Bounds for the Computational Power and Learning Complexity of Analog Neural Nets
(Extended Abstract)

Wolfgang Maass*

Institute for Theoretical Computer Science
Technische Universität Graz
Klosterwiesgasse 32/2
A-8010 Graz, Austria
E-mail: maass@igi.tu-graz.ac.at

Abstract

It is shown that high order feedforward neural nets of constant depth with piecewise polynomial activation functions and arbitrary real weights can be simulated for boolean inputs and outputs by neural nets of a somewhat larger size and depth with linear threshold gates and weights from $\{-1,0,1\}$. This provides the first known upper bound for the computational power and VC-dimension of the former type of neural nets. It is also shown that in the case of first order nets with piecewise linear activation functions one can replace arbitrary real weights by rational numbers with polynomially many bits, without changing the boolean function that is computed by the neural net. In order to prove these results we introduce two new methods for reducing nonlinear problems about weights in multi-layer neural nets to linear problems for a transformed set of parameters.

In addition we improve the best known lower bound for the VC-dimension of a neural net with w weights and gates that use the heaviside function (or other common activation functions such as σ) from $\Omega(w)$ to $\Omega(w \log w)$. This implies the somewhat surprising fact that the Baum-Haussler upper bound for the VC-dimension of a neural net with linear threshold gates is asymptotically optimal.

Finally it is shown that neural nets with piecewise polynomial activation functions and a constant number of analog inputs are probably approximately learnable (in Valiant's model for PAC-learning).

*also affiliated with the University of Illinois at Chicago; the research for this paper was partially supported by NSF grant CCR-910406

25th ACM STOC '93-5/93/CA,USA
© 1993 ACM 0-89791-591-7/93/0005/0335...$1.50

1 Introduction

We examine in this paper the computational power and learning complexity of high order analog feedforward neural nets $\mathcal{N}$, i.e. of circuits with analog computational elements in which certain variables are treated as programmable parameters. We focus on neural nets $\mathcal{N}$ of bounded depth in which each gate g computes a function from $\mathbf{R}^m$ into $\mathbf{R}$ of the form $<y_1, \ldots, y_m> \mapsto \gamma^g(Q^g(y_1, \ldots, y_m))$. We assume that for each gate g , γ^g is some fixed piecewise polynomial activation function (also called response function). This function is applied to some polynomial $Q^g(y_1, \ldots, y_m)$ of bounded degree with arbitrary real coefficients, where $y_1, \ldots, y_m$ are the real valued inputs to gate g. One usually refers to the degree of the polynomial Q^g as the "order" of the gate g. The coefficients ("weights") of Q^g are the programmable parameters of $\mathcal{N}$, whose values may arise from some learning process.

We are primarily interested in the case where the neural net $\mathcal{N}$ computes (respectively learns) a boolean valued function. For that purpose we assume that the real valued output of the output gate g_{out} of $\mathcal{N}$ is "rounded off". More precisely, we assume that there is an "outer threshold" T_{out} (which belongs to the programmable parameters of $\mathcal{N}$) such that the output of $\mathcal{N}$ is "1" whenever the real valued output z of g_{out} satisfies $z \geq T_{out}$, and "0" if $z < T_{out}$. In some results of this paper we also assume that the input $<x_1, \ldots, x_n>$ of $\mathcal{N}$ is boolean-valued. It should be noted, that this does not affect the capacity of $\mathcal{N}$ to carry out on its intermediate levels (i.e. in its "hidden units") computations over reals, whose real-valued results are then transmitted to the next layer of gates.

Circuits of this type have rarely been considered in computational complexity theory, and they give rise to the principal question whether these intermediate *analog* computational elements will allow the circuit to compute more complex boolean functions than a circuit with a similar layout but *digital* computational elements. Note that circuits with analog computational elements have an extra source of potentially unlimited parallelism at their disposal, since they can execute operations on numbers of arbitrary bit-length in one step, and they can transmit numbers of arbitrary bit-length from one gate to the next.

One already knows quite a bit about the special case of such neural nets $\mathcal{N}$ where each gate g is a "linear threshold gate". In this case each polynomial $Q^g(y_1, \ldots, y_m)$ is of degree ≤ 1 (i.e. a weighted sum), and each activation function γ^g in $\mathcal{N}$ is the "heaviside function" (also called "hard limiter") $\mathcal{H}$ defined by $\mathcal{H}(y) = 1$ if $y \geq 0$ and $\mathcal{H}(y) = 0$ if $y < 0$ (e. g. see [R], [Ni], [Mu], [MP], [PS], [HMPST], [GHR], [SR], [SBKH], [BH], [A], [L]). The "analog versus digital" issue does not arise in this case, since the output of each gate is a single bit. A key result for the analysis of these circuits was the discovery of Muroga et. al. (see [Mu]) that it is sufficient to consider for a linear threshold gate with m boolean inputs only weights $\alpha_1, \ldots, \alpha_m$ and a bias α_0 that are integers of size $2^{O(m \log m)}$ (this upper bound is optimal according to a recent result of Hastad [Has]). With the help of this a-priori-bound on the relevant bit-length of weights it is easy to show that the same arrays $(F_n)_{n \in \mathbf{N}}$ of boolean functions $F_n : \{0,1\}^n \to \{0,1\}$ are computable by arrays $(\mathcal{N}_n)_{n \in \mathbf{N}}$ of neural nets of depth $O(1)$ and size $O(n^{O(1)})$ with linear threshold gates, no matter whether one uses as weights arbitrary reals, rationals, integers, or elements of $\{-1, 0, 1\}$; see [Mu], [CSV], [HMPST], [GHR], [MT]. The resulting class of arrays $(F_n)_{n \in \mathbf{N}}$ of boolean functions is called (nonuniform-) TC^0 ([HMPST], [J]).

In comparison, very little is known about upper bounds for the computational power and the learning complexity of feedforward neural nets whose gates g employ more general types of activation functions γ^g. This holds in spite of the fact that "real neurons and real physical devices have continuous input-output relations" (Hopfield [Ho]). In the analysis of information processing in natural neural systems one usually views the firing rate of a neuron as its current output. Such firing rates are known to change between a very small number of spikes and several hundred spikes per second (see ch. 20 in [MR]). Hence the activation function γ^g of a gate g that models such a neuron should have a "graded response". It should also be noted that the customary learning algorithms for artificial neural nets (such as backwards propagation [RM]) are based on gradient descent methods, which *require* that all gates g employ *smooth* activation functions γ^g.

In addition, it has frequently been pointed out that it is both biologically plausible and computationally relevant to consider gates g that pass to γ^g instead of a weighted sum $\sum_{i=1}^{m} \alpha_i y_i + \alpha_0$ some polynomial $Q^g(y_1, \ldots, y_m)$ of bounded degree, where $y_1, \ldots, y_m$ are circuit inputs or outputs of the immediate predecessors of g. Such gates are called sigma-pi units or high order gates in the literature (see p. 73 and ch. 10 in [RM], also [DR], [H], [PG], [MD]). Apparently Theorem 3.1 and Theorem 4.4 of this paper provide the first upper bounds for the computational power and learning complexity of high order feedforward neural nets with non-boolean activation functions.

The power of feedforward neural nets with other activation functions besides $\mathcal{H}$ has previously been investigated in [RM] (ch.10), [S1], [S2], [H], [MSS], [DS], [SS]. In particular Sontag [S2] constructed an arbitrarily smooth monotone function Θ and neural nets $\mathcal{N}_n$ of size 2 (!) with activation function Θ such that $\mathcal{N}_n$ can compute with sufficiently large weights *any* boolean function $F_n : \{0,1\}^n \to \{0,1\}$ (hence $\mathcal{N}_n$ has VC-dimension 2^n).

It has also been shown in [MSS] that for the very simple piecewise linear function π defined by $\pi(y) = 0$ if $y \leq 0$, $\pi(y) = y$ if $0 \leq y \leq 1$, and $\pi(y) = 1$ if $y \geq 1$ ([L] refers to a gate g with $\gamma^g = \pi$ as a "threshold logic element") there are functions $f : \{0,1\}^n \to \{0,1\}$ that can (for any n) be computed on some constant size neural nets of depth 2 with activation function π and small integer weights, but which cannot be computed by any constant size neural nets of depth 2 with linear threshold gates (and arbitrary weights). [DS] exhibits an even stronger increase in computational power for the case of quadratic activation functions.

Hence even *simple* non-boolean activation functions provide more computational power to a neural net than the heaviside-function. However it has been open by *how much* they can increase the computational power in the presence of arbitrary weights (only the case of polynomially bounded weights and uniformly continuous activation functions was previously covered in [MSS]). E. Sontag has pointed out that known methods do not even suffice to show for a constant depth neural net $\mathcal{N}_n$ of size $O(n^{O(1)})$ with n inputs and activation function π, that there is *any* boolean function $F_n : \{0,1\}^n \to \{0,1\}$ that can *not* be computed on $\mathcal{N}_n$ with a suitable weight-assignment. Correspondingly no better upper bound than the trivial 2^n could be given for the VC-dimension of such $\mathcal{N}_n$ (with n boolean inputs). From the technical point of view, this inability was caused by the lack of an upper bound on the amount of information that can be encoded in such neural net by the assignment of weights. For this model it is no longer sufficient to analyze a single gate with boolean inputs and outputs. Even if the inputs and outputs of the neural net are boolean valued, the "signals" that are transmitted between the hidden units are real valued. Furthermore one can give no a-priori bound on the precision required for such analog signals between hidden units, since one has no control over the maximal size of weights in the neural net. Note that a computation on a multi-layer neural net of the here considered type involves *products* of weights from subsequent levels. Hence, if some of the weights are arbitrarily large, one needs arbitrarily high precision for the other weights.

The main technical contribution of this paper are two new methods for reducing nonlinear problems about weights in multi-layer neural nets to linear problems for a transformed set of parameters. These two methods are presented in the sections 2 and 3 of this paper. We introduce in section 2 a method that allows us to prove an upper bound for the information-capacity of weights for neural nets $\mathcal{N}_n$ of constant depth and polynomial size with piecewise linear activation functions (hence in particular for π). As a consequence one can simulate any such analog neural net by a digital neural net of constant depth and polynomial size with the heaviside activation function (i.e. linear threshold gates) and weights from $\{-1,0,1\}$. This result also implies that the VC-dimension of $\mathcal{N}_n$ can be bounded above by a polynomial in n.

In section 3 we introduce another proof-technique, that allows us to derive the same two consequences also for neural nets with piecewise *polynomial* activation functions and nonlinear gate-inputs $Q^g(y_1,\ldots,y_m)$ of bounded degree.

In section 4 we derive in Theorem 4.1 a new *lower* bound of $\Omega(w \cdot \log w)$ for the maximal possible VC-dimension of a neural net with w weights. This result implies that the well-known upper bound of [BH] for neural nets with linear threshold gates is optimal up to constant factors. We conclude section 4 with a positive result for learning on neural nets in Valiant's model [V] for probably approximately correct learning ("PAC-learning"). We exploit here the implicit "linearization" of the requirements for the desired weight-assignment that is achieved in the new proof-techniques from sections 2 and 3.

More detailed proofs of the results of this extended abstract can be found in [M].

The following definitions are used throughout this paper.

Definition 1.1 *A network architecture (or "neural net") $\mathcal{N}$ of order k is a labelled acyclic directed graph $<V,E>$. Its nodes of fan-in 0 are labelled by the input variables $x_1,\ldots,x_n$. Each node g of fan-in $m>0$ is called a computation node (or gate), and is labelled by some activation function $\gamma^g : \boldsymbol{R} \rightarrow \boldsymbol{R}$ and some polynomial $Q^g(y_1,\ldots y_m)$ of degree $\leq k$. Furthermore $\mathcal{N}$ has a unique node of fan-out 0, which is called the output node of $\mathcal{N}$ and which carries as an additional label a certain real number T_{out} (called "the outer threshold of $\mathcal{N}$").*

The coefficients of all polynomials $Q^g(y_1,\ldots y_m)$ for gates g in $\mathcal{N}$ and the outer threshold T_{out} are called the programmable parameters of $\mathcal{N}$. Assume that $\mathcal{N}$ has w programmable parameters, and that some numbering of these has been fixed. Then each assignment $\underline{\alpha} \in \boldsymbol{R}^w$ of reals to the programmable parameters in $\mathcal{N}$ defines an analog circuit $\mathcal{N}^{\underline{\alpha}}$, which computes a function $\underline{x} \mapsto \mathcal{N}^{\underline{\alpha}}(\underline{x})$ from $\boldsymbol{R}^n$ into $\{0,1\}$ in the following way: Assume that some input $\underline{x} \in \boldsymbol{R}^n$ has been assigned to the input nodes of $\mathcal{N}$. If a gate g in $\mathcal{N}$ has m immediate predecessors in $<V,E>$ which output $y_1,\ldots,y_m \in \boldsymbol{R}$, then g outputs $\gamma^g(Q^g(y_1,\ldots,y_m))$. Finally, if g_{out} is the output gate of $\mathcal{N}$ and g_{out} gives the real valued output z (according to the preceding inductive definition) we define $\mathcal{N}^{\underline{\alpha}}(\underline{x}) = 1$ if $z \geq T_{out}$ and $\mathcal{N}^{\underline{\alpha}}(x) = 0$ if $z < T_{out}$, where T_{out} is the outer threshold that has been assigned by $\underline{\alpha}$ to g_{out}.

Any parameters that occur in the definitions of the activation functions γ^g of $\mathcal{N}$ are refered to as architectural parameters of $\mathcal{N}$.

Definition 1.2 *A function $\gamma : \boldsymbol{R} \rightarrow \boldsymbol{R}$ is called piecewise polynomial if there are thresholds $t_1,\ldots,t_k \in \boldsymbol{R}$ and polynomials $P_0,\ldots,P_k$ such that $t_1 < \ldots < t_k$ and for each $i \in \{0,\ldots,k\} : t_i \leq x < t_{i+1} \Rightarrow \gamma(x) = P_i(x)$ (we set $t_0 := -\infty$ and $t_{k+1} := \infty$).*

If k is chosen minimal for γ, we refer to k as the number of polynomial pieces of γ, to $P_0,\ldots,P_k$ as the polynomial pieces of γ, and to $t_1,\ldots,t_k$ as the thresholds of γ. Furthermore we refer to $t_1,\ldots,t_k$ together with all coefficients in the polynomials $P_0,\ldots,P_k$ as the parameters of γ. The maximal degree of $P_0,\ldots,P_k$ is called the degree of γ. If the degree of γ is ≤ 1 then we call γ piecewise linear, and we refer to $P_0,\ldots,P_k$ as the linear pieces of γ. Note that we do not require that γ is continuous (or monotone).

If γ occurs as activation function γ^g of some network architecture $\mathcal{N}$, then one refers to the parameters of γ as architectural parameters of $\mathcal{N}$.

Definition 1.3 *Assume that $\mathcal{N}$ is an arbitrary network architecture with n inputs and w programmable parameters, and $S \subseteq \boldsymbol{R}^n$ is an arbitrary set. Then one defines the VC-dimension of $\mathcal{N}$ over S in the following way: VC-dimension$(\mathcal{N},S) := max\{\,|S'|\,\big|\, S' \subseteq S$ has the property that for every function $F : S' \rightarrow \{0,1\}$ there exists a parameter assignment $\underline{\alpha} \in \boldsymbol{R}^w$ such that $\forall \underline{x} \in S'(\mathcal{N}^{\underline{\alpha}}(\underline{x}) = F(\underline{x}))\}$.*

"VC-dimension" is an abbreviation for "Vapnik-Chervonenkis dimension". It has been shown in [BEHW] (see also [BH], [A]) that the VC-dimension of a network architecture $\mathcal{N}$ determines the number of examples that are needed to train $\mathcal{N}$.

2 A Bound for the Information - Capacity of Weights in Neural Nets with Piecewise Linear Activation Functions

We consider for arbitrary $a \in \mathbf{N}$ the following set of rationals with up to a bits before and after the comma: $\mathbf{Q}_a := \left\{ r \in \mathbf{Q} \,\middle|\, r = s \cdot \sum_{i=-a}^{a-1} b_i \cdot 2^i \text{ for } b_i \in \{0,1\}, i = -a, \ldots, a-1 \text{ and } s \in \{-1,1\} \right\}$. Note that for any $r \in \mathbf{Q}_a$: $|r| \leq 2^a \leq 2^{2a} \cdot \min\{|r'| \mid r' \in \mathbf{Q}_a \text{ and } r' \neq 0\}$.

Theorem 2.1 *Consider an arbitrary network architecture $\mathcal{N}$ of order 1 over a graph $< V, E >$ with n input nodes, in which every computation node has fan-out ≤ 1. Assume that each activation function γ^g in $\mathcal{N}$ is piecewise linear with parameters from $\mathbf{Q}_a$. Let $w := |V| + |E| + 1$ be the number of programmable parameters in $\mathcal{N}$.*

Then for every $\underline{\alpha} \in R^w$ there exists a vector $\underline{\alpha}' = < \frac{s_1}{t}, \ldots, \frac{s_w}{t} > \in Q^w$ with integers $s_1, \ldots, s_w, t$ of absolute value $\leq (2w+1)! \cdot 2^{2a(2w+1)}$ such that $\forall \underline{x} \in Q_a^n \left(\mathcal{N}^{\underline{\alpha}}(\underline{x}) = \mathcal{N}^{\underline{\alpha}'}(\underline{x}) \right)$. In particular $\mathcal{N}^{\underline{\alpha}'}$ computes the same boolean function as $\mathcal{N}^{\underline{\alpha}}$.

Remark 2.2 The condition of Theorem 2.1 that all computation nodes in $\mathcal{N}$ have fan-out ≤ 1 is automatically satisfied for $d \leq 2$. For larger d one can simulate any network architecture $\mathcal{N}$ of depth d with s nodes by a network architecture $\mathcal{N}'$ with $\leq \frac{s}{s-1} \cdot s^{d-1} \leq \frac{3}{2} s^{d-1}$ nodes and depth d that satisfies this condition. Hence this condition is not too restrictive for network architectures of a constant depth d.

It should also be pointed out that there is in the assumption of Theorem 2.1 no explicit bound on the number of linear pieces of γ^g (apart from the requirement that its thresholds are from $\mathbf{Q}_a$). For example these activation functions may consist of 2^a linear pieces (with discontinuous jumps in between). Furthermore γ^g is not required to be continuous or monotone.

Remark 2.3 Previously one had *no* upper bound for the computational power (or for the VC-dimension) of multi-layer neural nets $\mathcal{N}$ with arbitrary weights and analog computational elements (i.e. activation functions with non-boolean output). Theorem 2.1 implies that any $\mathcal{N}$ of the considered type can compute with the help of arbitrary parameter assignments $\underline{\alpha} \in \mathbf{R}^w$ at most $2^{O(w^2(a+\log w))}$ different functions from $\mathbf{Q}_a^n$ into $\{0,1\}$, hence VC-dimension $(\mathcal{N}, \mathbf{Q}_a^n) = O(w^2(a+\log w))$ (see Remark 3.3 for a slightly better bound, and for a related bound for the case of inputs from $\mathbf{R}^n$).

Furthermore Theorem 2.1 implies that one can *replace* all *analog computations inside $\mathcal{N}$* by *digital arithmetical operations on not too large integers* (the proof gives an upper bound of $O(wa + w\log w)$ for their bit-length). Thus one can simulate for inputs from $\{0,1\}^n$ any depth d network architecture $\mathcal{N}$ as in Theorem 2.1 with arbitrary parameter assignments $\underline{\alpha} \in \mathbf{R}^w$ by a network architecture of depth $O(d)$ and size $O(a^{O(1)} \cdot w^{O(1)})$ with heaviside-gates and weights from $\{-1,0,1\}$ ([CSV], [PS], [HMPST], [GHR], [SR], [SBKH]). The same holds for inputs from $\mathbf{Q}_a^n$ if they are given to $\mathcal{N}$ in digital form.

Sketch of the proof of Theorem 2.1 We use the following result.

Lemma 2.4 (folklore; see [MT], [M]) *Consider a system $A\underline{x} \leq \underline{b}$ of some arbitrary finite number of linear inequalities in l variables. Assume that all entries in A and $\underline{b}$ are integers of absolute value $\leq a$.*

If this system has any solution in R^l, then it has a solution of the form $< \frac{s_1}{t}, \ldots \frac{s_l}{t} >$, where $s_1, \ldots, s_l, t$ are integers of absolute value $\leq (2l+1)!\, a^{2l+1}$. ∎

The difficulty of the proof of Theorem 2.1 lies in the fact that with *analog* computational elements one can no longer treat each gate separately, since intermediate values are no longer integers. Furthermore the total computation of $\mathcal{N}$ can in general *not* be described by a system of *linear* inequalities, where the w programmable parameters of $\mathcal{N}$ are the variables in the inequalities (and the circuit inputs as well as the architectural parameters of $\mathcal{N}$ are the constants). This becomes obvious if one just considers the composition of two very simple analog gates g_1 and g_2 on levels 1 and 2 of $\mathcal{N}$, whose activation functions γ_1, γ_2 satisfy $\gamma_1(y) = \gamma_2(y) = y$. Assume $x = \sum_{i=1}^{n} \alpha_i x_i + \alpha_0$ is the input to gate g_1, and g_2 receives as input $\sum_{j=1}^{m} \alpha'_j y_j + \alpha'_0$ where $y_1 = \gamma_1(x) = x$ is the output of gate g_1. Then g_2 outputs $\alpha'_1 \cdot \left(\sum_{i=1}^{n} \alpha_i x_i + \alpha_0 \right) + \sum_{j=2}^{m} \alpha'_j y_j + \alpha'_0$. Obviously this term is not linear in the weights $\alpha'_1, \alpha_1, \ldots, \alpha_n$. Hence if the output of gate g_2 is compared with a fixed threshold at the next gate, the resulting inequality is not linear in the weights of the gates in $\mathcal{N}$.

If the activation functions of all gates in $\mathcal{N}$ were linear (as in the example for g_1 and g_2), then there would be no problem because a composition of linear functions is linear. However for *piecewise* linear activation functions it is not sufficient to consider their composition, since intermediate results have to be compared with boundaries between linear pieces of the next gate.

We introduce in this paper a new method in order to handle this difficulty. We simulate $\mathcal{N}^{\underline{\alpha}}$ by another neural net $\hat{\mathcal{N}}[\underline{c}]^{\hat{\underline{\alpha}}}$ (which one may view as a "normal form" for $\mathcal{N}^{\underline{\alpha}}$) that uses the same graph $< V, E >$ as

$\mathcal{N}$, but different activation functions and different values $\hat{\underline{\alpha}}$ for its programmable parameters. The activation functions of $\hat{\mathcal{N}}[\underline{c}]$ depend on $|V|$ new architectural parameters $\underline{c} \in \mathbf{R}^{|V|}$, which we call *scaling parameters* in the following. Although this new neural net has the *disadvantage* that it requires $|V|$ additional parameters $\underline{c}$, it has the *advantage* that we can choose in $\hat{\mathcal{N}}[\underline{c}]$ all weights on edges *between* computation nodes to be from $\{-1, 0, 1\}$. Since these weights from $\{-1, 0, 1\}$ are already of the desired bit-length, we can treat them as constants in the system of inequalities that describes computations of $\hat{\mathcal{N}}[\underline{c}]$. Thereby we can achieve that all variables that appear in the inqualities that describe computations of $\hat{\mathcal{N}}[\underline{c}]$ (the variables for weights of gates on level 1, the variables for the biases of gates on all levels, the variable for the outer threshold, *and the new variables for the scaling parameters* $\underline{c}$) appear only *linearly* in those inqualities. Hence we can apply Lemma 2.4 to the system of inequalities that describes the computations of $\hat{\mathcal{N}}$ for inputs from $\mathbf{Q}_a^n$, and thereby get a "nice" solution $\hat{\underline{\alpha}}', \underline{c}'$ for the programmable parameters $\hat{\alpha}$ and the architectural parameters $\underline{c}$ in $\hat{\mathcal{N}}$. Finally we observe that we can transform $\hat{\mathcal{N}}[\underline{c}']^{\hat{\underline{\alpha}}'}$ back into the original network architecture $\mathcal{N}$ with an assignment of small "numbers" $\underline{\alpha}'$ to all programmable parameters in $\mathcal{N}$.

Consider the activation function γ of an arbitrary gate g in $\mathcal{N}$. Since γ is piecewise linear, there are fixed architectural parameters $t_1 < \cdots < t_k$, $a_0, \ldots, a_k$, $b_0, \ldots, b_k$ in $\mathbf{Q}_a$ (which may be different for different gates g) such that with $t_0 := -\infty$ and $t_{k+1} := +\infty$ one has $\gamma(x) = a_i x + b_i$ for $x \in \mathbf{R}$ with $t_i \leq x < t_{i+1}$; $i = 0, \ldots, k$. For an arbitrary scaling parameter $c \in \mathbf{R}^+$ we associate with γ the following piecewise linear activation function γ^c: the thresholds of γ^c are $c \cdot t_1, \cdots, c \cdot t_k$ and its output is $\gamma^c(x) = a_i x + c \cdot b_i$ for $x \in \mathbf{R}$ with $c \cdot t_i \leq x < c \cdot t_{i+1}$; $i = 0, \ldots, k$ (set $c \cdot t_0 := -\infty$, $c \cdot t_{k+1} := +\infty$). Thus for all reals $c > 0$ the function γ^c is related to γ through the equality: $\forall x \in \mathbf{R}\ (\gamma^c(c \cdot x) = c \cdot \gamma(x))$.

Assume that $\underline{\alpha} \in \mathbf{R}^w$ is some arbitrary given assignment to the programmable parameters in $\mathcal{N}$. We transform $\mathcal{N}^{\underline{\alpha}}$ into a "normal form" $\hat{\mathcal{N}}[\underline{c}]^{\hat{\underline{\alpha}}}$ in which all weights on edges between computation nodes are from $\{-1, 0, 1\}$, such that $\forall \underline{x} \in \mathbf{R}^n \left(\mathcal{N}^{\underline{\alpha}}(\underline{x}) = \hat{\mathcal{N}}[\underline{c}]^{\hat{\underline{\alpha}}}(\underline{x})\right)$. We proceed inductively from the output level towards the input level. Assume that the output gate g_{out} of $\mathcal{N}^{\underline{\alpha}}$ receives as input $\sum_{i=1}^{m} \alpha_i y_i + \alpha_0$, where $\alpha_1, \ldots, \alpha_m, \alpha_0$ are the weights and the bias of g_{out} (under the assignment $\underline{\alpha}$) and $y_1, \ldots, y_m$ are the (real valued) outputs of the immediate predecessors $g_1, \ldots, g_m$ of g. For each $i \in \{1, \ldots, m\}$ with $\alpha_i \neq 0$ such that g_i is not an input node we replace the activation function γ_i of g_i by $\gamma_i^{|\alpha_i|}$, and we multiply the weights and the bias of gate g_i with $|\alpha_i|$. Finally we replace the weight α_i of gate g_{out} by $\text{sgn}(\alpha_i)$, where $\text{sgn}(\alpha_i) := 1$ if $\alpha_i > 0$ and $\text{sgn}(\alpha_i) := -1$ if $\alpha_i < 0$. This operation has the effect that the multiplication with $|\alpha_i|$ is carried out *before* the gate g_i (rather than after g_i, as done in $\mathcal{N}^{\underline{\alpha}}$), but that the considered output gate g_{out} still receives the same input as before. The analogous operation is then inductivily carried out for the predecessors g_i of g_{out} (note however that the weights of g_i are no longer the original ones from $\mathcal{N}^{\underline{\alpha}}$, since they have been changed in the preceding step). We exploit here the assumption that each gate has fan-out ≤ 1.

Let $\hat{\underline{\alpha}}$ consist of the new weights on edges adjacent to input nodes, of the resulting biases of all gates in $\hat{\mathcal{N}}$, and of the (unchanged) outer threshold T_{out}. Let $\underline{c}$ consist of the resulting scaling parameters at the gates of $\mathcal{N}$. Then we have $\forall \underline{x} \in \mathbf{R}^n \left(\mathcal{N}^{\underline{\alpha}}(\underline{x}) = \hat{\mathcal{N}}[\underline{c}]^{\hat{\underline{\alpha}}}(\underline{x})\right)$.

Finally we have to replace all *strict* inequalities of the form "$s_1 < s_2$" that are needed to describe the computation of $\hat{\mathcal{N}}[\underline{c}]^{\hat{\underline{\alpha}}}$ for some input $\underline{x} \in \mathbf{Q}_a^n$ by inequalities of the form "$s_1 + 1 \leq s_2$". This concerns inequalities of the form $s < c \cdot t_i$, where $c \cdot t_i$ is the threshold of some gate g in $\hat{\mathcal{N}}[\underline{c}]$ and s is its gate input, inequalities of the form $s < T_{out}$ where s is the output of g_{out}, and inequalities of the form $0 < c$ for each scaling parameter c. In order to achieve this stronger separation it is sufficient to multiply all parameters $\hat{\underline{\alpha}}$, $\underline{c}$ in $\hat{\mathcal{N}}$ by a sufficiently large constant K. For simplicity we write again $\hat{\underline{\alpha}}$, $\underline{c}$ for the resulting parameters. We now specify a system $A\underline{z} \leq \underline{b}$ of linear inequalities in w variables $\underline{z}$ that play the role of the w parameters $\hat{\underline{\alpha}}$, $\underline{c}$ in the computations of $\hat{\mathcal{N}}[\underline{c}]^{\hat{\underline{\alpha}}}$ for all inputs $\underline{x}$ from $\mathbf{Q}_a^n$. The constants of these inequalities are the coordinates of all inputs $\underline{x} \in \mathbf{Q}_a^n$, the architectural parameters of the activation functions γ in $\mathcal{N}$, the constants $-1, 1$ that occur in $\hat{\mathcal{N}}$ as weights of edges between computation nodes, and the constant 1 that arises from the replacement of strict inequalities "$s_1 < s_2$" by "$s_1 + 1 \leq s_2$".

For each fixed input $\underline{x} \in \mathbf{Q}_a^n$ one places into the system $A\underline{z} \leq \underline{b}$ up to two linear inequalities for each gate g in $\mathcal{N}$. These inequalities are defined by induction on the depth of g. If g has depth 1, $t_1 < \cdots < t_k$ are the thresholds of its activation function γ in $\mathcal{N}$, and its input $\sum_{i=1}^{n} \alpha_i x_i + \alpha_0$ in $\hat{\mathcal{N}}[\underline{c}]^{\hat{\underline{\alpha}}}$ satisfies $c \cdot t_j \leq \sum_{i=1}^{n} \alpha_i x_i + \alpha_0$ and $\sum_{i=1}^{n} \alpha_i x_i + \alpha_0 + 1 \leq c \cdot t_{j+1}$, then one adds these two inequalities to the system (more precisely: if $j = 0$ or $j = k$ then only one inequality is needed since the other one is automatically true).

If g' is a successor gate of g, it receives from g for some specific $j \in \{0, \ldots, k\}$ an output of the form

$a_j \cdot (\sum_{i=1}^{n} \alpha_i x_i + \alpha_0) + c \cdot b_j$ (where c is the scaling parameter of gate g). Note that this term is linear, since a_j, b_j are fixed parameters of gate g. In this way one can express for circuit input $\underline{x}$ the input $\mathrm{I}(\underline{x})$ of gate g' as a term that is linear in the weights, biases and scaling parameters of its preceding gates (we exploit here that in $\hat{\mathcal{N}}$ the weight on the edge between g' and each predecessor gate is a fixed parameter from $\{-1, 0, 1\}$, not a variable). If this input $\mathrm{I}(\underline{x})$ satisfies in $\hat{\mathcal{N}}[\underline{c}]^{\hat{\underline{\alpha}}}$ the inequalities $c' \cdot t'_{j'} \le \mathrm{I}(\underline{x})$ and $\mathrm{I}(\underline{x}) + 1 \le c' \cdot t'_{j'+1}$ (where $t'_1 < \ldots < t'_{k'}$ are the thresholds of g' in $\mathcal{N}$, and c' is the scaling parameter of g' in $\hat{\mathcal{N}}$), then one adds these two inequalities to the system $\mathcal{A}\underline{z} \le \underline{b}$ (respectively only one if $j' = 0$ or $j' = k'$). Note that all resulting inequalities are linear, in spite of the fact that they may contain variables for the scaling parameters and biases of *all* gates. It should also be pointed out that the definition of this system of inequalities is more involved than it may first appear, since the sum of terms $I(\underline{x})$ depends on the chosen inequalities for all predecessor gates (e.g. on j in the example above).

It is clear that the resulting system $\mathcal{A}\underline{z} \le \underline{b}$ has a solution in $\mathbf{R}^w$, since $\underline{z} := <\hat{\underline{\alpha}}, \underline{c}>$ is a solution. Hence we can apply Lemma 2.4, which provides a solution $\underline{z}'$ of the form $< \frac{s_i}{t} >_{i=1,\ldots,w}$ with integers $s_1, \ldots, s_w, t$ of absolute value $\le (2w+1)!\ 2^{2a(2w+1)}$. Let $\hat{\mathcal{N}}[\underline{c}']^{\hat{\underline{\alpha}}'}$ be the net $\hat{\mathcal{N}}$ with this new assignment $< \hat{\underline{\alpha}}', \underline{c}' > := \underline{z}'$ of "small" parameters. By definition we have $\forall \underline{x} \in \mathbf{Q}_a^n (\mathcal{N}^{\underline{\alpha}}(\underline{x}) = \hat{\mathcal{N}}[\underline{c}']^{\hat{\underline{\alpha}}'})$. We show that one can transform this net $\hat{\mathcal{N}}[\underline{c}']^{\hat{\underline{\alpha}}'}$ into a net $\mathcal{N}^{\underline{\alpha}'}$ with the *same activation functions* as $\mathcal{N}^{\underline{\alpha}}$ but a new assignment $\underline{\alpha}'$ of "small" parameters (that can easily be computed from $\hat{\underline{\alpha}}', \underline{c}'$). This transformation proceeds inductively from the input level towards the output level. Consider some gate g on level 1 in $\hat{\mathcal{N}}$ that uses (for the new parameter assignment $\underline{c}'$) the scaling parameter $c > 0$ for its activation function γ. Then we replace the weights $\alpha_1, \ldots, \alpha_n$ and bias α_0 of gate g in $\hat{\mathcal{N}}[\underline{c}']^{\hat{\underline{\alpha}}'}$ by $\frac{\alpha_1}{c}, \ldots, \frac{\alpha_n}{c}, \frac{\alpha_0}{c}$, and γ^c by γ. Furthermore if $r \in \{-1, 0, 1\}$ was in $\hat{\mathcal{N}}$ the weight on the edge between g and its successor gate g', we assign to this edge the weight $c \cdot r$. Note that g' receives in this way from g the same input as in $\hat{\mathcal{N}}[\underline{c}']^{\hat{\underline{\alpha}}'}$ (for every circuit input). Assume now that $\alpha'_1, \ldots, \alpha'_m$ are the weights that the incoming edges of g' get assigned in this way, that α'_0 is the bias of g' in the assignment $\underline{z}' = <\hat{\underline{\alpha}}', \underline{c}'>$, that $c' > 0$ is the scaling parameter of g' in $\hat{\mathcal{N}}[\underline{c}']^{\hat{\underline{\alpha}}'}$, that γ' is the activation function of g' in $\mathcal{N}$. Then we assign the new weights $\frac{\alpha'_1}{c'}, \ldots, \frac{\alpha'_m}{c'}$ and the new bias $\frac{\alpha'_0}{c'}$ to g', and we multiply the weight on the outgoing edge from g' by c'.

By construction we have that $\forall \underline{x} \in \mathbf{R}^n\ (\mathcal{N}^{\underline{\alpha}'}(\underline{x}) = \hat{\mathcal{N}}[\underline{c}']^{\hat{\underline{\alpha}}'}(\underline{x}))$, hence
$\forall \underline{x} \in \mathbf{Q}_a^n\ (\mathcal{N}^{\underline{\alpha}'}(\underline{x}) = \mathcal{N}^{\underline{\alpha}}(\underline{x}))$. ∎

3 Upper Bounds for Neural Nets with Piecewise Polynomial Activation Functions

Theorem 3.1 *Consider an arbitrary array $(\mathcal{N}_n)_{n \in \mathbf{N}}$ of network architectures $\mathcal{N}_n$ of order $O(1)$ and of depth $O(1)$ with n inputs and $O(n^{O(1)})$ gates, in which the gate function γ^g of each gate g is piecewise polynomial of degree $O(1)$ with $O(n^{O(1)})$ polynomial pieces, and where the architectural parameters of γ^g are arbitrary reals (γ^g need not be continuous, or monotone).*

Then there exists an array $(\tilde{\mathcal{N}}_n)_{n \in \mathbf{N}}$ of network architectures $\tilde{\mathcal{N}}_n$ of order 1 and of depth $O(1)$ with n inputs and $O(n^{O(1)})$ gates, such that each gate g in $\tilde{\mathcal{N}}_n$ uses as its activation function the heaviside function $\mathcal{H}$ (i.e. g is a linear threshold gate), and such that for each assignment $\underline{\alpha}$ of arbitrary reals to the programmable parameters in $\mathcal{N}_n$ there is an assignment $\tilde{\underline{\alpha}}$ of $O(n^{O(1)})$ numbers from $\{-1, 0, 1\}$ to the programmable parameters in $\tilde{\mathcal{N}}_n$ such that $\forall \underline{x} \in \{0,1\}^n$ $(\mathcal{N}_n^{\underline{\alpha}}(\underline{x}) = \tilde{\mathcal{N}}_n^{\tilde{\underline{\alpha}}}(\underline{x}))$.

In particular VC-dimension$(\mathcal{N}_n, \{0,1\}^n) = O(n^{O(1)})$. Furthermore for any assignment $(\underline{\alpha}_n)_{n \in \mathbf{N}}$ of real valued parameters the boolean functions that are computed by $(\mathcal{N}_n^{\underline{\alpha}_n})_{n \in \mathbf{N}}$ are in TC^0.

Remark 3.2 Very recently Macintyre and Sontag [MS] have proven *finite* upper bounds for the VC-dimension of network architectures with the activation function $\sigma(y) = 1/(1 + e^{-y})$ and inputs from $\mathbf{R}^n$. However no *polynomial* upper bounds are known for such nets for either real or binary inputs. The preceding Theorem 3.1 provides polynomial upper bounds for the VC-dimension of network architectures whose activation functions are spline approximations to σ of any fixed degree (see Remark 3.3 for the case of real inputs).

Idea of the proof of Theorem 3.1 This proof is quite long and involved, even for the simplest nonlinear case where the activation functions consist of 2 polynomial pieces of degree 2. Note that in contrast to the model in [SS] the magnitude of the given weights in $\mathcal{N}_n$ may grow arbitrarily fast as a function of n.

We first note that by the following observation we may assume w.l.o.g. that the given network architectures $\mathcal{N}_n$ are of order 1. One has $y \cdot z = \frac{1}{2}((y+z)^2 - y^2 - z^2)$. Hence one can eliminate nonlinear polynomials Q^g as arguments of activation functions by introducing intermediate gates with linear gate inputs and quadratic activation functions.

One can transform each net $\mathcal{N}_n^{\underline{\alpha}_n}$ into a normal form $\hat{\mathcal{N}}_n^{\hat{\underline{\alpha}}_n}$ of constant depth and size $O(n^{O(1)})$ with all weights on edges between computation nodes from $\{-1, 0, 1\}$, in which all gates g have fan-out ≤ 1, and all gates g (except for the output gate) use as activation functions γ^g piecewise polynomial functions of the following special type: γ^g consists of up to 3 pieces, of which at most one is not identically 0, and in which the nontrivial piece outputs a constant, or computes a power $y \mapsto y^k$ (where $k \in \mathbf{N}$ satisfies $k = O(1)$). We can choose $\underline{\hat{\alpha}}_n$ such that one has "$s_1 + 1 \leq s_2$" for all strict inequalities "$s_1 < s_2$" that arise in $\hat{\mathcal{N}}_n^{\underline{\hat{\alpha}}_n}$ for inputs from $\{0,1\}^n$ when one compares some intermediate term s_1 with the threshold s_2 of some gate, or with the outer threshold (analogously as in the proof of Theorem 2.1). This transformation can be done in such a way that $\forall \underline{x} \in \{0,1\}^n(\mathcal{N}_n^{\underline{\alpha}_n}(\underline{x}) = \hat{\mathcal{N}}_n^{\underline{\hat{\alpha}}_n}(\underline{x}))$.

Although we have now transformed $\mathcal{N}_n^{\underline{\alpha}_n}$ into a similar "normal form" $\hat{\mathcal{N}}_n^{\underline{\hat{\alpha}}_n}$ as in the proof of Theorem 2.1, a new source of non-linearity arises when one tries to describe the role of the programmable parameters of $\hat{\mathcal{N}}_n$ by a system of inequalities. Assume for example that g is a gate on level 1 with input $\alpha_1 x_1 + \alpha_2 x_2$ and activation function $\gamma^g(y) = y^2$. Then this gate g outputs $\alpha_1^2 x_1^2 + 2\alpha_1\alpha_2 x_1 x_2 + \alpha_2^2 x_2^2$. Hence the variables α_1, α_2 will not occur linearly in an inequality which describes the comparison of the output of g with some threshold of a gate at the next level.

We solve this new problem in the following way. We fix an arbitrary assignment $\underline{\hat{\alpha}}_n$ of real numbers to the programmable parameters of $\hat{\mathcal{N}}_n$. We introduce for the system of inequalities $L(\hat{\mathcal{N}}_n^{\underline{\hat{\alpha}}_n}, \{0,1\}^n)$ (that describes the computations of $\hat{\mathcal{N}}_n^{\underline{\hat{\alpha}}_n}$ for all inputs $\underline{x} \in \{0,1\}^n$) new variables v for all nontrivial parameters in $\hat{\mathcal{N}}_n^{\underline{\hat{\alpha}}_n}$ (i.e. for the weights on edges between input nodes and computation nodes, for the bias of each gate g, for the output of gates with constant output, for the outer threshold T_{out} *and for the thresholds* t_1^g, t_2^g *of each gate* g). In addition we introduce new variables for all *products* of such parameters that arise in the computation of $\hat{\mathcal{N}}_n^{\underline{\hat{\alpha}}_n}$. We have to keep the inequalities linear in order to apply Lemma 2.4. Hence we cannot demand in these inequalities that the value of the variable $v_{v_1^g, v_2^g}$ (that represents the product of α_1^g and α_2^g) is the product of the values of the variables v_1^g and v_2^g (that represent the weights α_1^g respectively α_2^g). We solve this problem by describing in detail in the linear inequalities $L(\hat{\mathcal{N}}_n^{\underline{\hat{\alpha}}_n}, \{0,1\}^n)$ which *role* the product of α_1^g and α_2^g plays in the computations of $\hat{\mathcal{N}}_n^{\underline{\hat{\alpha}}_n}$ for inputs from $\{0,1\}^n$. It turns out that this can be done in such a way that *it does not matter* whether a solution $\mathcal{A}$ of $L(\hat{\mathcal{N}}_n^{\underline{\hat{\alpha}}_n}, \{0,1\}^n)$ assigns to the variable $v_{v_1^g, v_2^g}$ a value $\mathcal{A}(v_{v_1^g, v_2^g})$ that is equal to the product of the values $\mathcal{A}(v_1^g)$ and $\mathcal{A}(v_2^g)$. In any case $\mathcal{A}(v_{v_1^g, v_2^g})$ is forced to *behave like* the product of $\mathcal{A}(v_1^g)$ and $\mathcal{A}(v_2^g)$ in the computations of $\hat{\mathcal{N}}_n^{\underline{\hat{\alpha}}_n}$.

In more abstract terms, one may view $\mathcal{A}$ as a model of a certain "linear fragment" $L(\hat{\mathcal{N}}_n^{\underline{\hat{\alpha}}_n}, \{0,1\}^n)$ of the theory of the role of the parameters $\underline{\hat{\alpha}}_n$ in the computations of $\hat{\mathcal{N}}_n^{\underline{\hat{\alpha}}_n}$ on inputs form $\{0,1\}^n$. Such model $\mathcal{A}$ (which will be given by Lemma 2.4) is some type of "nonstandard model" of the theory of computations of $\hat{\mathcal{N}}_n$, since it replaces products of weights by "nonstandard products". Such nonstandard model $\mathcal{A}$ does not provide a new assignment of (small) weights to the net $\hat{\mathcal{N}}_n$, only to a "nonstandard version" $C^{\mathcal{A}}(\hat{\mathcal{N}}_n^{\underline{\hat{\alpha}}_n})$ of the net $\hat{\mathcal{N}}_n^{\underline{\hat{\alpha}}_n}$. However the linear fragment $L(\hat{\mathcal{N}}_n^{\underline{\hat{\alpha}}_n}, \{0,1\}^n)$ can be chosen in such a way that $C^{\mathcal{A}}(\hat{\mathcal{N}}_n^{\underline{\hat{\alpha}}_n})$ computes the same boolean function as $\hat{\mathcal{N}}_n^{\underline{\hat{\alpha}}_n}$. Furthermore, if $\mathcal{A}$ consists of a solution with "small" values as given by Lemma 2.4, then $C^{\mathcal{A}}(\hat{\mathcal{N}}_n^{\underline{\hat{\alpha}}_n})$ can be simulated by a constant-depth polynomial-size boolean circuit whose gates g are all MAJORITY-gates. This implies that the boolean functions that are computed by $(C^{\mathcal{A}}(\hat{\mathcal{N}}_n^{\underline{\hat{\alpha}}_n}))_{n\in\mathbf{N}}$ are in TC^0. However by construction these are the same boolean functions that are computed by $(\mathcal{N}_n^{\underline{\alpha}_n})_{n\in\mathbf{N}}$.

We refer to [M] for a detailed account of this proof. ∎

Remark 3.3 Sontag [S3] suggested using the "quasi-linearization" that is achieved in the proof of Theorem 3.1 in order to also get polynomial upper bounds for the VC-dimension of constant depth neural nets $\mathcal{N}_n$ with piecewise polynomial activation functions over the *continuous domain* $\mathbf{R}^n$: One counts the number of components into which the weightspace is partitioned by the hyperplanes that are defined by some arbitrary finite set $S \subseteq \mathbf{R}^n$ of inputs.

By letting $\underline{\alpha}_n$ vary and keeping the network architecture $\mathcal{N}_n$ and the input $\underline{x} \in S$ fixed one gets up to $2^{O(n^{O(1)})}$ different systems $L(\hat{\mathcal{N}}_n^{\underline{\hat{\alpha}}_n}, \underline{x})$ in the proof of Theorem 3.1. Hence the total number l_n of linear inequalities that arise in this way for different $\underline{x} \in S$ and different parameters $\underline{\alpha}_n$ is bounded by $|S| \cdot 2^{O(n^{O(1)})}$. Furthermore the total number w_n of variables that occur in these l_n inequalities is bounded by $O(n^{O(1)})$. This implies with the help of Theorem 1.3 in [E] that VC-dimension $(\mathcal{N}_n, \mathbf{R}^n) = O(n^{O(1)})$.

One can apply in a similar fashion the "linearization" that is achieved in the proof of Theorem 2.1. Consider a neural net $\mathcal{N}$ over a graph $<V, E>$ as in Theorem 2.1, but allow that each activation function γ^g consists of $\leq p$ linear pieces with arbitrary fixed *real* parameters. Then one can show that VC-dimension $(\mathcal{N}, \mathbf{R}^n) = O(w^2 \log p)$, where $w := |V| + |E| + 1$ is the number of programmable parameters in $\mathcal{N}$.

The preceding results, which first appeared in [M], have subsequently been generalized by [GJ].

4 Further Results about Learning on Neural Nets

All existing *lower* bounds for the VC-dimension of neural nets with common activation functions such as $\mathcal{H}, \pi$, and σ (with $\sigma(y) := 1/(1+e^{-y})$) are at best linear in the number of edges (respectively weights) of the net ([BH], [B], [S2]). The following result provides the first *superlinear* lower bound.

Theorem 4.1 *One can construct for arbitrarily large $n \in N$ network architectures $\mathcal{N}_n$ of depth 4 with n inputs and $\leq 33n$ edges such that VC-dimension $(\mathcal{N}_n, \{0,1\}^n) = n \cdot \log n$. These network architectures $\mathcal{N}_n$ can be realized with any of the common activation functions $\mathcal{H}, \pi, \sigma$ at its gates.*

Remark 4.2 This result solves an open problem of Baum and Haussler [BH]. It shows that their upper bound of $2w \log(eN) = O(w \cdot \log w)$ for the VC-dimension of an arbitrary neural net with linear threshold gates, w weights, and N nodes (see [C], [BH]) is optimal up to constant factors. The constant e denotes here the base of the natural logarithm.

Idea of the proof of Theorem 4.1 One applies some Shannon-type upper bound on circuit size, which shows that *all* boolean functions $F : \{0,1\}^m \rightarrow \{0,1\}$ can be computed by some threshold circuit C_F of depth 4 with $O\left(\frac{2^m}{m}\right)$ edges. One just has to verify in addition that these circuits C_F use the same underlying graph. It is easy to show that these requirements are satisfied by the construction of Neciporuk [N], which was later improved by Lupanov [Lu]. ∎

We now turn to the analysis of *computationally efficient learning* on analog neural nets in Valiant's model [V] for probably approximately correct learning ("PAC-learning"). More precisely we consider the common extension of this model to real valued domains due to [BEHW]. Unfortunately most results about PAC-learning on neural nets are negative (see [BR], [KV]). This could either mean that learning on neural nets is impossible, or that the common theoretical analysis of learning on neural nets is not quite adequate.

We want to point here to one somewhat problematic point of the traditional asymptotic analysis of PAC-learning on neural nets. In analogy to the standard asymptotic analysis of the runtime of algorithms in terms of the number n of input bits one usually formalizes PAC-learning on neural nets in exactly the same fashion. However in contrast to the common situation for computer algorithms (which typically receive their input in digital form as a long sequence of n bits) for many important applications of neural nets the input is given in analog form as a vector of a *small* number n of *analog* real valued parameters. These relatively few input parameters may consist for example of sensory data, or they may be the relevant components of a longer feature vector (which were extracted by some other mechanism). If one analyzes PAC-learning on neural nets in this fashion, the relevant asymptotic problem becomes a different one: Can a given analog neural net with a fixed number n of analog inputs approximate the target concept arbitrarily close after it has been shown sufficiently many training examples?

Theorem 4.3 *Let $\mathcal{N}$ be an arbitrary network architecture as in Theorem 2.1, where the architectural parameters of the piecewise linear activation functions may now be arbitrary reals. Let $C_{\mathcal{N}} := \{C \subseteq \boldsymbol{R}^n \mid \exists \underline{\alpha} \in \boldsymbol{R}^w \forall \underline{x} \in \boldsymbol{R}^n\, (\chi_C(\underline{x}) = \mathcal{N}^{\underline{\alpha}}(\underline{x}))\}$ be the associated concept class over the domain $\boldsymbol{R}^n$, where χ_C is the characteristic function of a concept C. Then $C_{\mathcal{N}}$ is properly PAC-learnable (i.e. $C_{\mathcal{N}}$ is PAC-learnable with hypotheses from $C_{\mathcal{N}}$).*

Sketch of the proof of Theorem 4.3 We have VC-dimension $(C_{\mathcal{N}}) < \infty$ by Remark 3.3. Hence according to [BEHW] it suffices to show that for any given set S of m examples for some target concept C_T one can compute from S within a number of computation steps that is polynomial in $m, \frac{1}{\varepsilon}, \frac{1}{\delta}$ an assignment $\underline{\alpha}_S \in \mathbf{R}^w$ to the variable parameters of $\mathcal{N}$ such that $\forall \underline{x} \in S(\chi_{C_T}(\underline{x}) = \mathcal{N}^{\underline{\alpha}_S}(\underline{x}))$. The construction in the proof of Theorem 2.1 implies that it is sufficient if one computes instead with polynomially in $m, \frac{1}{\varepsilon}, \frac{1}{\delta}$ computation steps an assignment $\underline{\hat{\alpha}}_S, \underline{c}_S$ of parameters for the associated neural net $\hat{\mathcal{N}}$ such that $\forall \underline{x} \in S\left(\chi_{C_T}(\underline{x}) = \hat{\mathcal{N}}[\underline{c}_S]^{\underline{\hat{\alpha}}_S}(\underline{x})\right)$. The latter task is easier because the role of the parameters $\underline{\hat{\alpha}}, \underline{c}$ in a computation of $\hat{\mathcal{N}}$ for a specific input $\underline{x}$ can be described by *linear* inequalities (provided one knows which linear piece is used at each gate).

Nevertheless the following technical problem remains. Although we know which *output* $\hat{\mathcal{N}}[\underline{c}_S]^{\underline{\hat{\alpha}}_S}$ should give for an input $\underline{x} \in S$, we do not know *in which way* this output should be produced by $\hat{\mathcal{N}}[\underline{c}_S]^{\underline{\hat{\alpha}}_S}$. More specifically, we don't know which particular piece of each piecewise linear activation function γ^g of $\hat{\mathcal{N}}$ will be used for this computation. However this detailed information would be needed for each $\underline{x} \in S$ and for all gates g of $\hat{\mathcal{N}}$ in order to describe the resulting constraints on the parameters $\underline{\hat{\alpha}}, \underline{c}$ by a system of linear inequalities. However one can generate a set of polynomially in m many systems of linear inequalities such that at least one of these systems provides for all $\underline{x} \in S$ satisfiable and sufficient constraints for $\underline{\hat{\alpha}}, \underline{c}$.

For each of the resulting polynomially in m many systems of inequalities we apply the method of the proof of Lemma 2.4 (i.e. we reduce the solution of each system of inequalities to the solution of polynomially in m many systems of linear equalities), or we apply Megiddo's polynomial time algorithm for linear programming in a fixed dimension [Me] in order to find values $\underline{\alpha}_S, \underline{c}_S$ for which $\tilde{\mathcal{N}}[\underline{c}_S]^{\underline{\alpha}_S}$ gives the desired outputs for all $\underline{x} \in S$. By construction, this algorithm will succeed for at least one of the selected system of inequalities. ∎

Theorem 4.4 *Let $\mathcal{N}$ be an arbitrary high order network architecture with arbitrary piecewise polynomial activation functions. Then the associated concept class $\mathcal{C}_{\mathcal{N}}$ is PAC-learnable with an hypothesis class of the form $\mathcal{C}_{\tilde{\mathcal{N}}}$ for a somewhat larger network architecture $\tilde{\mathcal{N}}$.*

Idea of the proof of Theorem 4.4: One uses as hypotheses sets which are defined by a neural net $\tilde{\mathcal{N}}$ of the same structure as the circuits $C^{\mathcal{A}}(\mathcal{N})$ in the detailed proof of Theorem 3.1 in [M]. For this network architecture $\tilde{\mathcal{N}}$ one can express the constraints on the assignment $\mathcal{A}$ by *linear* inequalities. Remark 3.3 implies that VC-dimension $(\tilde{\mathcal{N}}, \mathbf{R}^n) < \infty$. Hence by [BEHW] it suffices to show that for any set S of m examples for a target concept from $\mathcal{C}_{\mathcal{N}}$ one can compute in polynomial time a consistent hypothesis from $\mathcal{C}_{\tilde{\mathcal{N}}}$. This can be done by applying the method from the proof of Lemma 2.4 in a manner analogous to the proof of Theorem 4.3; or by applying linear programming in a fixed dimension [Me] to polynomially in m many systems of linear inequalities. There is one small obstacle in generating the associated partitions of S, since the corresponding inequalities are not linear in the circuit inputs $\underline{x}$. One overcomes this difficulty by going to an input space of higher dimension. ∎

Acknowledgements

We would like to thank Eduardo D. Sontag for drawing our attention to the problem of finding upper bounds for neural nets with π-gates, and for his insightful comments. We thank Peter Auer, Franz Aurenhammer, Eric Baum, David Haussler, Philip M. Long, György Turan, and Gerhard Wöginger for various helpful discussions on this research. We also thank György Turan for making his translation of the work of Neciporuk available to us.

References

[A] Y. S. Abu-Mostafa, "The Vapnik-Chervonenkis dimension: information versus complexity in learning", *Neural Computation*, vol. 1, 1989, 312 - 317

[B] P. L. Bartlett, "Lower bounds on the Vapnik-Chervonenkis dimension of multilayer threshold networks", *preprint* (July 1992)

[BH] E. B. Baum, D. Haussler, "What size net gives valid generalization?", *Neural Computation*, vol. 1, 1989, 151 - 160

[BR] A. Blum, R. L. Rivest, "Training a 3-node neural network is NP-complete", *Proc. of the 1988 Workshop on Computational Learning Theory*, Morgan Kaufmann (San Mateo, 1988), 9 - 18

[CSV] A. K. Chandra, L. Stockmeyer, U. Vishkin, "Constant depth reducibility", *SIAM J. Computing*, vol. 13 (2), 1984, 423 - 439

[C] T. M. Cover, "Capacity problems for linear machines", in: *Pattern Recognition*, L. Kanal ed., *Thompson Book Co.*, 1988, 283 - 289

[DS] B. DasGupta, G. Schnitger, "Efficient approximations with neural networks: a comparison of gate functions", *preprint* (Aug. 1992)

[DR] R. Durbin, D. E. Rumelhart, "Product units: a computationally powerful and biologically plausible extension to backpropagation networks", *Neural Computation*, vol. 1, 1989, 133 - 142

[E] H. Edelsbrunner, "Algorithms in Combinatorial Geometry", Springer (Berlin, 1987)

[EOS] H. Edelsbrunner, J. O'Rourke, R. Seidel, "Constructing arrangements of lines and hyperplanes with applications", *SIAM J. Comp.*, vol. 15, 1986, 341 - 363

[GJ] P. Goldberg, M. Jerrum, "Bounding the Vapnik - Chervonenkis dimension of concept classes parameterized by real numbers", *preprint* (February 1993).

[GHR] M. Goldmann, J. Hastad, A. Razborov, "Majority gates vs. general weighted threshold gates", *Proc. of the 7th Structure in Complexity Theory Conference*, 1992, 2 - 13

[HMPST] A. Hajnal, W. Maass, P. Pudlak, M. Szegedy and G. Turan, "Threshold circuits of bounded depth", *Proc. of the 28th Annual IEEE Symp. on Foundations of Computer Science*, 1987, 99 - 110. Full version to appear in *J. Comp. Syst. Sci.* 1993

[Has] J. Hastad, "On the size of weights for threshold gates", *preprint* (September 1992)

[H] D. Haussler, "Decision theoretic generalizations of the PAC model for neural nets and other learning applications", *Information and Computation*, vol. 100, 1992, 78 - 150

[Ho] J. J. Hopfield, "Neurons with graded response have collective computational properties like those of two-state neurons", *Proc. Nat. Acad. of Sciences USA*, 1984, 3088 - 3092

[J] D. S. Johnson, "A catalog of complexity classes", in: *Handbook of Theoretical Computer Science* vol. A, J. van Leeuwen ed., *MIT Press* (Cambridge, 1990)

[KV] M. Kearns, L. Valiant, "Cryptographic limitations on learning boolean formulae and finite automata", *Proc. of the 21st ACM Symposium on Theory of Computing*, 1989, 433 - 444

[L] R. P. Lippmann, "An introduction to computing with neural nets", *IEEE ASSP Magazine*, 1987, 4 - 22

[Lu] O. B. Lupanov, "On circuits of threshold elements", *Dokl. Akad. Nauk* SSSR, vol. 202, 1288 - 1291; engl. translation in: *Sov. Phys. Dokl.*, vol. 17, 1972, 91 - 93

[M] W. Maass, "Bounds for the computational power and learning complexity of analog neural nets", *IIG - Report 349 of the Technische Universität Graz*, (October 1992).

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

[MT] W. Maass, G. Turan, "How fast can a threshold gate learn?", in: *Computational Learning Theory and Natural Learning Systems: Constraints and Prospects*, G. Drastal, S. J. Hanson and R. Rivest eds., *MIT Press*, to appear

[MS] A. Macintyre, E. D. Sontag, "Finiteness result for sigmoidal "neural" networks", *Proc. of the 25th ACM Symposium on Theory of Computing*, 1993.

[MR] J. L. McClelland, D. E. Rumelhart "Parallel Distributed Processing", vol. 2, *MIT Press* (Cambridge, 1986)

[Me] N. Megiddo, "Linear Programming in linear time when the dimension is fixed", *J. of the ACM*, vol. 31, 1984, 114 - 127

[MP] M. Minsky, S. Papert, "Perceptrons: An Introduction to Computational Geometry", Expanded Edition, *MIT Press* (Cambridge, 1988)

[MD] J. Moody, C. J. Darken, "Fast learning in networks of locally-tuned processing units", *Neural Computation*, vol. 1, 1989, 281 - 294

[Mu] S. Muroga, "Threshold Logic and its Applications", *Wiley* (New York, 1971)

[N] E. I. Neciporuk, "The synthesis of networks from threshold elements", *Probl. Kibern.* No. 11, 1964, 49 - 62; engl. translation in: *Autom. Expr.*, vol. 7, No. 1, 1964, 35 - 39

[Ni] N. J. Nilsson, Learning Machines, McGraw-Hill (New York, 1971)

[PS] I. Parberry, G. Schnitger, "Parallel computation with threshold functions", *Lecture Notes in Computer Science* vol. 223, Springer (Berlin, 1986), 272 - 290

[PG] T. Poggio, F. Girosi, "Networks for approximation and learning", *Proc. of the IEEE*, vol. 78(9), 1990, 1481 - 1497

[R] F. Rosenblatt, "Principles of Neurodynamics", *Spartan Books* (New York, 1962)

[RM] D. E. Rumelhart, J. L. McClelland, "Parallel Distributed Processing", vol. 1, *MIT Press* (Cambridge, 1986)

[Sch] A. Schrijver, "Theory of Linear and Integer Programming", *Wiley* (New York, 1986)

[SS] H. T. Siegelmann, E. D. Sontag, "Neural networks with real weights: analog computational complexity", *Report SYCON*-92-05, Rutgers Center for Systems and Control (Oct. 1992)

[SBKH] K. Y. Siu, J. Bruck, T. Kailath, T. Hofmeister, "Depth efficient neural networks for division and related problems", to appear in *IEEE Transactions on Inf. Theory*

[SR] K. Y. Siu, V. Roychowdhury, "On optimal depth threshold circuits for multiplication and related problems", *Tech. Report* ECE - 92-05, University of California, Irvine (March 1992)

[S1] E. D. Sontag, "Remarks on interpolation and recognition using neural nets", in: *Advances in Neural Information Processing Systems* 3, R. P. Lippmann, J. Moody, D. S. Touretzky, eds., Morgan Kaufmann (San Mateo, 1991), 939 - 945

[S2] E. D. Sontag, "Feedforward nets for interpolation and classification", *J. Comp. Syst. Sci.*, vol. 45, 1992, 20 - 48

[S3] E. D. Sontag, private communication (July 1992)

[T] G. Turan, private notes (1989)

[V] L. G. Valiant, "A theory of the learnable", *Comm. of the ACM*, vol. 27, 1984, 1134 - 1142