# Bounds for the Computational Power and Learning Complexity of Analog Neural Nets

**(Extended Abstract)**

**Wolfgang Maass***

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

October 23, 1992

**Abstract**

It is shown that 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 heaviside gates and weights from $\{0,1\}$. This provides the first known upper bound for the computational power and VC-dimension of such neural nets. It is also shown that in the case of 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 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 $\sigma$) 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 heaviside 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

# 1 Introduction

We examine in this paper the computational power and learning complexity of analog feedforward neural nets $\mathcal{N}$, i.e. of circuits with analog computational elements in which certain parameters are treated as programmable variables. 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 $g$ , $\gamma^g$ 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 $g$. The coefficients ("weights") of $Q^g$ are the programmable variables 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 variable 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") computation 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 $\leq 1$ (i.e. a weighted sum), and each activation function $\gamma^g$ in $\mathcal{N}$ is the "heaviside function" (also called "hard limiter") $\mathcal{H}$ defined by

$$\mathcal{H}(y) = \begin{cases} 1 \ , \text{ if } y \geq 0 \\ 0 \ , \text{ if } y < 0 \end{cases}$$

(e. g. see [R], [Ni], [M], [MP], [PS], [HMPST], [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. Still, it requires some work to bound the potential power of arbitrary weights (in the weighted sums) for the compution of boolean functions on such circuit. Since there are only finitely many boolean circuit inputs, it is obvious that only rational weights have to be considered. The key result for the analysis of these circuits was the discovery

of Muroga et. al. (see [M]) 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 $\alpha_0$ that are integers of size $2^{O(m \log m)}$ (this upper bound is optimal according to a recent result of Hastad [Has]). This upper bound follows from a well-known result about linear inequalities (see Lemma 2.4). With the help of this a-priori-bound on the relevant bit-length of weights it is easy to show that 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 can compute the same arrays $(F_n)_{n \in \mathbf{N}}$ of boolean functions $F_n : \{0,1\}^n \to \{0,1\}$, no matter whether one uses as weights arbitrary reals, rationals, integers, or single bits (i.e. all weights are from $\{0,1\}$); see [M], [CSV], [HMPST], [MT]. The resulting class of arrays $(F_n)_{n \in \mathbf{N}}$ of boolean functions is called (nonuniform-) $\mathrm{TC}^0$ in computational complexity theory ([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 $\gamma^g$. 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 few and several hundred spikes per second (see ch. 20 in [MR]). Hence the activation function $\gamma^g$ 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 $\gamma^g$.

In addition, it has frequently been pointed out that it is both biologically plausible and computationally relevant to consider gates $g$ that pass to $\gamma^g$ 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 in the literature (see p. 73 and ch. 10 in [RM], also [DR], [H], [PG], [MD]). From the point of view of approximation theory there has been particular interest in the case where $Q^g(y_1, \ldots, y_m) = \sum_{i=1}^{m} \alpha_i (y_i - c_i)^2$ measures a "distance" of its input $< y_1, \ldots, y_m >$ from some "center" $< c_1, \ldots, c_m >$ (the latter may be determined through a learning process). Apparently Corollary 3.2 and Theorem 4.5 of this paper provide the first upper bounds for the computational power and learning complexity of neural nets with non-boolean activation functions that employ non-linear gate inputs $Q^g(y_1, \ldots, y_m)$ with arbitrary coefficients.

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]. It was shown in [MSS] for a very general class of activation functions $\gamma^g$ that neural nets $(\mathcal{N}_n)_{n \in \mathbf{N}}$ of constant depth and size $O(n^{O(1)})$ with real weights of size $O(n^{O(1)})$ and separation $\Omega(1/n^{O(1)})$ (between the un-rounded circuit-outputs for rejected and accepted inputs) can compute only boolean functions in $\mathrm{TC}^0$. It follows from a result of Sontag [S2] that the assumptions on the weight-size and separation are essential for this upper bound: he constructed an arbitrarily smooth monotone function $\Theta$ (which can be made to satisfy the conditions on $\gamma^g$ in the quoted result of [MSS]) and neural nets

$\mathcal{N}_n$ of size 2 (!) with activation function $\Theta$ 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 $2^n$).

These results leave open the question about the computational power and learning complexity of neural nets with arbitrary weights that employ common analog activation functions $\gamma^g$, such as the piecewise linear function $\pi$ defined by

$$\pi(y) = \begin{cases} 0, & \text{if } y \leq 0 \\ y, & \text{if } 0 \leq y \leq 1 \\ 1, & \text{if } y \geq 1 \end{cases}$$

([L] refers to a gate $g$ with $\gamma^g = \pi$ as a "threshold logic element"). It has already been shown in [MSS] that constant size neural nets of depth 2 with activation function $\pi$ and small integer weights can compute more boolean functions than 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). 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 $\pi$, 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 the case of neural nets with heaviside gates this upper bound on the information-capacity of weights is provided by the quoted result of Muroga et. al. [M]. But no such upper bound has been known for any activation function with non-boolean output.

We introduce in section 2 of this paper a method that allows us to prove an upper bound for the information-capacity of weights for neural nets with piecewise linear activation functions (hence in particular for $\pi$). It is shown that for the computation of boolean functions on neural nets $\mathcal{N}_n$ of constant depth and polynomially in $n$ many gates (where $n$ is the number of input variables) it is sufficient to use as weights rational numbers with polynomially in $n$ many bits. 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 binary weights (i.e. weights from $\{0,1\}$). 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. These results show that in spite of the previously quoted evidence for the superiority of non-boolean activation functions in neural nets, there is some limit to their computational power as long as

the activation functions are piecewise polynomial. On the other hand the polynomial upper bound on the VC-dimension of such neural nets may be interpreted as *good news*: It shows that neural nets of this type can in principle be trained with a sequence of examples that is not too long.

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 consider the problem of learning on neural nets with a fixed number of analog (i.e. real valued) input variables. 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. In this way one can show that such neural nets are properly PAC-learnable in the case of piecewise linear activation functions, and PAC-learnable with a hypothesis class that is given by a somewhat larger neural net in the case of piecewise polynomial activation functions.

The following notion of a feedforward neural net (with linear gate inputs) is the main model of this paper. However Corollary 3.2 and Theorem 4.5 also hold for the more general case of arbitrary polynomials $Q^g(y_1, \ldots, y_m)$ of bounded degree as gate inputs.

**Definition 1.1** *A (feedforward) neural net $\mathcal{N}$ (with linear gate inputs) 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 $> 0$ is called a computation node (or gate), and is labelled by some activation function $\gamma^g : \boldsymbol{R} \to \boldsymbol{R}$. Furthermore $\mathcal{N}$ has a unique node of fan-out 0, which is called the output node of $\mathcal{N}$.*

*For $w := |V| + |E| + 1$ the neural net $\mathcal{N}$ has exactly $w$ variable parameters:*
*- a weight for each edge of the graph $< V, E >$*
*- a bias for each gate of $\mathcal{N}$*
*- an outer threshold $T_{out}$ for the output gate $g_{out}$ of $\mathcal{N}$.*
*We assume that some numbering of all edges and gates in $\mathcal{N}$ has been fixed. Then each assignment $\underline{\alpha} \in \boldsymbol{R}^w$ of reals to the variable 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 $\gamma^g(\sum_{i=1}^{m} \alpha_i^g y_i + \alpha_0^g)$, where $\alpha_1^g, \ldots, \alpha_m^g$ are the weights that have been assigned by $\underline{\alpha}$ to the corresponding incoming edges of $g$, and $\alpha_0^g$ is the bias that has been assigned by $\underline{\alpha}$ to the gate $g$. 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*

$$\mathcal{N}^{\underline{\alpha}}(\underline{x}) := \begin{cases} 1, & \text{if } z \geq T_{out} \\ 0, & \text{if } z < T_{out} \end{cases},$$

*where $T_{out}$ is the outer threshold that has been assigned by $\underline{\alpha}$ to $g_{out}$.*

**Definition 1.2** *Assume that $\mathcal{N}$ is an arbitrary neural net with $n$ inputs and $w$ variable 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:*

$$\text{VC-dimension}(\mathcal{N}, S) := \max\{\, |S'| \;\big|\; S' \subseteq S \text{ has the property that for every function } F : S' \to \{0,1\} \text{ there exists a parameter assignment } \underline{\alpha} \in \mathbf{R}^w \text{ such that } \forall \underline{x} \in S'(\mathcal{N}^{\underline{\alpha}}(\underline{x}) = F(\underline{x}))\}.$$

**Remark 1.3** "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 neural net $\mathcal{N}$ determines the number of examples that are needed to train $\mathcal{N}$ (in Valiant's model for probably approximately correct learning [V]). Sontag [S2] has shown that the VC-dimension of a neural net can be drastically increased by using activation functions with non-boolean output instead of the heaviside function $\mathcal{H}$.

**Definition 1.4** *A function $\gamma : \boldsymbol{R} \to \boldsymbol{R}$ 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 $\gamma$, we refer to $k$ as the number of polynomial pieces of $\gamma$, to $P_0, \ldots, P_k$ as the polynomial pieces of $\gamma$, and to $t_1, \ldots, t_k$ as the thresholds of $\gamma$. 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 $\gamma$. The maximal degree of $P_0, \ldots, P_k$ is called the degree of $\gamma$. If the degree of $\gamma$ is $\leq 1$ then we call $\gamma$ piecewise linear, and we refer to $P_0, \ldots, P_k$ as the linear pieces of $\gamma$. Note that we do not require that $\gamma$ is continuous (or monotone).*

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

We define for arbitrary $a \in \mathbf{N}$

$$\mathbf{Q}_a := \left\{ r \in \mathbf{Q} \;\middle|\; r = s \cdot \sum_{i=-a}^{a-1} b_i \cdot 2^i \text{ for } b_i \in \{0,1\}, i = -a, \ldots, a-1 \text{ and } s \in \{-1, 1\} \right\}.$$

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 neural net $\mathcal{N}$ over a graph $< V, E >$ with $n$ input nodes, in which every computation node has fan-out $\leq 1$. Assume that each activation function $\gamma^g$ in $\mathcal{N}$ is piecewise linear with parameters from $\boldsymbol{Q}_a$. Let $w := |V| + |E| + 1$ be the number of variable parameters in $\mathcal{N}$.*

*Then for every $\underline{\alpha} \in \boldsymbol{R}^w$ there exists a vector $\underline{\alpha}' = < \frac{s_1}{t}, \ldots, \frac{s_w}{t} > \in \boldsymbol{Q}^w$ with integers $s_1, \ldots, s_w, t$ of absolute value $\leq (2w+1)! 2^{2a(2w+1)}$ such that $\forall \underline{x} \in \boldsymbol{Q}_a^n \left( \mathcal{N}^{\underline{\alpha}}(\underline{x}) = \mathcal{N}^{\underline{\alpha}'}(\underline{x}) \right)$. 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 $\leq 1$ is automatically satisfied for $d \leq 2$. For larger $d$ one can simulate any neural net $\mathcal{N}$ of depth $d$ with $s$ nodes by a neural net $\mathcal{N}'$ 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 neural nets 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 $\gamma^g$ (apart from the requirement that its thresholds are from $\mathbf{Q}_a$). For example these activation functions may consist of $2^a$ linear pieces.

**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 $2^{O(aw^2 \log w)}$ different functions from $\mathbf{Q}_a^n$ into $\{0,1\}$, hence VC-dimension $(\mathcal{N}, \mathbf{Q}_a^n) = O(aw^2 \log w)$ (see Remark 3.4 for a slightly better bound).

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). It is well-known that each of these digital arithmetical operations (multiple addition, multiplication, division) can be carried out on a circuit of small constant depth with $O(a^{O(1)} \cdot w^{O(1)})$ MAJORITY-gates, hence also on a neural net of depth $O(1)$ and size $O(a^{O(1)} \cdot w^{O(1)})$ with heaviside gates and weights from $\{0,1\}$ ([CSV], [PS], [HMPST], [SR], [SBKH]). Thus one can simulate for inputs from $\{0,1\}^n$ any depth $d$ neural net $\mathcal{N}$ as in Theorem 2.1 with arbitrary parameter assignments $\underline{\alpha} \in \mathbf{R}^w$ by a neural net of depth $O(d)$ and size $O(a^{O(1)} \cdot w^{O(1)})$ with heaviside-gates and weights from $\{0,1\}$. 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:** In the special case where $\gamma^g = \mathcal{H}$ for all gates in $\mathcal{N}$ this result is well known ([M]). It follows by applying separately to each gate in $\mathcal{N}$ the following result.

**Lemma 2.4 (folklore; see [MT] for a proof)** *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 $\boldsymbol{R}^l$, 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 $\leq (2l+1)!\, a^{2l+1}$.*

**Sketch of the proof for Lemma 2.4:** Let $k$ be the number of inequalities in $A\underline{x} \leq \underline{b}$. One writes each variable in $\underline{x}$ as a difference of 2 nonnegative variables, and one adds to each inequality a "slack variable". In this way one gets an equivalent system

$$A'\underline{x}' = \underline{b} \quad , \quad \underline{x}' \geq \underline{0} \tag{1}$$

over $l' := 2l + k$ variables, for some $k \times l'$ matrix $A'$. The $k$ columns of $A'$ for the $k$ slack-variables in $\underline{x}'$ form an identity matrix. Hence $A'$ has rank $k$.

The assumption of the Lemma implies that (1) has a solution over $\mathbf{R}$. Hence by Caratheodory's Theorem (Corollary 7.1i in [Sch]) one can conclude that there is also a

solution over $\mathbf{R}$ of a system

$$A''\underline{x}'' = \underline{b} \quad , \quad \underline{x}'' \geq \underline{0} \tag{2}$$

where $A''$ consists of $k$ linearly independent columns of $A'$. Since $A''$ has full rank, (2) has in fact a unique solution that is given by Cramer's rule: $x_j'' = \det(A_j'')/\det A''$ for $j = 1, \ldots, k$, where $A_j''$ results form $A''$ by replacing its $j^{\text{th}}$ column by $\underline{b}$. Since all except up to $2l$ columns of $A''$ contain exactly one 1 and else only 0's, we can bring each of the matrices $A'', A_j''$ by permutations of rows and columns into a form

$$B = \begin{pmatrix} C & 0 \\ D & I \end{pmatrix}$$

where $C$ is a square matrix with $2l+1$ rows. Hence the determinant of $B$ is an integer 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 $w$ variable parameters of $\mathcal{N}$ are the variables in the inequalities (and the fixed 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 $\gamma_1, \gamma_2$ 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.

Hence the natural approach would be to describe the computations of $\mathcal{N}$ by a system of *polynomial inequalities*, and to prove a suitable generalization of Lemma 2.4 to polynomial inequalities. Unfortunately there is no such generalization: The system $x^2 \leq 2$, $x^2 \geq 2$ has a solution, but no solution of short bit-length.

We introduce in this paper a new method in order to handle this difficulty. We simulate $\mathcal{N}^{\underline{\alpha}}$ by another neural net $\hat{\mathcal{N}}[\underline{c}]^{\underline{\hat{\alpha}}}$ (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 $\underline{\hat{\alpha}}$ for its variable parameters. The activation functions of $\hat{\mathcal{N}}[\underline{c}]$ depend on $|V|$ new 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 $\underline{\hat{\alpha}}', \underline{c}'$ for all variable parameters in $\hat{\mathcal{N}}$. Finally we observe that we can transform $\hat{\mathcal{N}}[\underline{c}']^{\underline{\hat{\alpha}}'}$ back into the original neural net $\mathcal{N}$ with an assignment of small "numbers" $\underline{\alpha}'$ to all variable parameters in $\mathcal{N}$.

We will now fill in some of the missing details. Consider the gate function $\gamma$ of an arbitrary gate $g$ in $\mathcal{N}$. Since $\gamma$ is piecewise linear, there are fixed 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 $\gamma$ the following piecewise linear activation function $\gamma^c$: the thresholds of $\gamma^c$ 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 $\gamma^c$ is related to $\gamma$ 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 variable parameters in $\mathcal{N}$. We transform $\mathcal{N}^{\underline{\alpha}}$ into a "normal form" $\hat{\mathcal{N}}[\underline{c}]^{\underline{\hat{\alpha}}}$ in which all weights on edges between computation nodes are from $\{-1,0,1\}$, such that $\forall \underline{x} \in \mathbf{R}^n \left(\mathcal{N}^{\underline{\alpha}}(\underline{x}) = \hat{\mathcal{N}}[\underline{c}]^{\underline{\hat{\alpha}}}(\underline{x})\right)$. 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 $\gamma_i$ 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 $\alpha_i$ of gate $g_{out}$ by $sgn(\alpha_i) := \begin{cases} 1 & , \text{ if } \alpha_i > 0 \\ -1 & , \text{ if } \alpha_i < 0 \end{cases}$. 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 $\leq 1$.

Let $\underline{\hat{\alpha}}$ 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 factors at the gates of $\mathcal{N}$. Then we have $\forall \underline{x} \in \mathbf{R}^n \left(\mathcal{N}^{\underline{\alpha}}(\underline{x}) = \hat{\mathcal{N}}[\underline{c}]^{\underline{\hat{\alpha}}}(\underline{x})\right)$.

Finally we have to replace all *strict* inequalities of the form "$s_1 < s_2$" that are

needed to describe the computation of $\hat{\mathcal{N}}[\underline{c}]^{\underline{\hat{\alpha}}}$ 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 $\underline{\hat{\alpha}}$, $\underline{c}$ in $\hat{\mathcal{N}}$ by a sufficiently large constant $K$. For simplicity we write again $\underline{\hat{\alpha}}$, $\underline{c}$ for the resulting parameters. We now specify a system $\mathcal{A}\underline{z} \leq \underline{b}$ of linear inequalities in $w$ variables $\underline{z}$ that play the role of the $w$ parameters $\underline{\hat{\alpha}}$, $\underline{c}$ in the computations of $\hat{\mathcal{N}}[\underline{c}]^{\underline{\hat{\alpha}}}$ 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 parameters of the activation functions $\gamma$ in $\mathcal{N}$, the constants $-1, 1$ that occur in $\hat{\mathcal{N}}$ as weights of edges between computation nodes, and the constants 1 that arise 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 $\mathcal{A}\underline{z} \leq \underline{b}$ 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 functions $\gamma$ in $\mathcal{N}$, and its input $\sum_{i=1}^{n} \alpha_i x_i + \alpha_0$ in $\hat{\mathcal{N}}[\underline{c}]^{\underline{\hat{\alpha}}}$ 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 factor 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 linear term in the weights, biases and scaling factors 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 $\hat{\mathcal{N}}[\underline{c}]^{\underline{\hat{\alpha}}}$ the inequalities $c' \cdot t'_{j'} \leq \mathrm{I}(\underline{x})$ and $\mathrm{I}(\underline{x}) + 1 \leq c' \cdot t'_{j'+1}$ (where $t'_1 < \ldots < t'_{k'}$ are the thresholds of $g'$ in $\mathcal{N}$, and $c'$ is the scaling factor of $g'$ in $\hat{\mathcal{N}}$), then one adds these two inequalities to the system $\mathcal{A}\underline{z} \leq \underline{b}$ (respectively only one if $j' = 0$ or $j' = k'$). Note that all resulting inequalities are linear, in spite of the fact that it contains variables for the 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). Hence a precise definition has to be similar to that of the more detailed proof of Theorem 3.1 in the Appendix.

It is clear that the resulting system $\mathcal{A}\underline{z} \leq \underline{b}$ has a solution in $\mathbf{R}^w$, since $\underline{z} :=< \underline{\hat{\alpha}}, \underline{c} >$ 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 $\leq (2w+1)!\, 2^{2a(2w+1)}$. Let $\hat{\mathcal{N}}[\underline{c}']^{\underline{\hat{\alpha}}'}$ be the neural net $\hat{\mathcal{N}}$ with this new assignment $< \underline{\hat{\alpha}}', \underline{c}' >:= \underline{z}'$ of "small" parameters. By definition we have $\forall \underline{x} \in \mathbf{Q}_a^n (\mathcal{N}^{\underline{\alpha}}(\underline{x}) = \hat{\mathcal{N}}[\underline{c}']^{\underline{\hat{\alpha}}'})$. We show that one can transform this neural net $\hat{\mathcal{N}}[\underline{c}']^{\underline{\hat{\alpha}}'}$ 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 $\underline{\hat{\alpha}}', \underline{c}'$). 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 factor $c > 0$ for its activation function $\gamma$. Then we replace the weights $\alpha_1, \ldots, \alpha_n$ and bias $\alpha_0$ 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 $\gamma^c$ by $\gamma$. 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 $\alpha'_0$ is the bias of $g'$ in the assignment $\underline{z}' = < \hat{\underline{\alpha}}', \underline{c}' >$, that $c' > 0$ is the scaling factor of $g'$ in $\hat{\mathcal{N}}[\underline{c}']^{\hat{\underline{\alpha}}'}$, that $\gamma'$ 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 $(\mathcal{N}_n)_{n \in \boldsymbol{N}}$ of neural nets $\mathcal{N}_n$ of depth $O(1)$ with $n$ inputs and $O(n^{O(1)})$ gates, in which the gate function $\gamma^g$ of each gate $g$ is piecewise polynomial of degree $O(1)$ with $O(n^{O(1)})$ polynomial pieces, and where the parameters of $\gamma^g$ are arbitrary reals.*

*Then there exists an array $(\tilde{\mathcal{N}}_n)_{n \in \boldsymbol{N}}$ of neural nets $\tilde{\mathcal{N}}_n$ 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 reals to the variable parameters in $\mathcal{N}_n$ there is an assignment $\tilde{\underline{\alpha}}$ of numbers from $\{0, 1\}$ to the variable 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 $(\underline{\alpha}_n)_{n \in \boldsymbol{N}}$ of real valued parameters the boolean functions that are computed by $(\mathcal{N}_n^{\underline{\alpha}_n})_{n \in \boldsymbol{N}}$ are in $TC^0$.*

**Corollary 3.2** *The claim of Theorem 3.1 remains valid if each gate $g$ in $\mathcal{N}_n$ passes to its activation function $\gamma^g$ some arbitrary polynomial $\boldsymbol{Q}^g(y_1, \ldots, y_m)$ of bounded degree (instead of a weighted sum of $y_1, \ldots, y_m$). In this case all coefficients of the polynomials $\boldsymbol{Q}^g$ are considered as variable parameters ("weights") of $\mathcal{N}_n$.*

*The simulating neural nets $\tilde{\mathcal{N}}_n$ can still be chosen so that each gate in $\tilde{\mathcal{N}}_n$ is a linear threshold gate, and so that $(\tilde{\mathcal{N}}_n)_{n \in \boldsymbol{N}}$ also has all the other properties that are claimed in Theorem 3.1.*

**Idea of the proof of Corollary 3.2 from Theorem 3.1:** 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. In this way one can transform the given array of neural nets into an array of neural nets according to Definition 1.1 that satisfies the

assumptions of Theorem 3.1. ∎

**Remark 3.3** Theorem 3.1 yields no bound for the computational power of neural nets with the activation function $\sigma(y) = 1/(1+e^{-y})$. However it provides bounds for the case where the activation functions are spline approximations to $\sigma$ of arbitrarily high degree $d$, provided that $d \in \mathbf{N}$ is fixed.

**Remark 3.4** Sontag [S3] suggested using the "quasi-linearization" that is achieved in the proof of Theorem 3.1 in order to also get upper bounds for the VC-dimension over $\mathbf{R}^n$, by counting 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 neural net $\mathcal{N}_n$ 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 (see the Appendix for details). 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 $O(n^{O(1)})$. Therefore the hyperplanes that are associated with these $l_n$ inequalities partition the range $\mathbf{R}^{w_n}$ of the variables into at most $\sum_{k=0}^{w_n} \sum_{i=0}^{k} \binom{w_n - i}{k-i} \binom{l_n}{w_n - i} = |S|^{O(n^{O(1)})}$ faces (Theorem 1.3 in [E]). Each $\mathcal{A} \in \mathbf{R}^{w_n}$ gives rise to at most $2^{O(n^{O(1)})}$ different circuits $C^{\mathcal{A}}(\hat{\mathcal{N}}_n^{\underline{\hat{\alpha}}_n})$ when $\mathcal{N}_n$ and $S$ are kept fixed, but the parameters $\underline{\alpha}_n$ (and hence the weights from $\{-1, 0, 1\}$ between computation nodes) vary. Thus each $\mathcal{A} \in \mathbf{R}^{w_n}$ can be used to compute at most $2^{O(n^{O(1)})}$ different functions $S \to \{0,1\}$ on the resulting circuits. Furthermore, if $\mathcal{A}$ and $\tilde{\mathcal{A}}$ belong to the *same* face of the partition of $\mathbf{R}^{w_n}$ then for all $\underline{\alpha}_n$ the circuits $C^{\mathcal{A}}(\hat{\mathcal{N}}_n^{\underline{\hat{\alpha}}_n})$ and $C^{\tilde{\mathcal{A}}}(\hat{\mathcal{N}}_n^{\underline{\hat{\alpha}}_n})$ compute the same function $S \to \{0,1\}$. Hence if $S$ is shattered by $\mathcal{N}_n$ (i.e. *any* function $S \to \{0,1\}$ can be computed by $\mathcal{N}_n^{\underline{\alpha}_n}$ for suitable parameters $\underline{\alpha}_n$) then $2^{|S|} \leq |S|^{O(n^{O(1)})} \cdot 2^{O(n^{O(1)})}$, hence $|S| = O(n^{O(1)})$. This implies that VC-dimension $(\mathcal{N}_n, \mathbf{R}^n) = O(n^{O(1)})$. Note that this estimate remains true if one allows in $\mathcal{N}_n$ also non-linear gate inputs as in Corollary 3.2.

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 $\gamma^g$ 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 variable parameters in $\mathcal{N}$. It is sufficient to observe that for different $\underline{x} \in S$ and different initial assignments $\underline{\alpha}$ altogether at most $|S| \cdot 2^{O(w \log p)}$ linear inequalities arise in the description of the computations of the associated nets $\hat{\mathcal{N}}[\underline{c}]^{\underline{\hat{\alpha}}}$ for input $\underline{x}$. The associated hyperplanes partition the "weight space" $\mathbf{R}^w$ for the variable parameters $\underline{\hat{\alpha}}, \underline{c}$ into $\leq |S|^{O(w)} \cdot 2^{O(w^2 \log p)}$ faces. The vectors from each face can be used to compute at most $2^{O(w)}$ different functions $S \to \{0,1\}$ (note that in general more than one function $S \to \{0,1\}$ can be computed because of different weights from $\{-1,0,1\}$ between computation nodes). Hence $2^{|S|} \leq |S|^{O(w)} \cdot 2^{O(w^2 \log p)} \cdot 2^{O(w)}$ if $S$ is shattered by $\mathcal{N}$, thus $|S| = O(w^2 \log p)$.

**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. We first note that one can transform each given neural net $\mathcal{N}_n^{\underline{\alpha}_n}$

into a normal form $\hat{\mathcal{N}}_n^{\underline{\hat{\alpha}}_n}$ 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 $\leq 1$, and all gates $g$ use as activation functions $\gamma^g$ piecewise polynomial functions of the following special type: $\gamma^g$ 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 variable 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 $\alpha_1, \alpha_2$ 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 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 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 $\alpha_1^g$ and $\alpha_2^g$) is the product of the values of the variables $v_1^g$ and $v_2^g$ (that represent the weights $\alpha_1^g$ respectively $\alpha_2^g$). 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 $\alpha_1^g$ and $\alpha_2^g$ 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 $\mathcal{A}(v_2^g)$ (that are assigned by $\mathcal{A}$ to the variables $v_1^g$ and $v_2^g$). 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 neural net $\hat{\mathcal{N}}_n$, only to a "nonstandard version" $C^{\mathcal{A}}(\hat{\mathcal{N}}_n^{\underline{\hat{\alpha}}_n})$ of the neural 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 $C^{\mathcal{A}}(\hat{\mathcal{N}}_n^{\hat{\underline{\alpha}}_n})$ can be simulated by a constant-depth polynomial-size boolean circuit whose gates $g$ are all MAJORITY-gates (i.e. $g(y_1,\ldots,y_m)=1$ if $\sum_{i=1}^{m} y_i \geq m/2$, otherwise $g(y_1,\ldots,y_m)=0$). This implies that the boolean functions that are computed by $(C^{\mathcal{A}}(\hat{\mathcal{N}}_n^{\hat{\underline{\alpha}}_n}))_{n\in\mathbf{N}}$ are in $\mathrm{TC}^0$. 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 the Appendix for a more detailed proof. ∎

# 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 $\sigma$ (with $\sigma(y) := 1/(1+e^{-y})$) are at best linear in the number $e$ 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 $n \in \mathbf{N}$ neural nets $\mathcal{N}_n$ 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 neural nets $\mathcal{N}_n$ 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.

**Idea of the proof of Theorem 4.1**: One applies some Shannon-type upper bound on circuit size, which shows that *all* boolean functions $F : \{0,1\}^m \to \{0,1\}$ 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 briefly sketch the idea of this construction. Assume that $m$ is a power of 2. Fix some arbitrary function $F : \{0,1\}^m \to \{0,1\}$. Set $a := \frac{m-\log m}{2}$, $b := a+2$, $c := (\log m) - 2$. Thus $a+b+c = m$, $2^a = 2^{m/2}/\sqrt{m}$, $2^c = m/4 \leq a$. Define $g : \{0,1\}^{a+b} \to \{0,1\}^a$ by defining for $j \leq 2^c$ the $j^{\text{th}}$ component of $g(\underline{x},\underline{y})$ to be $F(\underline{x},\underline{y},\underline{z})$, where $\underline{z}$ is a binary representation of $j$. Call a function $q : \{0,1\}^{a+b} \to \{0,1\}^a$ *special* if $q(\cdot,\underline{y})$ is 1-1 for every $\underline{y} \in \{0,1\}^b$. It is not difficult to show that for $g : \{0,1\}^{a+b} \to \{0,1\}^a$ there exist special functions $g_1, g_2, g_3, g_4$ such that $\forall \underline{x} \in \{0,1\}^a \; \forall \underline{y} \in \{0,1\}^b$ :

$$g(\underline{x},\underline{y}) = \begin{cases} g_1(\underline{x},\underline{y}) \oplus g_2(\underline{x},\underline{y}), & \text{if } |\underline{x}| < 2^{a-1} \\ g_3(\underline{x},\underline{y}) \oplus g_4(\underline{x},\underline{y}), & \text{if } |\underline{x}| \geq 2^{a-1} \end{cases}$$

(we write $|\underline{x}|$ for the number whose binary representation is $\underline{x}$). Hence it is essentially sufficient to construct suitable circuits for the special functions $g_1, \ldots, g_4$. Fix some $k \in \{1, \ldots, 4\}$. Assume that one has already subcircuits that compute the $2^b$ characteristic functions $f : \{0,1\}^b \to \{0,1\}$ for singletons from $\{0,1\}^b$. Then one can compute with these subcircuits and with $3 \cdot 2^a + a$ additional linear threshold gates the special function $g_k$. For that purpose one first constructs a circuit with $3 \cdot 2^a$ additional gates that computes $h_k : \{0,1\}^{a+b} \to \{0,1\}^{2^a}$ defined by $h_k(\underline{x}, \underline{y}) := \underline{e}_{|g_k(\underline{x},\underline{y})|}$ (i.e. $h_k(\underline{x}, \underline{y})$ is the $|g_k(\underline{x}, \underline{y})|^{\text{th}}$ unit vector of length $2^a$). Each component $h_{k,i}$ of $h_k$ ($i = 1, \ldots, 2^a$) can be computed with 3 linear threshold gates: We have

$$h_{k,i}(\underline{x}, \underline{y}) \;=\; 1 \;\Leftrightarrow\; \sum_{r=1}^{a} x_r \cdot 2^{r-1} \;=\; \sum_{j=0}^{2^b-1} c_j^i \cdot S_{bin(j)}(\underline{y}) \;,$$

where $bin(j)$ is the binary representation of $j$, $S_{\underline{z}} : \{0,1\}^b \to \{0,1\}$ is the characteristic function of $\underline{z}$, and $c_j^i \in \mathbf{N}$ is defined by the requirement that $bin(i) = g_k(bin(c_j^i), bin(j))$. Obviously $h_{k,i}$ can be computed as the AND of two linear threshold gates. Obviously $g_k$ can be computed from $h_k$ with $a$ additional OR-gates.

With a little bit of work one can show that this construction can be implemented by circuits $C_F$ over a fixed graph of depth 4 with $\leq 33 \cdot \frac{2^m}{m}$ edges. In order to get a linear size circuit one adds $2^m/m - m$ dummy inputs to $C_F$ and sets $n := 2^m/m$. ∎

We now turn to the analysis of 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?

We show that for those types of neural nets which were considered in the preceding sections the previously discussed PAC-learning problem has in fact a positive solution:

**Theorem 4.3** *Let $\mathcal{N}$ be an arbitrary neural net as in Theorem 2.1, where the fixed*

*parameters of the piecewise linear activation functions may now be arbitrary reals. Let* $\mathcal{C}_{\mathcal{N}} := \{C \subseteq \boldsymbol{R}^n \mid \exists \underline{\alpha} \in \boldsymbol{R}^w \forall \underline{x} \in \boldsymbol{R}^n \, (\chi_C(\underline{x}) = \mathcal{N}^{\underline{\alpha}}(\underline{x}))\}$ *be the associated concept class, where* $\chi_C$ *is the characteristic function of a concept* $C$*. Then* $\mathcal{C}_{\mathcal{N}}$ *is properly PAC-learnable.*

*This means that there exists a learning algorithm* $LA^{\mathcal{N}}$ *for* $\mathcal{N}$ *such that for any distribution* $Q$ *over* $\boldsymbol{R}^n$*, any target concept* $C_T \in \mathcal{C}_{\mathcal{N}}$*, and any given* $\varepsilon, \delta \in \boldsymbol{R}^+$ *the learning algorithm* $LA^{\mathcal{N}}$ *with inputs* $\varepsilon$ *and* $\delta$ *carries out in* $O\left((\frac{1}{\varepsilon})^{O(1)}, (\frac{1}{\delta})^{O(1)}\right)$ *computations steps (w.r.t. the uniform cost criterion on a RAM) the following task: It computes a suitable number* $m$ *and draws some sequence* $S$ *of* $m$ *examples for* $C_T$ *according to distribution* $Q$*. Then it computes from* $S$ *an assignment* $\underline{\alpha}_S \in \boldsymbol{R}^w$ *for the variable parameters of* $\mathcal{N}$ *such that* $Q\left[\{\underline{x} \in \boldsymbol{R}^n \mid \chi_{C_T}(\underline{x}) \neq \mathcal{N}^{\underline{\alpha}_S}(\underline{x})\}\right] \leq \varepsilon$ *with probability* $\geq 1 - \delta$*.*

**Idea of the proof:** We have VC-dimension $(\mathcal{C}_{\mathcal{N}}) < \infty$ by Remark 3.4. Hence according to [BEHW] it suffices to show that for any given set $S$ of $m$ examples for $C_T$ 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 $\gamma^g$ 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 on of these systems provides for all $\underline{x} \in S$ satisfiable and sufficient constraints for $\underline{\hat{\alpha}}, \underline{c}$. By definition of $\mathcal{C}_{\mathcal{N}}$ we know that there are parameters $\underline{\hat{\alpha}}_T, \underline{c}_T$ such that $\hat{\mathcal{N}}[\underline{c}_T]^{\underline{\hat{\alpha}}_T}$ computes $\chi_{C_T}$. Consider any inequality $\mathrm{I}(\underline{\hat{\alpha}}, \underline{c}, \underline{x}) \leq 0$ (with $\mathrm{I}(\underline{\hat{\alpha}}, \underline{c}, \underline{x})$ linear in $\underline{\hat{\alpha}}, \underline{c}$ for fixed $\underline{x}$, *and* linear in $\underline{x}$ for fixed $\underline{\hat{\alpha}}, \underline{c}$) as they were introduced in the proof of Theorem 2.1 in order to describe the comparison with a threshold at some gate $g$ of $\hat{\mathcal{N}}$. The hyperplane $\{\underline{x} \in \mathbf{R}^n \mid \mathrm{I}(\underline{\hat{\alpha}}_T, \underline{c}_T, \underline{x}) = 0\}$ defines a partition of $S$ into $\{\underline{x} \in S \mid \mathrm{I}(\underline{\hat{\alpha}}_T, \underline{c}, \underline{x}) \leq 0\}$ and $\{\underline{x} \in S \mid \mathrm{I}(\underline{\hat{\alpha}}_T, \underline{c}, \underline{x}) > 0\}$. Hence it suffices to produce (e.g. with the algorithm of [EOS]) in polynomially in $m$ many computation steps all partitions of $S$ that can be generated by as many hyperplanes as there are linear inequalities $\mathrm{I}(\underline{\hat{\alpha}}, \underline{c}, \underline{x}) \leq 0$ in the proof of Theorem 2.1. One of these partitions will agree with the partition of $S$ that is defined by the hyperplanes $\{\underline{x} \in \mathbf{R}^n \mid \mathrm{I}(\underline{\hat{\alpha}}_T, \underline{c}_T, \underline{x}) = 0\}$ for the "correct values" $\underline{\hat{\alpha}}_T, \underline{c}_T$ of the parameters. Each of these partitions corresponds to a "guess" which linear pieces of the activation functions $\gamma^g$ of $\hat{\mathcal{N}}$ are used for the different inputs $\underline{x} \in S$, and hence it defines a unique system of linear inequalities in $\underline{\hat{\alpha}}, \underline{c}$

(with the inputs $\underline{x} \in S$ as fixed coefficients). Furthermore it is guaranteed that one of these "guesses" is correct for $\underline{\hat{\alpha}}_T, \underline{c}_T$.

For each of the resulting polynominally 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 $\underline{\hat{\alpha}}_s, \underline{c}_s$ for which $\hat{\mathcal{N}}[\underline{c}_s]^{\underline{\hat{\alpha}}_s}$ 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. ∎

**Remark 4.4** Assume $\mathcal{N}$ is some arbitrary neural net according to Definition 1.1 with arbitrary piecewise linear activation functions, and $\mathcal{N}$ does not satisfy the condition that all computation nodes of $\mathcal{N}$ have fan-out $\leq 1$. Then Theorem 4.3 does not show that $C_{\mathcal{N}}$ is *properly* PAC-learnable. However it implies that $C_{\mathcal{N}}$ is PAC-learnable, with $\mathcal{C}_{\mathcal{N}'}$ for a somewhat larger neural net $\mathcal{N}'$ of the same depth used as hypothesis class (see Remark 2.2 for the definition of $\mathcal{N}'$).

Note that this result may lead towards a theoretical explanation of an effect that has been observed in many experiments: One often achieves better learning results on artificial neural nets if one uses a neural net with somewhat more units than necessary (i.e. necessary in order to compute the target concept on the neural net).

**Theorem 4.5** *Let $\mathcal{N}$ be an arbitrary neural net with arbitrary piecewise polynomial activation functions and arbitrary polynomial gate inputs $Q^g(y_1, \ldots, y_m)$. 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 neural net $\tilde{\mathcal{N}}$.*

**Idea of the proof:** One can reduce this problem to the case of neural nets with linear gate inputs as indicated in Corollary 3.2. 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 the Appendix. For this neural net $\tilde{\mathcal{N}}$ one can express the constraints on the assignment $\mathcal{A}$ by *linear* inequalities. Remark 3.4 implies that VC-dimension $(\tilde{\mathcal{N}}, \mathbf{R}^n) < \infty$.

One applies the method from the proof of Lemma 2.4 in a manner analogous to the proof of Theorem 4.3 or 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. ∎

# 5 Appendix

In the following we provide details to the *proof of Theorem 3.1.* Because of the somewhat abstract nature of this proof one has to give precise definitions for various auxiliary

notions.

In the preceding sketch of the proof of Theorem 3.1 (after Remark 3.4) we have already described how the given neural nets $\mathcal{N}_n^{\underline{\alpha}_n}$ (with an arbitrary given weight assignment $\underline{\alpha}_n$) can be transformed into neural nets $\hat{\mathcal{N}}_n^{\underline{\hat{\alpha}}_n}$ in normal form. We assume now that $n$ and $\underline{\hat{\alpha}}_n$ have been fixed. We will give in the following a precise definition of a system $L(\hat{\mathcal{N}}_n^{\underline{\hat{\alpha}}_n}, \{0,1\}^n)$ of linear inequalities. This system describes in a weak manner (*weak* because of the required linearity) the role which the currently fixed values $\underline{\hat{\alpha}}_n$ of the parameters of $\hat{\mathcal{N}}_n$ play in the computations of $\hat{\mathcal{N}}_n^{\underline{\hat{\alpha}}_n}$ for arbitrary inputs $\underline{x} \in \{0,1\}^n$. As in the proof of Theorem 2.1, the inputs $\underline{x} \in \{0,1\}^n$ occur as constants in this system of linear inequalities, whereas the variable parameters of $\hat{\mathcal{N}}_n$ (i.e. the weights on edges between input nodes and computation nodes, the biases and thresholds of all gates, and the outer threshold $T_{out}$) will be variables in this system of linear inequalities. We will exploit at the end of this proof (after Lemma 5.6) that this system has *some* solution, because the currently fixed values $\underline{\hat{\alpha}}_n$ provide such a solution.

In order to simplify our notation, we will write $\mathcal{N}$ instead of $\mathcal{N}_n^{\underline{\hat{\alpha}}_n}$ in the following. We assume that some concrete real values (given by $\underline{\hat{\alpha}}_n$) have been assigned to the variable parameters of this neural net $\mathcal{N}$. We define for each gate $g$ in $\mathcal{N}$ by induction on the depth of $g$:

- in Definition 5.1 a set $V^g$ of variables and a set $M^g$ of formal terms that are needed to describe the operation of gate $g$. [The intuition is here that one writes for any circuit input $\underline{x}$ the output of $g$ as a sum of products (of weights, and of components of $\underline{x}$). *Which* of these terms will occur for a specific circuit input $\underline{x}$ will depend on the course of the computation in $\mathcal{N}$ up to gate $g$: for different inputs the involved gates may use different polynomial pieces of their activation function. The set $M^g$ contains a separate formal term for each product that *may* possibly occur in this sum. Each term in $M^g$ consists of a variable $w \in V^g$ (that represents a weight, a bias, a constant gate-output, or some product of weights and biases) and of a product $P \equiv \pm \boldsymbol{x}_1^{j_1} \cdot \ldots \cdot \boldsymbol{x}_n^{j_n}$ of input *variables* $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n$.]
- in Definition 5.2 for any fixed circuit input $\underline{x} \in \mathbf{R}^n$ a set $L^g(\underline{x})$ of linear inequalities associated with gate $g$, that hold for the computation of $\mathcal{N}$ on input $\underline{x}$ if all formal terms $t \in M^g$ are replaced by their actual value $W(t, \underline{x})$ in $\mathcal{N}$; we also define in Definition 5.2 a set $S^g(\underline{x})$ of formal terms whose sum represents the input of $g$, and a set $T^g(\underline{x}) \subseteq M^g$ of formal terms that represents the output of $g$ for circuit input $\underline{x}$. [$L^g(\underline{x})$ specifies in particular which piece of $\gamma^g$ is used by gate $g$ for circuit input $\underline{x}$.]
- in Definition 5.4 for any input set $S \subseteq \mathbf{R}^n$, any solution $\mathcal{A}$ of the resulting system $L(\mathcal{N}, S) := \bigcup \{L^g(\underline{x}) \mid \underline{x} \in S \text{ and } g \text{ is a gate in } \mathcal{N}\}$ of linear inequalities, and any term $t \in M^g$ a circuit $C^{\mathcal{A},g}(t)$ that decides for any circuit input $\underline{x} \in S$ whether $t$ is present in the output of $g$ in $\mathcal{N}$. [For any input $\underline{x} \in S$ the circuits $(C^{\mathcal{A},g}(t))_{t \in M^g}$ together compute the characteristic function of the set $T^g(\underline{x}) \subseteq M^g$. In this way one can replace in a recursive manner the analog computations in $\mathcal{N}$ by digital manipulations of formal terms, with "nonstandard products" in place of real products.]

One verifies in Lemma 5.3 that $L(\mathcal{N}, S)$ describes correctly the role of the parameters $\underline{\hat{\alpha}}_n$ in the computations of $\mathcal{N} := \hat{\mathcal{N}}_n^{\underline{\hat{\alpha}}_n}$ for inputs $\underline{x} \in S$. Unfortunately $L(\mathcal{N}, S)$ does not provide a *complete* description of the properties of the parameters $\underline{\hat{\alpha}}_n$ in these computations, since it represents only a "linear fragment" of their theory. Nevertheless one can prove with the help of Lemma 5.5 and Lemma 5.6 that for *any* solution $\mathcal{A}$ of $L(\mathcal{N}, S)$ the digital circuits $C^{\mathcal{A},g}(t)$ carry out a truthful simulation of the corresponding subcircuits of $\mathcal{N}$.

We would like to point out a difference to the proof of Theorem 2.1 regarding the treatment of parameters. In the proof of Theorem 3.1 the variable parameters $\underline{\alpha}_n$ of $\mathcal{N}_n^{\underline{\alpha}_n}$ (the weights on edges, the biases of gates, and the outer threshold $T_{out}$) *and the fixed parameters* of the given neural nets $\mathcal{N}_n$ (the thresholds of activation functions $\gamma^g$, and the coefficients of the polynomial pieces of $\gamma^g$) are all changed simultaneously in the transformation to $\hat{\mathcal{N}}_n^{\underline{\hat{\alpha}}_n}$. Consequently $\underline{\hat{\alpha}}_n$ denotes the values of *all* nontrivial parameters in $\hat{\mathcal{N}}_n^{\underline{\hat{\alpha}}_n}$ (the weights on edges between input nodes and computation nodes, the biases of all gates, the outer threshold, *and also the thresholds* $t_1^g, t_2^g$ *and the values of constant outputs* $a^g$ of all gates $g$ in $\hat{\mathcal{N}}_n^{\underline{\hat{\alpha}}_n}$). As a consequence of this treatment of parameters one can allow in the given neural nets $\mathcal{N}_n$ of Theorem 3.1 *arbitrary reals* as fixed parameters (i.e. for the thresholds and coefficients of the polynomial pieces of the given activation functions $\gamma^g$).

We refer to an analog neural net $\mathcal{N}$ with the properties of $\hat{\mathcal{N}}_n^{\underline{\hat{\alpha}}_n}$ (as described in section 3) as a neural net in *normal form*. This means that all gates in $\mathcal{N}$ have fan-out $\leq 1$, all weights on edges *between two computation nodes* are from $\{-1, 0, 1\}$, all gates $g$ in $\mathcal{N}$ use as activation function $\gamma^g$ a piecewise polynomial function that consists of 3 pieces, of which at most one piece is not identically 0, and in which the nontrivial piece (if it exists) outputs a constant from $\mathbf{R}$ or computes a power $y \mapsto y^k$ for some $k \in \mathbf{N}$. The argument of $\gamma^g$ is as usual a weighted sum $\sum_{i=1}^{m} \alpha_i^g y_i + \alpha_0^g$, where $y_1, \ldots, y_m \in \mathbf{R}$ are the inputs to gate $g$.

In order to simplify notation, we assume that for a neural net $\mathcal{N}$ in normal form the nontrivial piece of the activation function $\gamma^g$ of each gate $g$ is defined over a half-open interval $[t_1^g, t_2^g)$ with certain reals $t_1^g < t_2^g$. It is easy to see that the subsequent proof can also be carried out without this simplifying assumption. We also assume w.l.o.g. that $\mathcal{N}$ is levelled, i.e. each gate $g$ in $\mathcal{N}$ has the property that all paths in $\mathcal{N}$ from an input node to $g$ have the same length.

**Definition 5.1** *Assume that* $\mathcal{N}$ *is a neural net in normal form with* $n$ *input variables* $x_1, \ldots, x_n$*, where arbitrary reals have been assigned to all parameters of* $\mathcal{N}$*. We define by induction on the depth of* $g$ *for each gate* $g$ *in* $\mathcal{N}$ *a set* $V^g$ *of variables, a value* $W(v)$ *for each variable* $v \in V^g$*, and a set* $M^g$ *of (formal) terms. Each element of* $M^g$ *is of the form* $v \cdot P$*, where* $v \in V^g$ *is a variable and* $P$ *is some formal polynomial term of the form* $\pm\, \boldsymbol{x}_1^{j_1} \cdot \ldots \cdot \boldsymbol{x}_n^{j_n}$*, with* $j_1, \ldots, j_n \in \boldsymbol{N}$*. The here occuring* formal variables $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n$ *for the input components should be distinguished from the* concrete values $x_1, \ldots, x_n \in \boldsymbol{R}$ *for these variables that are considered later (starting in Definition 5.2). For any set* $M$

*of terms we set*

$$\begin{aligned} 1 \cdot M &:= M \\ (-1) \cdot M &:= \{w \cdot (-P) \mid w \cdot P \in M, \text{ where } w \text{ is a variable}\} \\ 0 \cdot M &:= \phi \, . \end{aligned}$$

*Hence we can also talk about sets of terms $\alpha_i^g \cdot M$, where $\alpha_i^g$ is a weight on an edge between computation nodes in $\mathcal{N}$ ($\alpha_i^g \in \{-1, 0, 1\}$, since $\mathcal{N}$ is assumed to be in normal form).*

*We consider first the case where $g$ has depth 1. If $\gamma^g$ gives on its nontrivial piece $[t_1^g, t_2^g)$ a constant $a^g$ as output, we set*

$$V^g := \{v_0^g, \ldots, v_n^g\} \cup \{v_{const}^g\} \cup \{v_I^g, v_{II}^g\} \text{ and } M^g := \{v_{const}^g\}.$$

*We define $W(v_i^g) := \alpha_i^g$ for $i = 0, \ldots, n$, $W(v_{const}^g) := a^g, W(v_I^g) := t_1^g$, and $W(v_{II}^g) := t_2^g$ ($\alpha_1^g, \ldots, \alpha_n^g$ are the weights and $\alpha_0^g$ is the bias of gate $g$ in $\mathcal{N}$; $t_1^g, t_2^g$ are the thresholds of the activation function $\gamma^g$). In the other case $\gamma^g$ computes a power $y \mapsto y^k$ on its nontrivial piece. Then we introduce for each $k$-tuple $< w_1 \cdot P_1, \ldots, w_k \cdot P_k > \in (\{v_0^g\} \cup \{v_1^g \cdot x_1, \ldots, v_n^g \cdot x_n\})^k$ a new variable $v_{w_1,\ldots,w_k}^g$ in $V^g$ and a term $v_{w_1,\ldots,w_k}^g \cdot \prod_{i=1}^k P_i$ in $M^g$. We assume here that a formal multiplication $P \cdot P'$ for formal terms $P, P'$ of the form $\pm \boldsymbol{x}_1^{j_1} \cdot \ldots \cdot \boldsymbol{x}_n^{j_n}$ is defined in the obvious way. Thus we get*

$$V^g := \{v_0^g, \ldots, v_n^g\} \cup \{v_I^g, v_{II}^g\} \cup \{v_{w_1,\ldots,w_k}^g \mid < w_1, \ldots, w_k > \in \{v_0^g, \ldots, v_n^g\}^k\} \, .$$

*We set $W(v_{w_1,\ldots,w_k}^g) := \prod_{i=1}^k W(w_i)$, and we define $W(v)$ for the other variables as before. We define*

$$M^g := \{v_{w_1,\ldots,w_k}^g \cdot \prod_{i=1}^k P_i \mid < w_1 \cdot P_1, \ldots, w_k \cdot P_k > \in (\{v_0^g\} \cup \{v_1^g \cdot \boldsymbol{x}_1, \ldots, v_n^g \cdot \boldsymbol{x}_n\})^k\} \, .$$

[The terms in $M^g$ play the role of the summands that one gets from the output $(\alpha_0^g + \sum_{i=1}^m \alpha_i^g \cdot x_i)^k$ of $\gamma^g$ by rewriting this output as a sum of products.]

*We now consider the case where $g$ is a gate on level $l+1$, with edges from the gates $g_1, \ldots, g_m$ on level $l$ leading into $g$. Assume that $\alpha_1^g, \ldots, \alpha_n^g \in \{-1, 0, 1\}$ are the weights on these edges in $\mathcal{N}$ and that $\alpha_0^g$ is the bias of $g$. If $\gamma^g$ outputs a constant $a^g$ on its nontrivial piece, we set*

$$V^g := \{v_0^g, v_{const}^g\} \cup \{v_I^g, v_{II}^g\} \text{ and } M^g := \{v_{const}^g\}.$$

*We set $W(v_0^g) := \alpha_0^g, W(v_{const}^g) := a^g, W(v_I^g) := t_1^g$, and $W(v_{II}^g) := t_2^g$. If $\gamma^g$ computes the power $y \mapsto y^k$ on its nontrivial piece, we introduce for each $k$-tuple*

$$< w_1 \cdot P_1, \ldots, w_k \cdot P_k > \in (\{v_0^g\} \cup \bigcup_{j=1}^m \alpha_j^g \cdot M^{g_j})^k$$

*a new variable $v_{w_1,\ldots,w_k}^g$ in $V^g$, and a term $v_{w_1,\ldots,w_k}^g \cdot \prod_{i=1}^k P_i$ in $M^g$. Thus we set*

$$V^g := \{v_0^g\} \cup \{v_I^g, v_{II}^g\} \cup$$
$$\{v^g_{w_1,\ldots,w_k} \mid < w_1 \cdot P_1, \ldots, w_k \cdot P_k > \in (\{v_0^g\} \cup \bigcup_{j=1}^{m} \alpha_j^g \cdot M^{g_j})^k,$$
$$\textit{for arbitrary polynomial terms } P_1, \ldots, P_k$$
$$\textit{and variables } w_i \in (\{v_0^g\} \cup \bigcup_{j=1}^{m} V^{g_j})\}.$$

*We define* $W(v^g_{w_1,\ldots,w_k}) := \prod_{i=1}^{k} W(w_i)$, $W(v_0^g) := \alpha_0^g$, $W(v_I^g) := t_1^g$, $W(v_{II}^g) := t_2^g$. *We set* $M^g := \{v^g_{w_1,\ldots,w_k} \cdot \prod_{i=1}^{k} P_i \mid < w_1 \cdot P_1, \ldots, w_k \cdot P_k > \in (\{v_0^g\} \cup \bigcup_{j=1}^{m} \alpha_j^g \cdot M^{g_j})^k\}$. [The argument of $\gamma^g$ is a sum of $\alpha_0^g$ and of summands that are denoted by terms in $\bigcup_{j=1}^{m} \alpha_j^g \cdot M^{g_j}$. Hence the terms in $M^g$ correspond to the summands that one gets by rewriting the output of $\gamma^g$ as a sum of products.]

*Finally, for the output gate $g_{out}$ of $\mathcal{N}$, we place into $V^g$ in addition the variable $v^{g_{out}}$. We define $W(v^{g_{out}})$ as the value of the outer threshold of $\mathcal{N}$.*

**Definition 5.2** *Assume that $\mathcal{N}$ is a neural net in normal form with $n$ input variables. Let $\underline{x} \in \boldsymbol{R}^n$ be a fixed input for $\mathcal{N}$. We define for each gate $g$ in $\mathcal{N}$ by simultaneous induction on the depth of $g$*

- *a set $L^g(\underline{x})$ of inequalities (that are linear in the variables from $V^g$)*
- *a set $S^g(\underline{x})$ of formal terms (whose sum represents the argument of $\gamma^g$ for circuit input $\underline{x}$)*
- *a set $T^g(\underline{x}) \subseteq M^g$ (whose sum represents the output of $g$ for circuit input $\underline{x}$).*

*Since $\underline{x}$ is now a fixed element of $\boldsymbol{R}^n$, one can assign a specific value $W(P, \underline{x}) \in \boldsymbol{R}$ to each term $P$ of the form $\pm\, \boldsymbol{x}_1^{j_1} \cdot \ldots \cdot \boldsymbol{x}_n^{j_n}$ that occurs in a formal term of the preceding definition. Hence one can assign to any formal term $t = v \cdot P$ (that belongs to some $M^g$) a specific value $W(t, \underline{x}) := W(v) \cdot W(P, \underline{x})$. For a set $S$ of formal terms we define $W(S, \underline{x}) := \sum_{t \in S} W(t, \underline{x})$. For the case $S = \phi$ we set $W(\phi, \underline{x}) := 0$.*

*If $g$ has depth 1, then we define*

$$S^g(\underline{x}) := \{v_0^g\} \cup \{v_i^g \cdot \boldsymbol{x}_i \mid i = 1, \ldots, n\}.$$

*Assume that $g$ is a gate on level $l+1$ with edges from gates $g_1, \ldots, g_m$ on level $l$ leading into $g$. Let $\alpha_0^g, \alpha_1^g, \ldots, \alpha_m^g$ be the bias and the weights of gate $g$ in $\mathcal{N}$. Then we set*

$$S^g(\underline{x}) := \{v_0^g\} \cup \bigcup_{j=1}^{m} \alpha_j^g \cdot T^{g_j}(\underline{x}).$$

*We define $L^g(\underline{x})$ and $T^g(\underline{x})$ as follows for any gate $g$ in $\mathcal{N}$. If $W(S^g(\underline{x}),\underline{x}) < t_1^g$, then $L^g(\underline{x})$ contains the inequality $\sum S^g(\underline{x}) + 1 \leq v_I^g$. If $W(S^g(\underline{x}),\underline{x}) \geq t_2^g$, then $L^g(\underline{x})$ contains the inequality $\sum S^g(\underline{x}) \geq v_{II}^g$. In either case we set $T^g(\underline{x}) := \emptyset$.*

*If $t_1^g \leq W(S^g(\underline{x}),\underline{x}) < t_2^g$, then $L^g(\underline{x})$ contains the inequalities $v_I^g \leq \sum S^g(\underline{x})$ and $\sum S^g(\underline{x}) + 1 \leq v_{II}^g$. If $\gamma^g$ gives on its nontrivial piece a constant $a^g$ as output, we set in this case $T^g(\underline{x}) := \{v_{const}^g\}$. If $\gamma^g$ computes on its nontrivial piece a power $y \mapsto y^k$, we set in this case $T^g(\underline{x})$ :=*

$$\{v^g_{w_1,\ldots,w_k} \cdot \prod_{i=1}^{k} P_i \mid \ < w_1 \cdot P_1, \ldots, w_k \cdot P_k > \ \in \ (\{v_0^g\} \ \cup \ \bigcup_{j=1}^{m} \alpha_j^g \cdot T^{g_j}(\underline{x}))\}.$$

*Finally, if $g$ is the output gate $g_{out}$ of $\mathcal{N}$ and $W(T^{g_{out}}(\underline{x}),\underline{x}) < W(v^{g_{out}})$, we add to $L^{g_{out}}(\underline{x})$ also the inequality $\sum T^{g_{out}}(\underline{x}) + 1 \leq v^{g_{out}}$. If $W(T^{g_{out}}(\underline{x}),\underline{x}) \geq W(v^{g_{out}})$, we add to $L^{g_{out}}(\underline{x})$ also the inequality $\sum T^{g_{out}}(\underline{x}) \geq v^{g_{out}}$.*

*We define*

$$L(\mathcal{N},\underline{x}) \ := \ \bigcup\{L^g(\underline{x}) \mid g \text{ is a gate of } \mathcal{N}\}\,,$$

*and for $S \subseteq \boldsymbol{R}^n$*

$$L(\mathcal{N},S) \ := \ \bigcup\{L(\mathcal{N},\underline{x}) \mid \underline{x} \in S\}\,.$$

The following Lemma verifies that for any $\underline{x} \in \mathbf{R}^n$ the system $L(\mathcal{N},\underline{x})$ of inequalities provides a truthful description of the computation of $\mathcal{N}$ for input $\underline{x}$.

**Lemma 5.3** *Assume that $\mathcal{N}$ is a neural net in normal form with $n$ input variables, and $\underline{x} \in \boldsymbol{R}^n$ is an arbitrary concrete input.*

*Then we have for any gate $g$ in $\mathcal{N}$:*
*$W(S^g(\underline{x}),\underline{x})$ is the input, and $W(T^g(\underline{x}),\underline{x})$ is the output of gate $g$ in the computation of $\mathcal{N}$ for input $\underline{x}$. Furthermore:*

$$\begin{array}{lcl}
W(S^g(\underline{x}),\underline{x}) < t_1^g & \Leftrightarrow & \text{“}S^g(\underline{x}) + 1 \ \leq \ v_I^g\text{”} \in L(\mathcal{N},\underline{x}) \\
W(S^g(\underline{x}),\underline{x}) < t_2^g & \Leftrightarrow & \text{“}S^g(\underline{x}) + 1 \ \leq \ v_{II}^g\text{”} \in L(\mathcal{N},\underline{x}) \\
W(S^g(\underline{x}),\underline{x}) \geq t_1^g & \Leftrightarrow & \text{“}S^g(\underline{x}) \ \geq \ v_I^g\text{”} \in L(\mathcal{N},\underline{x}) \\
W(S^g(\underline{x}),\underline{x}) \geq t_2^g & \Leftrightarrow & \text{“}S^g(\underline{x}) \ \geq \ v_{II}^g\text{”} \in L(\mathcal{N},\underline{x})\,.
\end{array}$$

**Proof:** Induction on the depth of $g$. ∎

**Definition 5.4** *Assume that $\mathcal{N}$ is a neural net of depth $d$ in normal form with $n$ inputs. Assume that $q$ is the maximal degree of a piecewise polynomial activation function in $\mathcal{N}$. Furthermore assume that $S \subseteq \boldsymbol{R}^n$ and $\mathcal{A} : V^{\mathcal{N}} \to \boldsymbol{R}$ is an arbitrary solution of the system $L(\mathcal{N},S)$ of inequalities with the variable set $V^{\mathcal{N}} := \bigcup\{V^g \mid g \text{ is a gate in } \mathcal{N}\}$.*

*We define by induction on the depth of gate $g$ in $\mathcal{N}$ for each term $t \in M^g$ a circuit $C^{\mathcal{A},g}(t)$. Together the circuits $(C^{\mathcal{A},g}(t))_{t\in M^g}$ mimic the subcircuit of $\mathcal{N}$ between the input*

*and gate $g$. The circuit $C^{\mathcal{A},g}(t)$ consists of linear threshold gates and multiplication gates of fan-in 2 that compute the product of any two real numbers. For any circuit input $\underline{x} \in S$ the output of the circuit $C^{\mathcal{A},g}(t)$ will be 1 if $t \in T^g(\underline{x})$, otherwise it will be 0.*

*One associates with each circuit $C^{\mathcal{A},g}(t)$ for $t \in M^g$ of the form $t \equiv v \cdot P$ another circuit $\tilde{C}^{\mathcal{A},g}(t)$ that outputs for any circuit input $\underline{x} \in S$ the real number*

$$\mathcal{A}(t,\underline{x}) \;:=\; \begin{cases} \mathcal{A}(v)\cdot\mathcal{A}(P,\underline{x}) & , \;\; \text{if } t \in T^g(\underline{x}) \\ 0 & , \;\; \text{if } t \in M^g - T^g(\underline{x}) \,, \end{cases}$$

*where one defines for each formal term $P \equiv \pm\, \boldsymbol{x}_1^{j_1}\cdot\ldots\cdot\boldsymbol{x}_n^{j_n}$ a value $\mathcal{A}(P,\underline{x}) \in \boldsymbol{R}$ as $\pm\, x_1^{j_1}\cdot\ldots\cdot x_n^{j_n}$ (i.e. formal variables $\boldsymbol{x}_i$ are replaced by the corresponding components of $\underline{x} \in S$). The extension from $C^{\mathcal{A},g}(t)$ to $\tilde{C}^{\mathcal{A},g}(t)$ is done in a canonical manner by adding $\leq q^d + 1$ product gates (we assume for the moment that the constants $\mathcal{A}(v)$ are available in analog form as separate inputs to the circuits).*

*The definition of a value $\mathcal{A}(t,\underline{x})$ for each term $t$ and each $\underline{x} \in S$ is extended in a canonical way to arbitrary sets $M$ of terms:*

$$\mathcal{A}(M,\underline{x}) := \sum_{t\in M}\mathcal{A}(t,\underline{x}), \quad \mathcal{A}(\phi,\underline{x}) \;:=\; 0 \;.$$

*We consider first the case where $g$ has depth 1. Let $H_1^g$ be a linear threshold gate that checks whether $\mathcal{A}(v_I^g) \leq \mathcal{A}(S^g(\underline{x}),\underline{x})$, and let $H_2^g$ be a linear threshold gate that checks whether $\mathcal{A}(S^g(\underline{x}),\underline{x}) + 1 \leq \mathcal{A}(v_{II}^g)$. For each term $t \in M^g$ we define $C^{\mathcal{A},g}(t)$ to be the AND of $H_1^g$ and $H_2^g$.*

*Assume then that $g$ is a gate on level $l+1$ with edges from the gates $g_1,\ldots,g_m$ on level $l$ leading into $g$. According to Definition 5.2 we have in this case $S^g(\underline{x}) = \{v_0^g\} \cup \bigcup_{j=1}^{m} \alpha_j^g \cdot T^{g_j}(\underline{x})$ for every $\underline{x} \in S$. By induction hypothesis we have already defined circuits $C^{\mathcal{A},g_j}(t)$, and hence also circuits $\tilde{C}^{\mathcal{A},g_j}(t)$ for all $t \in M^{g_j}$, $j = 1,\ldots,m$. For each term $t \in M^g$ the circuit $C^{\mathcal{A},g}(t)$ employs two linear threshold gates $H_1^g$ and $H_2^g$, which receive their inputs from the circuits $\tilde{C}^{\mathcal{A},g_j}(t)$ for $t \in M^{g_j}$, $j = 1,\ldots,m$. Their weights are from $\{-1,0,1\}$ as given by $\alpha_1^g,\ldots,\alpha_m^g$. The gate $H_1^g$ checks whether $\mathcal{A}(v_I^g) \leq \mathcal{A}(S^g(\underline{x}),\underline{x})$ and $H_2^g$ checks whether $\mathcal{A}(S^g(\underline{x}),\underline{x}) + 1 \leq \mathcal{A}(v_{II}^g)$.*

*If $\gamma^g$ outputs a constant on its nontrivial piece, $C^{\mathcal{A},g}(v_{const}^g)$ is defined as the AND of $H_1^g$ and $H_2^g$ (together with their subcircuits $\tilde{C}^{\mathcal{A},g_j}(t)$).*

*If $\gamma^g$ computes $y \mapsto y^k$ on its nontrivial piece, each $t \in M^g$ is of the form $v_{w_1,\ldots,w_k}^g \cdot \prod_{i=1}^{k} P_i$ for some $k$-tuple $< w_1\cdot P_1,\ldots,w_k\cdot P_k > \in \;(\{v_0^g\} \cup \bigcup_{j=1}^{m}\alpha_j^g\cdot M^{g_j})^k$. In this case $C^{\mathcal{A},g}(t)$ is defined as the AND of $H_1^g, H_2^g$, and of the outputs of the circuits $C^{\mathcal{A},g_j}(w_i\cdot(\alpha_j^g\cdot P_i))$ for all $i \in \{1,\ldots,k\}$ and $j \in \{1,\ldots,m\}$ with $\alpha_j^g \neq 0$ and $w_i\cdot P_i \in \alpha_j^g\cdot M^{g_j}$.* [One should recall that $\alpha_j^g \in \{-1,0,1\}$ since $\mathcal{N}$ is in normal form. Hence for $\alpha_j^g \neq 0$ one has $w_i\cdot P_i \in \alpha_j^g\cdot M^{g_j} \Leftrightarrow w_i\cdot(\alpha_j^g\cdot P_i) \in M^{g_j}$. One should also keep in mind that in general $\mathcal{A}(v_{w_1,\ldots,w_k}^g) \neq \prod_{i=1}^{k}\mathcal{A}(w_i)$].

*Finally we define the circuit $C^{\mathcal{A}}(\mathcal{N})$ by using as subcircuits the circuits $\tilde{C}^{\mathcal{A},g_{out}}(t)$ for all $t \in M^{g_{out}}$. The output of $C^{\mathcal{A}}(\mathcal{N})$ is given by a linear threshold gate $H$ that checks whether $\mathcal{A}(M^{g_{out}}, \underline{x}) \geq \mathcal{A}(v^{g_{out}})$.*

**Lemma 5.5** *Assume that $S \subseteq \boldsymbol{R}^n$ and $\mathcal{A}$ is an arbitrary solution of $L(\mathcal{N}, S)$. Then the following holds for any gate $g$ in $\mathcal{N}$, for any term $t \in M^g$, and any input $\underline{x} \in S$:*

*a) The output of the gate $H_1^g$ in $C^{\mathcal{A},g}(t)$ is 1 if and only if $L(\mathcal{N}, S)$ contains the inequality "$v_I^g \leq \sum S^g(\underline{x})$". Similarly the output of the gate $H_2^g$ in $C^{\mathcal{A},g}(t)$ is 1 if and only if $L(\mathcal{N}, S)$ contains the inequality "$\sum S^g(\underline{x}) + 1 \leq v_{II}^g$".*

*b) $t \in T^g(\underline{x}) \Leftrightarrow$ ($C^{\mathcal{A},g}(t)$ outputs 1 for circuit input $\underline{x}$).*

**Proof:** By induction on the depth of $g$. ∎

**Lemma 5.6** *Assume that $\mathcal{N}$ is a neural net in normal form with $n$ input variables, $S \subseteq \boldsymbol{R}^n$ is an arbitrary set of inputs, and $\mathcal{A}$ is an arbitrary solution of $L(\mathcal{N}, S)$. Then $\mathcal{N}$ and $C^{\mathcal{A}}(\mathcal{N})$ compute the same function form $S$ into $\{0,1\}$.*

**Proof:** Follows from Lemma 5.3 and Lemma 5.5. ∎

We are now in a position where we can complete the *proof of Theorem 3.1.* Assume that a given array $(\mathcal{N}_n)_{n\in\mathbf{N}}$ of neural nets satisfies the assumption of Theorem 3.1, and that $(\underline{\alpha}_n)_{n\in\mathbf{N}}$ is an arbitrary array of real valued assignments $\underline{\alpha}_n$ to the variable parameters in $\mathcal{N}_n$. One can transform the given neural nets $(\mathcal{N}_n^{\underline{\alpha}_n})_{n\in\mathbf{N}}$ into an array $(\hat{\mathcal{N}}_n^{\hat{\underline{\alpha}}_n})_{n\in\mathbf{N}}$ of neural nets in normal form (with properties as specified in section 3) such that $\hat{\mathcal{N}}_n^{\hat{\underline{\alpha}}_n}$ computes the same boolean function as $\mathcal{N}_n^{\underline{\alpha}_n}$. We then apply the machinery from the definition and Lemmas 5.1 to 5.6 to each neural net $\mathcal{N} := \hat{\mathcal{N}}_n^{\hat{\underline{\alpha}}_n}$ with $S := \{0,1\}^n$. By construction of $\hat{\mathcal{N}}_n^{\underline{\alpha}_n}$ the resulting system $L(\mathcal{N}, \{0,1\}^n)$ of inequalities has some solution over $\mathbf{R}$. We exploit here in particular that $\hat{\underline{\alpha}}_n$ was chosen so that all relevant strict inequalities "$s_1 < s_2$" in computations of $\hat{\mathcal{N}}_n^{\hat{\underline{\alpha}}_n}$ on inputs $\underline{x} \in \{0,1\}^n$ were strengthened to "$s_1 + 1 \leq s_2$". Since $|\bigcup\{M^g \mid g \text{ gate in } \hat{\mathcal{N}}_n^{\hat{\underline{\alpha}}_n}\}| = O(n^{O(1)})$, it follows that the number of gates in $C^{\mathcal{A}}(\hat{\mathcal{N}}_n^{\hat{\underline{\alpha}}_n})$ is bounded by $O(n^{O(1)})$. The number of variables in $L(\mathcal{N}, \{0,1\}^n)$ is polynomial in $n$ and it only contains constants from $\{-1, 0, 1\}$. Hence by Lemma 2.4 there is a solution $\mathcal{A}$ of $L(\mathcal{N}, \{0,1\}^n)$ that consists of rationals of the form $\frac{s}{t}$ (with a common integer $t$) such that $s$ and $t$ are integers of size $2^{O(n^{O(1)})}$. By Lemma 5.6 the associated circuit $C^{\mathcal{A}}(\mathcal{N})$ computes the same boolean function as $\mathcal{N}$. Furthermore all constants and parameters in $C^{\mathcal{A}}(\mathcal{N})$ are from $\{-1, 0, 1\}$ or are given by $\mathcal{A}$. Hence they are quotients of integers with polynomially in $n$ many bits. Thus (see [SBKH], [SR]) one can carry out all arithmetical operations in $C^{\mathcal{A}}(\mathcal{N})$ by polynomial size digital subcircuits of constant depth with linear threshold gates (or equivalently: with MAJORITY-gates, see [CSV]). In the resulting circuit all parameters from $\mathcal{A}$ are replaced by corresponding sequences of bits. Hence one gets in this way neural nets $\tilde{\mathcal{N}}_n$ which satisfy the claim of Theorem 3.1. ∎

### Acknowledgements

We would like to thank Eduardo D. Sontag for drawing our attention to the problem of finding upper bounds for neural nets with $\pi$-gates, and for his insightful comments. We thank 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 multi-layer 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

[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.* 1992

[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", *Tech Report* UCSC-CRL - 91-02 (1989)

[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

[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

[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

[M] 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, 1988)

[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