Agnostic PAC-Learning of Functions on Analog Neural Nets

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

Abstract

We consider learning on multi-layer neural nets with piecewise polynomial activation functions and a fixed number k of analog inputs. We exhibit arbitrarily powerful network architectures for which efficient and provably successful learning algorithms exist in the rather realistic refinement of Valiant's model for probably approximately correct learning ("PAC-learning") where no a-priori assumptions are required about the "target function" (agnostic learning), arbitrary noise is permitted in the training sample, and the target outputs as well as the network outputs may be arbitrary reals. The number of computation steps of the learning algorithm LEARN that we construct is bounded by a polynomial in the bit-length n of the fixed number of input variables, in the bound s for the allowed bit-length of weights, and in $\frac{1}{\varepsilon}$, where ε is some arbitrary given bound for the true error of the neural net after training and for the probability that the learning algorithm fails for a randomly drawn training sample.

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

1 Introduction

The investigation of learning on multi-layer feedforward neural nets has become a large and fruitful research area. It would be desirable to develop also an adequate *theory* of learning on neural nets that helps us to understand and predict the outcomes of experiments. The most commonly considered theoretical framework for learning on neural nets is Valiant's model [V] for probably approximately correct learning ("PAC-learning"). In this model one can analyze both the required number of training examples (the "sample complexity") and the required number of computation steps for learning on neural nets.

With regard to sample complexity the theoretical investigation of PAC-learning on neural nets has been rather successful. It has lead to the discovery of an essential mathematical parameter of each neural net $\mathcal{N}$: the Vapnik - Chervonenkis dimension of $\mathcal{N}$, commonly referred to as the VC-dimension of $\mathcal{N}$. The VC-dimension of $\mathcal{N}$ determines the number of randomly drawn training examples that are needed in the PAC-model to train $\mathcal{N}$ ([BEHW]). It has been shown that the VC-dimension of any feedforward neural net $\mathcal{N}$ with linear threshold gates and w weights can be bounded by $O(w \log w)$ (Cover [C], Baum and Haussler [BH]). Recently it has also been shown that this upper bound is optimal in the sense that there are arbitrarily large neural nets $\mathcal{N}$ with w weights whose VC-dimension is bounded from below by $\Omega(w \log w)$ [M 93a]. Since the PAC-model is a worst case model with regard to the choice of the distribution on the examples, it predicts bounds for the sample complexity that tend to be somewhat too large in comparison with experimental results.

The quoted upper bound for the VC-dimension of a neural net implies that the sample complexity provides no obstacle for efficient (i.e. polynomial time) learning on neural nets in Valiant's PAC-model. However a number of negative results due to Judd [J], Blum and Rivest [BR], Kearns and Valiant [KV] show that even for arrays $(\mathcal{N}_n)_{n\in\mathbf{N}}$ of very simple multi-layer feedforward neural nets (where the number of nodes in $\mathcal{N}_n$ is polynomially related to the parameter n) in the PAC - model there are no learning algorithms for $\mathcal{N}_n$ whose number of computation steps can be bounded by a polynomial in n. Although these negative results are based on unproven conjectures from computational complexity theory such as $NP \neq RP$, they have effectively halted the further theoretical investigation of learning algorithms for multi-layer neural nets within the framework of the PAC - model.

A closer look shows that the type of asymptotic analysis that has been carried out for these negative results is not the only one possible. In fact, a different kind of asymptotic analysis appears to be more adequate for a theoretical analysis of learning on relatively small neural nets with analog inputs. We propose to investigate PAC-learning on a *fixed* neural net $\mathcal{N}$, with a fixed number k of analog inputs (for example k sensory data). The asymptotic question that we consider is whether $\mathcal{N}$ can learn any target function with arbitrary precision if sufficiently many randomly drawn training examples are provided. More precisely we consider the question whether there exists an efficient learning algorithm for $\mathcal{N}$ whose number of computation steps can be bounded by a polynomial in the bit-length n of the k analog inputs, a bound s for the allowed bit-length of weights, and $\frac{1}{\varepsilon}$, where ε is an arbitrary given bound for the true error of $\mathcal{N}$ after the training (and a bound for the probability that the training fails for a randomly drawn sample).

In this paper, we simultaneously turn to a more realistic refinement of the PAC-model which is essentially due to Haussler [H] and which was further developed by Kearns, Schapire and Sellie [KSS]. This refinement of the PAC-model is more adequate for the analysis of learning on neural nets, since it requires no unrealistic a-priori assumptions about the nature of the "target concept" or "target function" that the neural net is supposed to learn ("agnostic learning"), and it allows for arbitrary noise in the sample. Furthermore it allows us to consider situations where both the target outputs in the sample and the actual outputs of the neural net are arbitrary real numbers (instead of boolean values). Hence in contrast to the regular PAC-model we can investigate in this more flexible framework also the learning (resp. approximation) of complicated real valued functions by a neural net.

We will give at the end of this section in Definition 1.1 and Definition 1.2 a precise definition of the type of neural network models that we consider in this paper: high order multi-layer feedforward neural nets with piecewise polynomial activation functions.

We will give in Definition 2.2 in section 2 a precise definition of the refinement of the PAC-learning model that we consider in this paper. We will show in Proposition 2.5 that, even in the stronger version of PAC-learning considered here, the required number of training examples provides no obstacle to efficient learning. This is demonstrated by giving an upper bound for the pseudo-dimension $\dim_P(\mathcal{F})$ of the associated function class $\mathcal{F}$. It was previously shown by Haussler [H] that for the learning of classes of functions with non-binary outputs the pseudo-dimension plays a role which is similar to the role of the VC-dimension for the learning of concepts.

We will prove in Theorem 2.1 that for arbitrarily complex first order neural nets $\tilde{\mathcal{N}}$ with piecewise linear activation functions there exists an efficient and provably successful learning algorithm for $\tilde{\mathcal{N}}$. This positive result is extended to high order neural nets with piecewise polynomial activation functions in Theorem 3.1.

One should note that these results do not show that there exists an efficient learning algorithm for *every* neural net. Rather they exhibit a special class of neural nets $\tilde{\mathcal{N}}$ for which there exist efficient learning algorithms. This special class of neural nets $\tilde{\mathcal{N}}$ is "universal" in the sense that there exists for every high order neural net $\mathcal{N}$ with piecewise polynomial activation functions a somewhat larger neural net $\tilde{\mathcal{N}}$ in this class such that every function computable on $\mathcal{N}$ is also computable on $\tilde{\mathcal{N}}$. Hence our positive results about efficient and provably successful learning on neural nets can in principle be applied to real-life learning problems in the following way. One first chooses a neural net $\mathcal{N}$ that is powerful enough to compute, respectively approximate, those functions or distributions that are potentially to be learned.

One then goes to a somewhat larger neural net $\tilde{\mathcal{N}}$ which can simulate $\mathcal{N}$ and which has the previously mentioned special structure which allows us to design an efficient learning algorithm for $\tilde{\mathcal{N}}$. One then trains $\tilde{\mathcal{N}}$ with a randomly drawn sample.

The previously described transition from $\mathcal{N}$ to $\tilde{\mathcal{N}}$ provides a curious theoretical counterpart to a common practize in the training of neural nets with backwards propagation: one often prefers to carry out such training on a neural net that has somewhat more units than necessary for computing the desired target functions.

The positive learning results of Theorem 2.1 and Theorem 3.1 are also of interest from the more general point of view of computational learning theory. Learnability in the here considered refinement of the PAC-model for "agnostic learning" (i.e. learning without a-priori assumptions about the target concept) is a rather strong property. In fact this property is so strong that so far there exist only very few positive results for learning of interesting concept classes resp. function classes in this model. Even some of the relatively few interesting concept classes that are learnable in the usual PAC-model (such as monomials of Boolean variables) are not learnable in the here considered refinement of the PAC-learning model (see [KSS]). Hence it is a rather noteworthy fact that function classes that are defined by arbitrarily complex analog neural nets are actually learnable in this refined version of the PAC-model.

Definition 1.1 *A network architecture (or "neural net") $\mathcal{N}$ of order v with k input nodes and l output nodes is a labelled acyclic directed graph $\langle V, E\rangle$. It has k nodes with fan-in 0 ("input nodes") that are labelled by $1,\dots,k$, and l nodes with fan-out 0 ("output nodes") that are labelled by $1,\dots,l$. Each node g of fan-in $r>0$ is called a computation node (or gate), and is labelled by some activation function $\gamma^g:\mathbf{R}\to\mathbf{R}$ and some polynomial $Q^g(y_1,\dots,y_r)$ of degree $\le v$. We assume that the ranges of activation functions of output nodes in $\mathcal{N}$ are bounded.*
The coefficients of all polynomials $Q^g(y_1,\dots,y_r)$ for gates g in $\mathcal{N}$ are called the programmable parameters of $\mathcal{N}$. Assume that $\mathcal{N}$ has w programmable parameters, and that some numbering of these has been fixed. Then each assignment $\underline{\alpha}\in\mathbf{R}^w$ of reals to the programmable parameters in $\mathcal{N}$ defines an analog circuit $\mathcal{N}^{\underline{\alpha}}$, which computes a function $\underline{x}\mapsto\mathcal{N}^{\underline{\alpha}}(\underline{x})$ from $\mathbf{R}^k$ into $\mathbf{R}^l$ in the following way: Assume that some input $\underline{x}\in\mathbf{R}$ has been assigned to the input nodes of $\mathcal{N}$. If a gate g in $\mathcal{N}$ has r immediate predecessors in $\langle V,E\rangle$ which output $y_1,\dots,y_r\in\mathbf{R}$, then g outputs $\gamma^g(Q^g(y_1,\dots,y_r))$.
Any parameters that occur in the definitions of the activation functions γ^g of $\mathcal{N}$ are referred to as architectural parameters of $\mathcal{N}$.

Definition 1.2 *A function $\gamma:\mathbf{R}\to\mathbf{R}$ is called piecewise polynomial if there are thresholds $t_1,\dots,t_s\in\mathbf{R}$ and polynomials $P_0,\dots,P_s$ such that $t_1<\dots<t_s$ and for each $i\in\{0,\dots,s\}: t_i\le x<t_{i+1}\ \Rightarrow\ \gamma(x)=P_i(x)$ (we set $t_0 := -\infty$ and $t_{s+1}:=\infty$).*
We refer to $t_1,\dots,t_s$ together with all coefficients in the polynomials $P_0,\dots,P_s$ as the parameters of γ. If the polynomials $P_0,\dots,P_s$ are of degree ≤ 1 then we call γ

piecewise linear
Note that we do not require that γ is continuous (or monotone).

2 Learning on Neural Nets with Piecewise Linear Activation Functions

We show in this section that for any network architecture $\mathcal{N}$ with piecewise polynomial activation functions there exists another network architecture $\tilde{\mathcal{N}}$ which can not only compute, but also *learn* any function $f : \mathbf{R}^k \to \mathbf{R}^l$ that can be computed by $\mathcal{N}$. The only difference between $\mathcal{N}$ and $\tilde{\mathcal{N}}$ is that each computation node in $\tilde{\mathcal{N}}$ has fan-out ≤ 1, whereas the nodes in $\mathcal{N}$ may have arbitrary fan-out.

If $\mathcal{N}$ has only one output node and depth ≤ 2 (i.e. $\mathcal{N}$ has at most one layer of "hidden units") then one can set $\tilde{\mathcal{N}} := \mathcal{N}$. For a general network architecture one applies the standard construction for transforming a directed acyclic graph into a tree. The construction of $\tilde{\mathcal{N}}$ from $\mathcal{N}$ proceeds recursively from the output level towards the input level: every computation node ν with fan-out $m > 1$ is replaced by m nodes with fan-out 1 which all use the same activation function as ν and which all get the same input as ν. It is obvious that for this classical construction from circuit theory (see [S]) the depth of $\tilde{\mathcal{N}}$ is the same as the depth of $\mathcal{N}$. In order to bound the size (i.e. number of gates) of $\tilde{\mathcal{N}}$, we first note that the fan-out of the input nodes does not have to be changed. Hence the transformation of the directed acyclic graph of $\mathcal{N}$ into a tree is only applied to the subgraph of depth $\text{depth}(\mathcal{N})-1$ which one gets from $\mathcal{N}$ by removing its input nodes. Furthermore one can easily see that the transformation does not increase the fan-in of any node. Obviously the fan-in of any gate in $\mathcal{N}$ is bounded by $\text{size}(\mathcal{N}) - 1$. Therefore the tree that provides the graph-theoretic structure for $\tilde{\mathcal{N}}$ has in addition to its k input-nodes up to $\sum_{i=0}^{\text{depth}(\mathcal{N})-1} \text{size}(\mathcal{N})^i \leq \frac{\text{size}(\mathcal{N})^{\text{depth}(\mathcal{N})}}{\text{size}(\mathcal{N})-1}$ computation nodes. Hence for bounded depth the increase in size is polynomially bounded.

Let $\mathbf{Q}_n$ be the set of rational numbers that can be written as quotients of integers with bit-length $\leq n$.

Let $F : \mathbf{R}^k \to \mathbf{R}^l$ be some arbitrary function, which we will view as a "prediction rule". For any given instance $\langle \underline{x}, \underline{y} \rangle \in \mathbf{R}^k \times \mathbf{R}^l$ we measure the *error* of F by $||F(\underline{x}) - \underline{y}||_1$, where $||\langle z_1, \ldots, z_l \rangle||_1 := \sum_{i=1}^{l} |z_i|$. For any distribution A over some subset of $\mathbf{R}^k \times \mathbf{R}^l$ we measure the *true error of F with regard to A* by $E_{\langle \underline{x}, \underline{y} \rangle \in A}[||F(\underline{x}) - \underline{y}||_1]$, i.e. the expected value of the error of F with respect to distribution A.

Theorem 2.1 *Let $\mathcal{N}$ be an arbitrary first order network architecture with k input nodes and l output nodes, and let $\tilde{\mathcal{N}}$ be the associated network architecture as defined above. We assume that all activation functions in $\mathcal{N}$ are piecewise linear with*

architectural parameters from $\mathbf{Q}$*. Let* $B \subseteq \mathbf{R}$ *be an arbitrary bounded set.*
Then there exists a polynomial $m(\frac{1}{\varepsilon}, \frac{1}{\delta})$ *and a learning algorithm LEARN such that for any given* $s, n \in \mathbf{N}$ *and any distribution* A *over* $\mathbf{Q}_n^k \times (\mathbf{Q}_n \cap B)^l$ *the following holds:*
For a sample $\zeta = (\langle \underline{x}_i, \underline{y}_i \rangle)_{i=1,\ldots,m}$ *of* $m \geq m(\frac{1}{\varepsilon}, \frac{1}{\delta})$ *examples that are independently drawn according to* A *the algorithm LEARN computes from* ζ, s, n *in polynomially in* m, s *and* n *many computation steps an assignment* $\underline{\tilde{\alpha}}$ *of rational numbers to the programmable parameters of the associated network architecture* $\tilde{\mathcal{N}}$ *such that*

$$E_{\langle \underline{x}, \underline{y} \rangle \in A}[||\tilde{\mathcal{N}}^{\underline{\tilde{\alpha}}}(\underline{x}) - \underline{y}||_1] \;\leq\; \varepsilon + \inf_{\underline{\alpha}\, \in \mathbf{Q}_s^w} E_{\langle \underline{x}, \underline{y} \rangle \in A}[||\mathcal{N}^{\underline{\alpha}}(\underline{x}) - \underline{y}||_1]$$

with probability $\geq 1 - \delta$ *(with regard to the random drawing of* ζ*).*

Consider the special case where the distribution A over $\mathbf{Q}_n^k \times (\mathbf{Q}_n \cap B)^l$ is of the form

$$A_{D, \underline{\alpha}_T}(\underline{x}, \underline{y}) = \begin{cases} D(\underline{x}) & , \text{ if } \underline{y} = \mathcal{N}^{\underline{\alpha}_T}(\underline{x}) \\ 0 & , \text{ otherwise} \end{cases}$$

for some *arbitrary* distribution D over the domain $\mathbf{Q}_n^k$ and some *arbitrary* $\underline{\alpha}_T \in \mathbf{Q}_s^w$. Then the term

$$\inf_{\underline{\alpha}\, \in \mathbf{Q}_s^w} E_{\langle \underline{x}, \underline{y} \rangle \in A}[||\mathcal{N}^{\underline{\alpha}}(\underline{x}) - \underline{y}||_1]$$

is equal to 0. Hence the preceding theorem implies that with learning algorithm LEARN the "learning network" $\tilde{\mathcal{N}}$ can "learn" with arbitrarily small true error any target function $\mathcal{N}^{\underline{\alpha}_T}$ that is computable on $\mathcal{N}$ with rational "weights" $\underline{\alpha}_T$. Thus by choosing $\mathcal{N}$ to be sufficiently large, one can guarantee that $\tilde{\mathcal{N}}$ can learn any target-function that might arise in the context of a specific learning problem.

In addition the theorem also applies to the quite realistic situation where the learner receives examples $\langle \underline{x}, \underline{y} \rangle$ of the form $\langle \underline{x}, \mathcal{N}^{\underline{\alpha}_T}(\underline{x})+$ noise$\rangle$, or even if there exists no "target function" $\mathcal{N}^{\underline{\alpha}_T}$ that would "explain" the actual distribution A of examples $\langle \underline{x}, \underline{y} \rangle$ ("agnostic learning").

Before we give the proof of Theorem 2.1 we first show that its claim may be viewed as a learning result within a refinement of Valiant's PAC-model [V]. This refined version of the PAC-model (essentially due to Haussler [H]) is better applicable to real world learning situations than the usual PAC-model:

- It makes no a-priori assumptions about the existence of a "target concept" or "target function" of a specific type which explains the empirical data (i.e. the "sample").
- It allows for arbitrary noise in the sample.
- It is not restricted to the learning of "concepts" (i.e. $0-1$ valued functions) since it allows arbitrary real numbers as predictions of the learner and as

target outputs in the sample. Hence it is for example also applicable for investigating learning (resp. approximation) of complicated real valued functions.

Of course one cannot expect miracles from a learner in such a real-world learning situation. It is in general impossible for him to produce a hypothesis with arbitrarily small true error with regard to the distribution A. This is clearly the case if the distribution A produces inconsistent data, or if A is generated by a target function (with added noise) that is substantially more complicated than any hypothesis function that the learner could possibly produce within his limited resources (e.g. with a fixed neural network architecture). Hence the best that one can expect from the learner is that he produces a hypothesis $\tilde{h}$ whose true error with regard to A is almost optimal in comparison with all possible hypotheses h from a certain pool $\mathcal{T}$ (the "touchstone class" in the terminology of [KSS]). This provides the motivation for the following definition, which slightly generalizes those in Haussler [H] and Kearns, Schapire, Sellie [KSS].

Definition 2.2 *Let $\mathcal{A} = \bigcup_{n \in \mathbf{N}} \mathcal{A}_n$ be an arbitrary set of distributions over finite subsets of $\mathbf{Q}^k \times \mathbf{Q}^l$ such that for any $n \in \mathbf{N}$ the bit-length of any point $\langle \underline{x}, \underline{y} \rangle$ that is drawn according to a distribution $A \in \mathcal{A}_n$ is bounded by a polynomial in n.*
Let $\mathcal{T} = (\mathcal{T}_s)_{s \in \mathbf{N}}$ be an arbitrary family of functions from $\mathbf{R}^k$ into $\mathbf{R}^l$ (with some fixed representation system) such that any $f \in \mathcal{T}_s$ has a representation whose bit-length is bounded by some polynomial in s. Let $\mathcal{H}$ be some arbitrary class of functions from $\mathbf{R}^k$ into $\mathbf{R}^l$.
One says that <u>$\mathcal{T}$ is efficiently learnable by $\mathcal{H}$ assuming $\mathcal{A}$</u> if there is an algorithm LEARN and a function $m(\varepsilon, \delta, s, n)$ that is bounded by a polynomial in $\frac{1}{\varepsilon}, \frac{1}{\delta}, s$ and n such that for any $\varepsilon, \delta \in (0,1)$ and any natural numbers s, n the following holds: If one draws independently $m \geq m(\varepsilon, \delta, s, n)$ examples according to some arbitrary distribution $A \in \mathcal{A}_n$, then LEARN computes from such a sample ζ with a number of computation steps that is polynomial in the parameter s and the bit-length of ζ the representation of some $\tilde{h} \in \mathcal{H}$ which has with probability $\geq 1 - \delta$ the property

$$E_{\langle \underline{x}, \underline{y} \rangle \in A}[||\tilde{h}(\underline{x}) - \underline{y}||_1] \ \leq \ \varepsilon + \inf_{h \in \mathcal{T}_s} E_{\langle \underline{x}, \underline{y} \rangle \in A}[||h(\underline{x}) - \underline{y}||_1].$$

In the special case $\mathcal{H} = \bigcup_{s \in \mathbf{N}} \mathcal{T}_s$ we say that <u>$\mathcal{T}$ is properly efficiently learnable assuming $\mathcal{A}$</u>.

Remark 2.3

a) It turns out in the learning results of Theorem 2.1 and Theorem 3.1 that the sample complexity $m(\varepsilon, \delta, s, n)$ can be chosen to be independent of s, n.

b) Note that Definition 2.2 contains as special case the common definition of PAC-learning [V]: Assume that $l = 1$ and $\mathcal{C}_s$ is some class of concepts over the domain $\mathbf{Q}^k$ so that each concept $C \in \mathcal{C}_s$ has a representation with $O(s)$ bits. Let $\mathcal{T}_s$ be the associated class of characteristic functions

$\chi_C : \mathbf{Q}^k \to \{0,1\}$ for concepts $C \in \mathcal{C}_s$. Let X_n be the domain $\mathbf{Q}_n^k$, and let $\mathcal{A}_n$ be the class of all distributions A over $X_n \times \{0,1\}$ such that there exists an arbitrary distribution D over X_n and some target concept $C_T \in \bigcup_{s \in \mathbf{N}} \mathcal{C}_s$ for which

$$A(\langle \underline{x}, y \rangle) = \begin{cases} D(\underline{x}) & , \quad \text{if } y = \chi_{C_T}(\underline{x}) \\ 0 & , \quad \text{otherwise.} \end{cases}$$

Then $(\mathcal{T}_s)_{s \in \mathbf{N}}$ is properly efficiently learnable assuming $\mathcal{A}$ in the sense of Definition 2.2 if and only if $(\mathcal{C}_s)_{s \in \mathbf{N}}$ is properly PAC-learnable in the sense of [V] (see [HKLW] for various equivalent versions of Valiant's definition of PAC-learning).

In addition the learning model considered here contains as special cases the model for agnostic PAC-learning of concepts from [KSS], and the model for PAC-learning of probabilistic concepts from [KS].

c) In the following the classes $\mathcal{T}_s$ and $\mathcal{H}$ will always be defined as classes of functions that are computable on a neural network $\mathcal{N}$ with a fixed architecture. For these classes one has a natural representation system: One may view any assignment of values $\underline{\alpha}$ to the programmable parameters of $\mathcal{N}$ as a representation for the function $\underline{x} \mapsto \mathcal{N}^{\underline{\alpha}}(\underline{x})$. We will always use this representation system in the following.

d) We may now rephrase Theorem 2.1 in terms of the general learning framework of Definition 2.2. Let $\mathcal{N}$ be as in Theorem 2.1, let $\mathcal{T}_s$ be the class of functions $f : \mathbf{R}^k \to \mathbf{R}^l$ computable on $\mathcal{N}$ with programmable parameters from $\mathbf{Q}_s$, and let $\mathcal{H}$ be the class of functions $f : \mathbf{R}^l \to \mathbf{R}^l$ that are computable with programmable parameters from $\mathbf{Q}$ on the associated network architecture $\tilde{\mathcal{N}}$. Let $\mathcal{A}_n$ be any class of distributions over $\mathbf{Q}_n^k \times (\mathbf{Q}_n \cap B)^l$.

Then $(\mathcal{T}_s)_{s \in \mathbf{N}}$ is efficiently learnable by $\mathcal{H}$ assuming $\bigcup_{n \in \mathbf{N}} \mathcal{A}_n$. Furthermore if all computation nodes in $\mathcal{N}$ have fan-out ≤ 1 then $(\mathcal{T}_s)_{s \in \mathbf{N}}$ is properly efficiently learnable assuming $\bigcup_{n \in \mathbf{N}} \mathcal{A}_n$.

For the proof of Theorem 2.1 we have to consider a suitable generalization of the notion of a VC-dimension for classes of real valued functions.

Definition 2.4 *(see Haussler [H]).*
Let X be some arbitrary domain, and let $\mathcal{F}$ be an arbitrary class of functions from X into $\mathbf{R}$. Then the pseudo-dimension of $\mathcal{F}$ is defined by

$$\begin{aligned} \dim_P(\mathcal{F}) \quad := \quad & \max \{|S| \ : \ S \subseteq X \text{ and } \exists h \ : \ S \to \mathbf{R} \text{ such that} \\ & \forall b \in \{0,1\}^S \ \exists f \in \mathcal{F} \ \forall x \in S \ (f(x) \geq h(x) \Leftrightarrow b(x) = 1)\}. \end{aligned}$$

Note that in the special case where $\mathcal{F}$ is a concept class (i.e. all $f \in \mathcal{F}$ are $0-1$ valued) the pseudo-dimension $\dim_P(\mathcal{F})$ coincides with the VC-dimension of $\mathcal{F}$.

Proposition 2.5 *Consider arbitrary network architectures $\mathcal{N}$ of order v with k input nodes, l output nodes, and w programmable parameters. Assume that each gate in $\mathcal{N}$ employs as activation function some piecewise polynomial (or piecewise rational) function of degree $\leq d$ with at most q pieces. For some arbitrary $p \in \{1, 2, \ldots\}$ we define*
$\mathcal{F} := \{ f : \mathbf{R}^{k+l} \to \mathbf{R} : \exists \underline{\alpha} \in \mathbf{R}^w \ \forall \underline{x} \in \mathbf{R}^k \ \forall \underline{y} \in \mathbf{R}^l (f(\underline{x}, \underline{y}) = ||\mathcal{N}^{\underline{\alpha}}(\underline{x}) - \underline{y}||_p)\}$.
Then one has $\dim_P(\mathcal{F}) = O(w^2 \log q)$ if $v, d, l = O(1)$.

Proof: Set $D := \dim_P(\mathcal{F})$. Then there are values $(\langle \underline{x}_i, \underline{y_i}, z_i \rangle)_{i=1,\ldots,D} \in (\mathbf{R}^{k+l+1})^D$ such that for every $b : \{1, \ldots, D\} \to \{0, 1\}$ there exists some $\underline{\alpha}_b \in \mathbf{R}^w$ so that for all $i \in \{1, \ldots, D\}$

$$||\mathcal{N}^{\underline{\alpha}_b}(\underline{x}_i) - \underline{y}_i||_p \geq z_i \Leftrightarrow b(i) = 1.$$

For each $i \in \{1, \ldots, D\}$ one can define in the theory of real numbers the set $\{\underline{\alpha} \in \mathbf{R}^w : ||\mathcal{N}^{\underline{\alpha}}(\underline{x}_i) - \underline{y_i}||_p \geq z_i\}$ by some first order formula Φ_i with real valued constants of the following structure: Φ_i is a disjunction of $\leq q^w \cdot 2^l$ conjunctions of $\leq 2w + l + 1$ atomic formulas, where each atomic formula is a polynomial inequality of degree $\leq (2vd)^w$. The disjunctions arise here from using different combinations of pieces from the activation functions. The conjunctions consist of all associated comparisons with thresholds of the activation functions. The factor 2 in the degree bound arises only in the case of piecewise rational activation functions.

By definition one has $\Phi_i(\underline{\alpha}_b) = 1 \Leftrightarrow b(i) = 1$ for $i = 1, \ldots, D$. Hence for any $b, \tilde{b} : \{1, \ldots, D\} \to \{0, 1\}$ with $b \neq \tilde{b}$ there exists some $i \in \{1, \ldots, D\}$ with $\Phi_i(\underline{\alpha}_b) \neq \Phi_i(\underline{\alpha}_{\tilde{b}})$. This implies that at least one of the $\leq S := D \cdot q^w \cdot 2^l \cdot (2w + l + 1)$ atomic formulas that occur in the D formulas $\Phi_i, \ldots, \Phi_D$ has different truth values for $\underline{\alpha}_b, \underline{\alpha}_{\tilde{b}}$.

On the other hand since each of the $\leq S$ atomic formulas is a polynomial inequality of degree $\leq (2vd)^w$, a theorem of Milnor [Mi] (see also [R]) implies that the number of different combinations of truth assignments to these atomic formulas that can be realized by different $\underline{\alpha} \in \mathbf{R}^w$ is bounded by $(S \cdot (2vd)^w)^{O(w)}$. Hence we have $2^D \leq (S \cdot (2vd)^w)^{O(w)}$, which implies by the definition of S that $D = O(w) \cdot (\log D + w \log q)$. This yields the desired estimate $D = O(w^2 \log q)$. ∎

Remark 2.6 This result generalizes earlier bounds for the VC-dimension of neural nets with piecewise polynomial activation functions and boolean network output from [M 92], [M 93a] (for bounded depth) and [GJ] (for unbounded depth). The preceding proof generalizes the argument from [GJ].

Proof of Theorem 2.1: We associate with $\mathcal{N}$ another network architecture $\tilde{\mathcal{N}}$ as defined before Theorem 2.1. By construction any function that is computable by $\mathcal{N}$ can also be computed by $\tilde{\mathcal{N}}$. Fix some interval $[b_1, b_2] \subseteq \mathbf{R}$ such that $B \subseteq [b_1, b_2], b_1 < b_2$, and such that the ranges of the activation functions of the output gates of $\mathcal{N}$ are contained in $[b_1, b_2]$. We define

$$b \ := \ l \cdot (b_2 - b_1) \ , \text{ and}$$

$$\mathcal{F} \;:=\; \{f\ :\mathbf{R}^k \times [b_1,b_2]^l \to [0,b] : \\ \exists \underline{\alpha} \in \mathbf{R}^w\ \forall \underline{x} \in \mathbf{R}^k\ \forall \underline{y} \in [b_1,b_2]^l\ (f(\underline{x},\underline{y}) = ||\tilde{\mathcal{N}}^{\underline{\alpha}}(\underline{x}) - \underline{y}||_1)\}.$$

Assume now that parameters $\varepsilon, \delta \in (0,1)$ with $\varepsilon \le b$ and $s, n \in \mathbf{N}$ have been fixed. For convenience we assume that s is sufficiently large so that all architectural parameters in $\mathcal{N}$ are from $\mathbf{Q}_s$. We define

$$m\left(\frac{1}{\varepsilon},\frac{1}{\delta}\right) := \frac{257\cdot b^2}{\varepsilon^2}\left(2\cdot \dim_P(\mathcal{F})\cdot ln\frac{33eb}{\varepsilon} + ln\frac{8}{\delta}\right).$$

Note that $\dim_P(\mathcal{F}) < \infty$ by Proposition 2.5. By Corollary 2 of Theorem 7 in Haussler [H] one has for $m \ge m(\frac{1}{\varepsilon},\frac{1}{\delta})$, $K := \frac{\sqrt{257}}{8} \in (2,3)$, and any distribution A over $\mathbf{Q}_n^k \times (\mathbf{Q}_n \cap [b_1,b_2])^l$

$$(1) \qquad Pr_{\zeta\in A^m}\left[\left\{\exists f \in \mathcal{F} : \left|\left(\frac{1}{m}\sum_{\langle \underline{x},\underline{y}\rangle \in \zeta} f(\underline{x},\underline{y})\right) - E_{\langle \underline{x},\underline{y}\rangle\in A}[f(\underline{x},\underline{y})]\right| > \frac{\varepsilon}{K}\right\}\right] \le \delta,$$

where $E_{\langle \underline{x},\underline{y}\rangle\in A}[f(\underline{x},\underline{y})]$ is the expectation of $f(\underline{x},\underline{y})$ with regard to distribution A.

We design an algorithm LEARN that computes for any $m \in \mathbf{N}$, any sample

$$\zeta = (\langle \underline{x_i}, \underline{y_i}\rangle)_{i\in\{1,\ldots,m\}} \in (\mathbf{Q}_n^k \times (\mathbf{Q}_n \cap [b_1,b_2])^l)^m,$$

and any given $s \in \mathbf{N}$ in polynomially in m, s, n computation steps an assignment $\underline{\tilde{\alpha}}$ of rational numbers to the parameters in $\tilde{\mathcal{N}}$ such that the function $\tilde{h}$ that is computed by $\tilde{\mathcal{N}}^{\underline{\tilde{\alpha}}}$ satisfies

$$(2) \qquad \frac{1}{m}\sum_{i=1}^m ||\tilde{h}(\underline{x_i}) - \underline{y_i}||_1 \le \left(1-\frac{2}{K}\right)\varepsilon + \inf_{\underline{\alpha}\in\mathbf{Q}_s^w} \frac{1}{m}\sum_{i=1}^m ||\mathcal{N}^{\underline{\alpha}}(\underline{x_i}) - \underline{y_i}||_1.$$

This suffices for the proof of Theorem 2.1, since (1) and (2) together imply that, for any distribution A over $\mathbf{Q}_n^k \times (\mathbf{Q}_n \cap [b_1,b_2])^l$ and any $m \ge m(\frac{1}{\varepsilon},\frac{1}{\delta})$, with probability $\ge 1-\delta$ (with respect to the random drawing of $\zeta \in A^m$) the algorithm LEARN outputs for inputs ζ and s an assignment $\underline{\tilde{\alpha}}$ of rational numbers to the parameters in $\tilde{\mathcal{N}}$ such that

$$E_{\langle \underline{x},\underline{y}\rangle\in A}[||\tilde{\mathcal{N}}^{\underline{\tilde{\alpha}}}(\underline{x}) - \underline{y}||_1] \le \varepsilon + \inf_{\underline{\alpha}\in\mathbf{Q}_s^w} E_{\langle \underline{x},\underline{y}\rangle\in A}[||\mathcal{N}^{\underline{\alpha}}(\underline{x}) - \underline{y}||_1].$$

The algorithm LEARN computes optimal solutions for polynomially in m many linear programming problems $LP_1,\ldots,LP_{p(m)}$ in order to find values $\underline{\tilde{\alpha}}$ for the programmable parameters in $\tilde{\mathcal{N}}$ so that $\tilde{\mathcal{N}}^{\underline{\tilde{\alpha}}}$ satisfies (2). The reduction of the computation of $\underline{\tilde{\alpha}}$ to *linear* programming is nontrivial, since for any fixed input $\underline{x}$ the output $\tilde{\mathcal{N}}^{\underline{\alpha}}(\underline{x})$ is in general not linear in the programmable parameters $\underline{\alpha}$. This becomes obvious if one considers for example the composition of two very simple gates g_1 and g_2 on levels 1 and 2 of $\tilde{\mathcal{N}}$, whose activation functions γ_1, γ_2 satisfy $\gamma_1(y) = \gamma_2(y) = y$. Assume $z = \sum_{i=1}^{k} \alpha_i x_i + \alpha_0$ is the input to gate g_1, and g_2 receives as input $\sum_{j=1}^{q} \alpha'_j y_j + \alpha'_0$ where $y_1 = \gamma_1(z) = z$ is the output of gate g_1. Then g_2 outputs $\alpha'_1 \cdot \left(\sum_{i=1}^{k} \alpha_i x_i + \alpha_0 \right) + \sum_{j=2}^{q} \alpha'_j y_j + \alpha'_0$. Obviously for fixed network input $\underline{x} = \langle x_1, \ldots, x_k \rangle$ this term is not linear in the weights $\alpha'_1, \alpha_1, \ldots, \alpha_k$.

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

We employ a method from [M 93a] that allows us to replace the nonlinear conditions on the programmable parameters $\underline{\alpha}$ of $\tilde{\mathcal{N}}$ by linear conditions for a transformed set $\underline{c}, \underline{\beta}$ of parameters. We simulate $\tilde{\mathcal{N}}^{\underline{\alpha}}$ by another network architecture $\hat{\mathcal{N}}[\underline{c}]^{\underline{\beta}}$ (which one may view as a "normal form" for $\tilde{\mathcal{N}}^{\underline{\alpha}}$) that uses the same graph $\langle V, E \rangle$ as $\tilde{\mathcal{N}}$, but different activation functions and different values $\underline{\beta}$ for its programmable parameters. The activation functions of $\hat{\mathcal{N}}[\underline{c}]$ depend on $|V|$ new architectural parameters $\underline{c} \in \mathbf{R}^{|V|}$, which we call *scaling parameters* in the following. Whereas the architectural parameters of a network architecture are usually kept fixed, we will be forced to change the scaling parameters of $\hat{\mathcal{N}}$ along with its programmable parameters $\underline{\beta}$. Although this new network architecture 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 $\{-1, 0, 1\}$. Hence we can treat them as constants with at most 3 possible values 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 $\hat{\mathcal{N}}[\underline{c}]$ for fixed network inputs (the variables for weights of gates on level 1, the variables for the biases of gates on all levels, *and the new variables for the scaling parameters* $\underline{c}$) appear only *linearly* in those inqualities.

We briefly indicate the construction of $\hat{\mathcal{N}}$. Consider the activation function γ of an arbitrary gate in $\tilde{\mathcal{N}}$. Since γ is piecewise linear, there are fixed architectural parameters $t_1 < \cdots < t_s$, $a_0, \ldots, a_s$, $b_0, \ldots, b_s$ (which may be different for different gates g) such that with $t_0 := -\infty$ and $t_{s+1} := +\infty$ one has $\gamma(x) = a_i x + b_i$ for $x \in \mathbf{R}$ with $t_i \leq x < t_{i+1}$; $i = 0, \ldots, s$. For an arbitrary scaling parameter $c \in \mathbf{R}^+$ we associate with γ the following piecewise linear activation function γ^c: the thresholds of γ^c are $c \cdot t_1, \ldots, c \cdot t_s$ and its output is $\gamma^c(x) = a_i x + c \cdot b_i$ for $x \in \mathbf{R}$

with $c \cdot t_i \leq x < c \cdot t_{i+1}; i = 0, \ldots, s$ (set $c \cdot t_0 := -\infty, c \cdot t_{s+1} := +\infty$). Thus for all reals $c > 0$ the function γ^c is related to γ through the equality:

$$\forall x \in \mathbf{R}(\gamma^c(c \cdot x) = c \cdot \gamma(x)).$$

Assume that $\underline{\alpha}$ is some arbitrary given assignment to the programmable parameters in $\tilde{\mathcal{N}}$. We transform $\tilde{\mathcal{N}}^{\underline{\alpha}}$ through a recursive process into a "normal form" $\hat{\mathcal{N}}[\underline{c}]^{\underline{\beta}}$ in which all weights on edges between computation nodes are from $\{-1, 0, 1\}$, such that

$$\forall \underline{x} \in \mathbf{R}^k \left(\tilde{\mathcal{N}}^{\underline{\alpha}}(\underline{x}) = \hat{\mathcal{N}}[\underline{c}]^{\underline{\beta}}(\underline{x}) \right).$$

Assume that an output gate g_{out} of $\tilde{\mathcal{N}}^{\underline{\alpha}}$ receives as input $\sum_{i=1}^{q} \alpha_i y_i + \alpha_0$, where $\alpha_1, \ldots, \alpha_q, \alpha_0$ are the weights and the bias of g_{out} (under the assignment $\underline{\alpha}$) and $y_1, \ldots, y_q$ are the (real valued) outputs of the immediate predecessors $g_1, \ldots, g_q$ of g. For each $i \in \{1, \ldots, q\}$ with $\alpha_i \neq 0$ such that g_i is not an input node we replace the activation function γ_i of g_i by $\gamma_i^{|\alpha_i|}$, and we multiply the weights and the bias of gate g_i with $|\alpha_i|$. Finally we replace the weight α_i of gate g_{out} by $\text{sgn}(\alpha_i)$, where $\text{sgn}(\alpha_i) := 1$ if $\alpha_i > 0$ and $\text{sgn}(\alpha_i) := -1$ if $\alpha_i < 0$. This operation has the effect that the multiplication with $|\alpha_i|$ is carried out *before* the gate g_i (rather than after g_i, as done in $\tilde{\mathcal{N}}^{\underline{\alpha}}$), but that the considered output gate g_{out} still receives the same input as before. If $\alpha_i = 0$ we want to "freeze" that weight at 0. This can be done by deleting g_i and all gates below g_i from $\hat{\mathcal{N}}$.

The analogous operations are recursively 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 $\tilde{\mathcal{N}}^{\underline{\alpha}}$, since they have been changed in the preceding step). We exploit here the assumption that each gate in $\hat{\mathcal{N}}$ has fan-out ≤ 1.

Let $\underline{\beta}$ consist of the new weights on edges adjacent to input nodes and of the resulting biases of all gates in $\hat{\mathcal{N}}$. Let $\underline{c}$ consist of the resulting scaling parameters at the gates of $\hat{\mathcal{N}}$. Then we have $\forall \underline{x} \in \mathbf{R}^k \left(\tilde{\mathcal{N}}^{\underline{\alpha}}(\underline{x}) = \hat{\mathcal{N}}[\underline{c}]^{\underline{\beta}}(\underline{x}) \right)$. Furthermore $c > 0$ for all scaling parameters c in $\underline{c}$.

At the end of this proof we will also need the fact that the previously described parameter transformation can be inverted. One can easily compute from any assignment $\underline{\tilde{c}}, \underline{\tilde{\beta}}$ to the parameters in $\hat{\mathcal{N}}$ with $c > 0$ for all c in $\underline{\tilde{c}}$ an assignment $\underline{\tilde{\alpha}}$ to the programmable parameters in $\tilde{\mathcal{N}}$ such that $\forall \underline{x} \in \mathbf{R}^k \left(\tilde{\mathcal{N}}^{\underline{\tilde{\alpha}}}(\underline{x}) = \hat{\mathcal{N}}[\underline{\tilde{c}}]^{\underline{\tilde{\beta}}}(\underline{x}) \right)$. This backwards transformation is also defined by recursion. Consider some gate g on level 1 in $\hat{\mathcal{N}}$ that uses (for the new parameter assignment $\underline{\tilde{c}}$) the scaling parameter $c > 0$ for its activation function γ^c. Then we replace the weights $\alpha_1, \ldots, \alpha_k$ and bias α_0 of gate g in $\hat{\mathcal{N}}[\underline{\tilde{c}}]^{\underline{\tilde{\beta}}}$ by $\frac{\alpha_1}{c}, \ldots, \frac{\alpha_k}{c}, \frac{\alpha_0}{c}$; and γ^c by γ. Furthermore if $r \in \{-1, 1\}$ 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 $\hat{\mathcal{N}}[\underline{\tilde{c}}]^{\underline{\tilde{\beta}}}$ (for every network input). Assume now that $\alpha'_1, \ldots, \alpha'_q$ are the weights that the incoming edges of g' get assigned in this way, that α'_0 is the bias of g' in the assignment $\underline{\tilde{\beta}}$, and that $c' > 0$ is the scaling parameter of g' in $\hat{\mathcal{N}}[\underline{\tilde{c}}]^{\underline{\tilde{\beta}}}$. Then we assign the new weights $\frac{\alpha'_1}{c'}, \ldots, \frac{\alpha'_q}{c'}$ and the new bias $\frac{\alpha'_0}{c'}$ to g', and we multiply the weight on the outgoing edge from g' by c'.

In the remainder of this proof we specify how the algorithm LEARN computes for any given sample $\zeta = (\langle \underline{x_i}, \underline{y_i} \rangle)_{i=1,\ldots,m} \in (\mathbf{Q}_n^k \times (\mathbf{Q}_n \cap [b_1, b_2])^l)^m$ and any given $s \in \mathbf{N}$ with the help of linear programming a new assignment $\underline{\tilde{c}}, \underline{\tilde{\beta}}$ to the parameters in $\hat{\mathcal{N}}$ such that the function $\tilde{h}$ that is computed by $\hat{\mathcal{N}}[\underline{\tilde{c}}]^{\underline{\tilde{\beta}}}$ satisfies (2). For that purpose we describe the computations of $\hat{\mathcal{N}}$ for the *fixed* inputs $\underline{x_i}$ from the sample $\zeta = (\langle \underline{x_i}, \underline{y_i} \rangle)_{i=1,\ldots,m}$ by polynomially in m many systems $L_1, \ldots, L_{p(m)}$ that each consist of $O(m)$ linear inequalities with the transformed parameters $\underline{c}, \underline{\beta}$ as variables. For each input $\underline{x_i}$ one uses for each gate g in $\hat{\mathcal{N}}$ two inequalities that specify the relation of the input s of g to two adjacent thresholds t, t' of the piecewise linear activation function γ^c of g. By construction of $\hat{\mathcal{N}}$ the gate input s can always be written as a linear expression in $\underline{c}, \underline{\beta}$ (provided one knows which linear pieces were used by the preceding gates). A problem is caused by the fact that this construction leads to a system of inequalities that contains both strict inequalities "$s_1 < s_1$" and weak inequalities "$s_1 \leq s_2$". Each scaling parameter c in $\underline{c}$ gives rise to a strict inequality $-c < 0$. Further strict inequalities "$s_1 < s_2$" arise when one compares the input s_1 of some gate g in $\hat{\mathcal{N}}$ with a threshold s_2 of the piecewise linear activation function γ^c of this gate g. Unfortunately linear programming cannot be applied directly to a system that contains both strict and weak inequalities. Hence we replace all strict inequalities "$s_1 < s_2$" by "$s_1 + 2^{-\rho} \leq s_2$", where

$$\rho := 2\,(s \cdot \mathrm{size}(\mathcal{N}))^{\mathrm{depth}(\mathcal{N})-1} \cdot \left(s^2 \cdot \mathrm{depth}(\mathcal{N}) \cdot (k+2) \cdot n\right).$$

This construction, as well as the particular choice of ρ will be justified later. A precise analysis shows that in the preceding construction we do not arrive at a single network architecture $\hat{\mathcal{N}}$ but at up to $2^{\tilde{w}}$ different architectures, where $\tilde{w}$ is the number of weights in $\tilde{\mathcal{N}}$. This is caused by the special clause in the transformation from $\tilde{\mathcal{N}}^{\underline{\alpha}}$ to $\hat{\mathcal{N}}[\underline{c}]^{\underline{\beta}}$ for the case that $\alpha_i = 0$ for some weight α_i in $\underline{\alpha}$ (in that case the initial segment of the network below that edge is deleted in $\hat{\mathcal{N}}$). There are at most $2^{\tilde{w}} = O(1)$ ways of assigning the weight 0 to certain edges in $\tilde{\mathcal{N}}$, and correspondingly there are at most $2^{\tilde{w}}$ variations of $\hat{\mathcal{N}}$ that have to be considered (which all arise from the full network by deleting certain initial segments). Each of these variations of $\hat{\mathcal{N}}$ gives rise to a different system of linear inequalities in the preceding construction.

A less trivial problem for describing the computations of $\hat{\mathcal{N}}$ for the fixed network inputs $\underline{x_1}, \ldots, \underline{x_m} \in \mathbf{Q}_n^k$ by systems of linear inequalities (with the parameters $\underline{c}, \underline{\beta}$ as variables) arises from the fact that for the same network input $\underline{x_i}$ different values of the variables $\underline{c}, \underline{\beta}$ will lead to the use of different linear pieces of the activation functions in $\hat{\mathcal{N}}$. Therefore one has to use a whole family $L_1, \ldots, L_{p(m)}$ of $p(m)$ different systems of linear inequalities, where each system L_j reflects one possibility for employing specific linear pieces of the activation functions in $\hat{\mathcal{N}}$ for specific network inputs $\underline{x}_1, \ldots, \underline{x}_m$, for deleting certain initial segments of $\hat{\mathcal{N}}$ as discussed before, and for employing different combinations of weights from $\{-1, 1\}$ for edges between computation nodes.

Each of these systems L_j has to be consistent in the following sense: if L_j contains for some network input $\underline{x}_i$ the inequalities $t \le s$ and $s + 2^{-\rho} \le t'$ for two adjacent thresholds t, t' of the activation function γ^c of some gate g in $\hat{\mathcal{N}}$, and if f is the linear piece of γ^c in the interval $[t, t')$, then this linear piece f is used for describing, for this network input $\underline{x}_i$ and for all subsequent gates g', the contribution of gate g for the input of g' in the two linear inequalities for g' in L_j. It should be noted on the side that the scaling parameter c occurs as variable both in the thresholds t, t' as well as in the definition of each linear piece f of the activation function γ^c. However this causes no problem since by construction of $\hat{\mathcal{N}}$ the considered terms s, t, t' as well as the terms involving f are linear in the variables $\underline{c}, \underline{\beta}$.

It looks as if this approach might lead to the consideration of exponentially in m many systems L_j: We may have to allow that for any set $S \subseteq \{1, \ldots, m\}$ one linear piece of the activation function γ^c of a gate g is used for network inputs $\underline{x}_i$ with $i \in S$, and another linear piece of γ^c is used for network inputs $\underline{x}_i$ with $i \notin S$. Hence each set S might give rise to a different system L_j.

One can show that it suffices to consider only polynomially in m many systems of inequalities L_j by exploiting that all inequalities are linear, and that the input space for $\hat{\mathcal{N}}$ has bounded dimension k. A single threshold t between two linear pieces of the activation function of some gate g on level 1 divides the m inputs $\underline{x}_1, \ldots, \underline{x}_m$ in at most $2^k \cdot \binom{m}{k}$ different ways. One arrives at this estimate by considering all $\binom{m}{k}$ subsets S of $\{\underline{x}_1, \ldots, \underline{x}_m\}$ of size k, and then all 2^k partitions of S into subsets S_1 and S_2. For any such sets S_1 and S_2 we consider a *pair* of halfspaces $H_1 := \{\underline{x} \in \mathbf{R}^k : \underline{x} \cdot \underline{\hat{\alpha}} + 2^{-\rho} \le t\}$ and $H_2 := \{\underline{x} \in \mathbf{R}^k : \underline{x} \cdot \underline{\hat{\alpha}} \ge t\}$, where the weights $\underline{\hat{\alpha}}$ for gate g are chosen in such a way that $\underline{x}_i \cdot \underline{\hat{\alpha}} + 2^{-\rho} = t$ for all $i \in S_1$ and $\underline{x}_i \cdot \underline{\hat{\alpha}} = t$ for all $i \in S_2$. If the halfspaces H_1, H_2 are uniquely defined by this condition and if they have the property that $\underline{x}_i \in H_1 \cup H_2$ for $i = 1, \ldots, m$, then they define one of the $\le 2^k \cdot \binom{m}{k}$ partitions of $\underline{x}_1, \ldots, \underline{x}_m$ which we consider for the threshold t of this gate g. It is easy to see that *each* setting $\underline{\hat{\alpha}}$ of the weights of gate g such that $\forall i \in \{1, \ldots, m\}(\underline{x}_i \in H_1 \cup H_2)$ for the associated halfspaces H_1 and H_2 defines via threshold t a partition of $\{\underline{x}_1, \ldots, \underline{x}_m\}$ which agrees with one of the previously described partitions. Each of these up to $2^k \cdot \binom{m}{k}$ many partitions may give rise to a different system L_j of linear inequalities.

In addition each threshold t' between linear pieces of a gate g' on level > 1 gives rise to different partitions of the m inputs, and hence to different systems L_j. In fact the partition of the m inputs that is caused by t' is in general of a more complicated structure. Assume that k' is the number of thresholds between linear pieces of activation functions of preceding gates. If each of these preceding thresholds partitions the m inputs by a hyperplane, then altogether they split the m inputs into up to $2^{k'}$

subsets. For each of these subsets the preceding gates will in general use different linear pieces of their activation functions (see the consistency condition described before). Hence threshold t' of gate g' will in general not partition the m network inputs by a single hyperplane, but by different hyperplanes for each of the $2^{k'}$ subsets of the m inputs. However we want to keep the number $p(m)$ of systems L_j *simply* exponential in the size of $\hat{\mathcal{N}}$. This is not relevant for the proof of Theorem 2.1, but for the parallelized speed-up of LEARN that will be considered in the subsequent Remark 2.7. Therefore we restructure the partition of the m inputs that is caused by any threshold t' of a gate g' on level > 1 in the following fashion. We view the input for gate g' for each of the m network inputs as a linear function, which has as variables the weights for gates on level 1 and the scaling factors of preceding gates in $\hat{\mathcal{N}}$. The number of these variables can be bounded by the number w of weights in $\tilde{\mathcal{N}}$. Each of the m network inputs generates different coefficients for these variables. These coefficients will also depend on the particular linear pieces of activation functions of preceding gates that are used for each particular network input. Nevertheless for each partition generated by thresholds of preceding gates we only have to consider for threshold t' all possibilities how a pair of hyperplanes with distance $2^{-\rho}$ can divide the resulting m different coefficient vectors of length w. Obviously there exist only $O(m^w)$ different possibilities for that.

Altogether the algorithm LEARN generates for each of the polynomially in m many partitions of $\underline{x_1}, \ldots, \underline{x_m}$ which arise in the previously described fashion from thresholds between linear pieces of activation functions of gates in $\hat{\mathcal{N}}$, and for each assignment of weights from $\{-1, 0, 1\}$ to edges between computation nodes in $\hat{\mathcal{N}}$ a separate system L_j of linear inequalities, for $j = 1, \ldots, p(m)$. By construction one can bound $p(m)$ by a polynomial in m.

We now expand each of the systems L_j (which has only $O(1)$ variables) into a linear programming problem LP_j with $O(m)$ variables. We add to L_j for each of the l output nodes ν of $\hat{\mathcal{N}}$ $2m$ new variables u_i^ν, v_i^ν for $i = 1, \ldots, m$, and the $4m$ inequalities

$$t_j^\nu(\underline{x_i}) \leq (\underline{y_i})_\nu + u_i^\nu - v_i^\nu, \quad t_j^\nu(\underline{x_i}) \geq (\underline{y_i})_\nu + u_i^\nu - v_i^\nu, \quad u_i^\nu \geq 0, \quad v_i^\nu \geq 0,$$

where $(\langle \underline{x_i}, \underline{y_i} \rangle)_{i=1,\ldots,m}$ is the fixed sample ζ and $(\underline{y_i})_\nu$ is that coordinate of $\underline{y_i}$ which corresponds to the output node ν of $\hat{\mathcal{N}}$. In these inequalities the symbol $t_j^\nu(\underline{x_i})$ denotes the term (which is by construction linear in the variables $\underline{c}, \underline{\beta}$) that represents the output of gate ν for network input $\underline{x_i}$ in this system L_j. One should note that these terms $t_j^\nu(\underline{x_i})$ will in general be different for different j, since different linear pieces of the activation functions at preceding gates may be used in the computation of $\hat{\mathcal{N}}$ for the same network input $\underline{x_i}$. Furthermore we expand the system L_j of linear inequalities to a linear programming problem LP_j in canonical form by adding the optimization requirement

$$\text{minimize} \qquad \sum_{i=1}^{m} \sum_{\substack{\nu \text{ output} \\ \text{node in } \hat{\mathcal{N}}}} (u_i^\nu + v_i^\nu).$$

The algorithm LEARN employs an efficient algorithm for linear programming (e.g. the ellipsoid algorithm, see [PS]) in order to compute in altogether polynomially in m, s and n many steps an optimal solution for each of the linear programming problems $LP_1, \ldots, LP_{p(m)}$. We write h_j for the function from $\mathbf{R}^k$ into $\mathbf{R}^l$ that is computed by $\hat{\mathcal{N}}[\underline{c}]^{\underline{\beta}}$ for the optimal solution $\underline{c}, \underline{\beta}$ of LP_j. The algorithm LEARN computes $\frac{1}{m}\sum_{i=1}^{m} ||h_j(\underline{x_i}) - \underline{y_i}||_1$ for $j = 1, \ldots, p(m)$. Let $\tilde{j}$ be that index for which this expression has a minimal value. Let $\underline{\tilde{c}}, \underline{\tilde{\beta}}$ be the associated optimal solution of $LP_{\tilde{j}}$ (i.e. $\hat{\mathcal{N}}[\underline{\tilde{c}}]^{\underline{\tilde{\beta}}}$ computes $h_{\tilde{j}}$). LEARN employs the previously described backwards transformation from $\underline{\tilde{c}}, \underline{\tilde{\beta}}$ into values $\underline{\tilde{\alpha}}$ for the programmable parameters of $\tilde{\mathcal{N}}$ such that $\forall \underline{x} \in \mathbf{R}^k (\tilde{\mathcal{N}}^{\underline{\tilde{\alpha}}}(\underline{x}) = \hat{\mathcal{N}}[\underline{\tilde{c}}]^{\underline{\tilde{\beta}}}(\underline{x}))$. These values $\underline{\tilde{\alpha}}$ are given as output of the algorithm LEARN.

We will show that $\tilde{h} := h_{\tilde{j}}$ satisfies condition (2), i.e.

$$\frac{1}{m}\sum_{i=1}^{m} ||\tilde{h}(\underline{x_i}) - \underline{y_i}||_1 \;\leq\; \left(1 - \frac{2}{K}\right) \cdot \varepsilon + \inf_{\underline{\alpha} \in \mathbf{Q}_s^w} \frac{1}{m}\sum_{i=1}^{m} ||\mathcal{N}^{\underline{\alpha}}(\underline{x_i}) - \underline{y_i}||_1 .$$

Fix some $\underline{\alpha'} \in \mathbf{Q}_s^w$ with

$$(3) \quad \frac{1}{m}\sum_{i=1}^{m} ||\mathcal{N}^{\underline{\alpha'}}(\underline{x_i}) - \underline{y_i}||_1 \;\leq\; \left(1 - \frac{2}{K}\right) \cdot \varepsilon + \inf_{\underline{\alpha} \in \mathbf{Q}_s^w} \frac{1}{m}\sum_{i=1}^{m} ||\mathcal{N}^{\underline{\alpha}}(\underline{x_i}) - \underline{y_i}||_1 .$$

Let $\underline{\tilde{\alpha'}}$ consist of corresponding values from $\mathbf{Q}_s$ such that $\forall \underline{x} \in \mathbf{R}^k (\mathcal{N}^{\underline{\alpha'}}(\underline{x}) = \tilde{\mathcal{N}}^{\underline{\tilde{\alpha'}}}(\underline{x}))$. According to the previously described construction one can transform $\underline{\tilde{\alpha'}}$ into parameters $\underline{c}, \underline{\beta}$ from $\mathbf{Q}_{s \cdot \text{depth}(\tilde{\mathcal{N}})}$ such that $\forall \underline{x} \in \mathbf{R}^k (\tilde{\mathcal{N}}^{\underline{\tilde{\alpha'}}}(\underline{x}) = \hat{\mathcal{N}}[\underline{c}]^{\underline{\beta}}(\underline{x}))$. We use here our assumption that all architectural parameters in $\mathcal{N}$ have values in $\mathbf{Q}_s$. Since by definition of the transformation from $\tilde{\mathcal{N}}$ into $\hat{\mathcal{N}}$ we delete initial segments of $\hat{\mathcal{N}}$ below edges with weight 0 in $\underline{\tilde{\alpha'}}$, we can assume $c > 0$ for all remaining scaling parameters c in $\underline{c}$.

It follows that for these values of $\underline{c}, \underline{\beta}$ each term that represents the input of some gate g in $\hat{\mathcal{N}}[\underline{c}]^{\underline{\beta}}$ for some network input from $\mathbf{Q}_n^k$ has a value in $\mathbf{Q}_{\rho/2}$ for $\rho := 2\,(s \cdot \text{size}(\mathcal{N}))^{\text{depth}(\mathcal{N})-1} \cdot \left(s^2 \cdot \text{depth}(\mathcal{N}) \cdot (k+2) \cdot n\right)$. Hence whenever the input s_1 of some gate g in $\hat{\mathcal{N}}[\underline{c}]^{\underline{\beta}}$ satisfies for some network input from $\mathbf{Q}_n^k$ the strict inequality "$s_1 < s_2$" (for some threshold s_2 of this gate g), the inequality "$s_1 + 2^{-\rho} \leq s_2$"

is also satisfied. Analogously each scaling parameter $c > 0$ in $\underline{c}$ satisfies $c \geq 2^{-\rho}$. These observations imply that the values for the parameters $\underline{c}, \underline{\beta}$ that result by the transformation from $\underline{\tilde{\alpha}'}$ give rise to a feasible solution for one of the linear programming problems LP_j, for some $j \in \{1, \ldots, p(m)\}$. The cost $\sum_{i=1}^{m} \sum_{\substack{\nu \text{ output} \\ \text{node in } \tilde{\mathcal{N}}}} (u_i^\nu + v_i^\nu)$ of this feasible solution can be chosen to be $\sum_{i=1}^{m} ||\mathcal{N}^{\underline{\alpha'}}(\underline{x_i}) - \underline{y_i}||_1$ (for each i, ν set at least one of u_i^ν, v_i^ν equal to 0). This implies that the optimal solution of LP_j has a cost of at most $\sum_{i=1}^{m} ||\mathcal{N}^{\underline{\alpha'}}(\underline{x_i}) - \underline{y_i}||_1$. Hence we have $\sum_{i=1}^{m} ||\tilde{h}(\underline{x_i}) - \underline{y_i}||_1 \leq \sum_{i=1}^{m} ||\mathcal{N}^{\underline{\alpha'}}(\underline{x_i}) - \underline{y_i}||_1$ by the definition of algorithm LEARN. Therefore the desired inequality (2) follows from (3). This completes the proof of Theorem 2.1. ∎

Remark 2.7 The algorithm LEARN can be speeded up substantially on a parallel machine. Furthermore if the individual processors of the parallel machine are allowed to use random bits, hardly any global control is required for this parallel computation. We use polynomially in m many processors. Each processor picks at random one of the systems L_j of linear inequalities and solves the corresponding linear programming problem LP_j. Then the parallel machine compares in a "competetive phase" the costs $\sum_{i=1}^{m} ||h_j(\underline{x_i}) - \underline{y_i}||_1$ of the solutions h_j that have been computed by the individual processors. It outputs the weights $\underline{\tilde{\alpha}}$ for $\tilde{\mathcal{N}}$ that correspond to the best ones of these solutions h_j.

In this parallelized version of LEARN the only interaction between individual processors occurs in the competetive phase. Even without any coordination between individual processors one can ensure that with high probability each of the relevant linear programming problems LP_j for $j = 1, \ldots, p(m)$ is solved by at least one of the individual processors, provided that there are slightly more than $p(m)$ such processors with random bits. Each processor simply picks at random one of the problems LP_j and solves it. It turns out that the computation time of each individual processor (and hence the parallel computation time of LEARN) is *polynomial in m and in the total number w of weights in $\tilde{\mathcal{N}}$*. The construction of the systems L_j (for $j = 1, \ldots, p(m)$) in the proof of Theorem 2.1 implies that only polynomially in m and w many random bits are needed in order to choose randomly one of the linear programming problems LP_j, $j = 1, \ldots, p(m)$. Furthermore with the help of some polynomial time algorithm for linear programming each problem LP_j can be solved with polynomially in m and w many computation steps.

The total number of processors for this parallel version of LEARN is simply exponential in w. However even on a parallel machine with fewer processors the same randomized parallel algorithm gives rise to a rather promising approximative learning algorithm. Such a "scaled-down" version of LEARN is no longer guaranteed to find probably an approximately optimal weight setting in the strict sence of the PAC-learning model. However it might very well provide a satisfactory performance for various practically important learning problems for multi-layer analog neural nets.

3 Learning on Neural Nets with Piecewise Polynomial Activation Functions

In this section we extend the learning result from section 2 to high order network architectures with piecewise polynomial activation functions.

Theorem 3.1 *Let $\mathcal{N}$ be some arbitrary high order network architecture with k inputs and l outputs. We assume that all activation functions of gates in $\mathcal{N}$ are piecewise polynomial with architectural parameters from* $\mathbf{Q}$. *Then one can construct an associated first order network architecture $\tilde{\mathcal{N}}$ with activation functions from the class $\{heaviside,\ x \mapsto x,\ x \mapsto x^2\}$ such that the same learning property as in Theorem 2.1 holds.*

Remark 3.2 Analogously to Remark 2.3 d) one can also formulate the result of Theorem 3.1 in terms of the strong version of the PAC-learning model from Definition 2.2. Furthermore on a parallel machine one can speed up the learning algorithm that is constructed in the proof of Theorem 3.1 in the same fashion as described in Remark 2.7 for the piecewise linear case.

Proof of Theorem 3.1: The only difference to the proof of Theorem 2.1 lies in the different construction of the "learning network" $\tilde{\mathcal{N}}$. One can easily see that because of the binomial formula $y \cdot z = \frac{1}{2}\left((y+z)^2 - y^2 - z^2\right)$ all high order gates in $\mathcal{N}$ can be replaced by first order gates through the introduction of new first order intermediate gates with activation function $x \mapsto x^2$. Nevertheless the construction of $\tilde{\mathcal{N}}$ is substantially more difficult compared with the construction in the preceding section. Piecewise polynomial activation functions of degree > 1 give rise to a new source of non-linearity when one tries to describe the role of the programmable parameters by a system of inequalities. Assume for example that g is a gate on level 1 with input $\alpha_1 x_1 + \alpha_2 x_2$ and activation function $\gamma^g(y) = y^2$. Then this gate g outputs $\alpha_1^2 x_1^2 + 2\alpha_1\alpha_2 x_1 x_2 + \alpha_2^2 x_2^2$. Hence the variables α_1, α_2 will not occur linearly in an inequality which describes the comparison of the output of g with some threshold of a gate at the next level. This example shows that it does not suffice to push all nontrivial weights to the first level. Instead one has to employ a more complex network construction which was introduced for a different purpose (in order to get an a-priori bound for the size of weights in the proof of Theorem 3.1 in [M 93a], see [M 93b] for a complete version).

That construction does not ensure that the output of the network architecture $\tilde{\mathcal{N}}$ is for all values of its programmable parameters contained in $[b_1, b_2]^l$ if the ranges of the activation functions of all output gates of $\mathcal{N}$ are contained in $[b_1, b_2]$. Therefore we supplement the network architecture from the proof of Theorem 3.1 in [M 93a] by adding after each output gate of that network a subcircuit which computes the function

$$z \mapsto \begin{cases} b_1 & , \text{ if } \quad z < b_1 \\ z & , \text{ if } \quad b_1 \le z \le b_2 \\ b_2 & , \text{ if } \quad z > b_2 . \end{cases}$$

This subcircuit can be realized with gates that use the heaviside activation function, gates with the activation function $x \mapsto x$, and "virtual gates" that compute the product $\langle y, z \rangle \mapsto y \cdot z$. These "virtual gates" can be realized with the help of 3 gates with activation function $x \mapsto x^2$ via the binomial formula (see above). The parameters b_1, b_2 of this subcircuit are treated like architectural parameters in the subsequent linear programming approach, since we want to keep them fixed.

Regarding the size of the resulting network architecture $\tilde{\mathcal{N}}$ we would like to mention that the number of gates in $\tilde{\mathcal{N}}$ is bounded by a polynomial in the number of gates in $\mathcal{N}$ and the number of polynomial pieces of activation functions in $\mathcal{N}$, provided that the depth of $\mathcal{N}$, the order of gates in $\mathcal{N}$, and the degrees of polynomial pieces of activation functions in $\mathcal{N}$ are bounded by a constant.

The key point of the resulting network architecture $\tilde{\mathcal{N}}$ is that for fixed network inputs the conditions on the programmable parameters of $\tilde{\mathcal{N}}$ can be expressed by *linear* inequalities, and that any function that is computable on $\mathcal{N}$ is also computable on $\tilde{\mathcal{N}}$. Apart from the different construction of $\tilde{\mathcal{N}}$ the definition and the analysis of the algorithm LEARN proceeds analogously as in the proof of Theorem 2.1. Only the parameter ρ is defined here slightly differently by $\rho := \text{size}(\tilde{\mathcal{N}}) \cdot (n+s) \cdot 3^{\text{ depth}(\tilde{\mathcal{N}})}$. If one assumes that all architectural parameters of $\mathcal{N}$ as well as b_1, b_2 are from $\mathbf{Q}_s$, one can show that any function $h : \mathbf{R}^k \to \mathbf{R}^l$ that is computable on $\mathcal{N}$ with programmable parameters form $\mathbf{Q}_s$ can be computed on $\tilde{\mathcal{N}}$ with programmable parameters from $\mathbf{Q}_{s \cdot 3^{\text{depth}(\tilde{\mathcal{N}})}}$. Furthermore any linear inequality "$s_1 < s_2$" that arises in the description of this computation of h on $\tilde{\mathcal{N}}$ for an input from $\mathbf{Q}_n^k$ (where s_1, s_2 are gate inputs, respectively thresholds) can be replaced by the stronger statement "$s_1 + 2^{-\rho} \le s_2$". This observation justifies the use of the parameter ρ in the linear programming problems that occur in the design of the algorithm LEARN. Note that in contrast to the proof of Theorem 2.1 there are no scaling factors involved in these linear programming problems (because of the different design of $\tilde{\mathcal{N}}$).

Since $\tilde{\mathcal{N}}$ contains gates with the heaviside activation function, the algorithm LEARN has to solve not only one, but polynomially in m many linear programming problems (analogously as in the proof of Theorem 2.1). ∎

4 Conclusion

It has been shown in this paper that positive theoretical results about efficient PAC-learning on neural nets are still possible, in spite of the well known negative results about learning of boolean functions with many input variables ([J], [BR], [KV]).

In the preceding negative results one had carried over the traditional asymptotic analysis of algorithms for digital computation, where one assumes that the number n of boolean input variables goes to infinity. However this analysis is not quite adequate for many applications of neural nets, where one considers a *fixed* neural net and the input is given in the form of relatively few analog inputs (e.g. sensory data). In addition for many practical applications of neural nets the number of input variables is first reduced by suitable preprocessing methods (e.g. principal component analysis). For such applications of neural nets we have shown in this paper that efficient and provably successful learning is possible, even in the most demanding refinement of the PAC-learning model. In this most realistic version of the PAC-learning model no a-priori assumptions are required about the nature of the "target function", and arbitrary noise in the input data is permitted. Furthermore this learning model is not restricted to neural nets with *boolean* output. Hence our positive learning results are also applicable to the learning resp. approximation of complicated real valued functions, such as they occur for example in process control.

This paper has introduced another idea into the theoretical analysis of learning on neural nets that promises to bear further fruits: Rather than insisting on designing an efficient learning algorithm for *every* neural net, we design learning algorithms for a subclass of neural nets $\tilde{\mathcal{N}}$ whose architecture is particularly suitable for learning. This may not be quite what we want, but it suffices as long as there are arbitrarily "powerful" network architectures $\tilde{\mathcal{N}}$ that support our learning algorithm. It is likely that this idea can be pursued further with the goal of identifying more sophisticated types of special network architectures that admit very fast learning algorithms.

Acknowledgements

I would like to thank Peter Auer, Phil Long and Hal White for their helpful comments.

References

[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

[BEHW] A. Blumer, A. Ehrenfeucht, D. Haussler, M. K. Warmuth, "Learnability and the Vapnik-Chervonenkis dimension", *J. of the ACM*, vol. 36(4), 1989, 929 - 965

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

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

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

[HKLW] D. Haussler, M. Kearns, N. Littlestone, R. E. Schapire, "Equivalence of models for polynomial learnability", *Information and Computation*, vol. 95, 1991, 129 - 161

[J] J. S. Judd, "Neural Network Design and the Complexity of Learning", *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

[KS] M. Kearns, R. E. Schapire, "Efficient distribution free learning of probabilistic concepts", *Proc. of the 31st IEEE Symposium on Foundations of Computer Science*, 1990, 382 - 391

[KSS] M. J. Kearns, R. E. Schapire, L. M. Sellie, "Toward efficient agnostic learning", *Proc. of the 5th ACM Workshop on Computational Learning Theory*, 1992, 341 - 352

[M 92] W. Maass, "Bounds for the computational power and learning complexity of analog neural nets", *IIG-Report 349 of the Technische Universität Graz*, (October 1992).

[M 93a] W. Maass, "Bounds for the computational power and learning complexity of analog neural nets" (extended abstract), *Proc. of the 25th ACM Symposium on Theory of Computing*, 1993, to appear

[M 93b] W. Maass, "Bounds for the computational power and learning complexity of analog neural nets" (Journal version), to appear

[Mi] J. Milnor, "On the Betti numbers of real varieties", *Proc. of the American Math. Soc.*, vol. 15, 1964, 275 - 280

[PS] C. H. Papadimitriou, K. Steiglitz, "Combinatorial Optimization: Algorithms and Complexity", Prentice Hall (Englewood Cliffs, 1982)

[R] J. Renegar, "On the computational complexity and geometry of the first order theory of the reals, Part I", *J. of Symbolic Computation*, vol. 13, 1992, 255 - 299

[S] J. E. Savage, "The Complexity of Computing", Wiley (New York, 1976)

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