumelhart, G. E. Hinton, and R. J. Williams [1986]. "Learning Internal Representations by Error Propagation," *Parallel Distributed Processing,* edited by D. E. Rumelhart and J. E. McClelland, MIT Press, Cambridge, MA (1986), 318–362.

L. J. Stockmeyer [1977]. "The Polynomial-Time Hierarchy," *Theoretical Computer Science*, **3** (1977), 1–22.

L. J. Stockmeyer and A. R. Meyer [1973]. "Word Problems Requiring Exponential Time: Preliminary Report," *Proceedings of the Fifth Annual Symposium on the Theory of Computing* (1973), 1–9.

L. G. Valiant [1984]. "A Theory of the Learnable," *Communications of the ACM,* **27**(11) (November 1984), 1134–1142.

A. Weibel [1989]. "Modular Construction of Time-Delay Neural Networks for Speech Recognition," *Neural Computation,* **1** (1989), 39–46.

A. Weibel and J. Hampshire [1989]. "Building Blocks for Speech," *Byte,* August 1989, 235–242.

C. Wrathall [1977]. "Complete Sets and the Polynomial-Time Hierarchy," *Theoretical Computer Science*, **3** (1977), 23–33.