Communicated by Nicolo Cesa-Bianchi

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

We consider learning on multilayer neural nets with piecewise polynomial activation functions and a fixed number k of numerical inputs. We exhibit arbitrarily large 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, in $1/\varepsilon$, where ε is some arbitrary given bound for the true error of the neural net after training, and in $1/\delta$ where δ is some arbitrary given bound for the probability that the learning algorithm fails for a randomly drawn training sample. However, the computation time of LEARN is exponential in the number of weights of the considered network architecture, and therefore only of interest for neural nets of small size. This article provides details to the previously published extended abstract (Maass 1994).

1 Introduction

The investigation of learning on multilayer 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 (Valiant 1984) 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 led 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 $\mathcal{N}$ (Blumer *et al.* 1989). 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 1968; Baum and Haussler 1989). 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 $\Omega(w \log w)$ (Maass 1993). 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 (1990), Blum and Rivest (1988), and Kearns and Valiant (1989) show that even for arrays $(\mathcal{N}_n)_{n \in \mathbf{N}}$ of very simple multilayer 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 multilayer 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 (i.e., numerical) inputs. We propose to investigate PAC learning on a *fixed* neural net $\mathcal{N}$, with a fixed number k of numerical 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 numerical inputs, a bound s for the allowed bit-length of weights, as well as $1/\varepsilon$, where ε is an arbitrary given bound for the true error of $\mathcal{N}$ after the training, and $1/\delta$, where δ is an arbitrary given 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 that is essentially due to Haussler (1992) and that was further developed by Kearns *et al.* (1990). 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 also investigate in this more flexible framework the learning (and approximation) of complicated real valued functions by a neural net.

In Definitions 1.1 and 1.2 we will give a precise definition of the type of neural network models that we consider in this paper: high order multilayer feedforward neural nets with piecewise polynomial activation functions.

In Definition 2.2 we will give a precise definition of the refinement of the PAC learning model that we consider in this paper. We will show in Theorem 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 (1992) that for the learning of classes of functions with nonbinary outputs the pseudo-dimension plays a role that 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, and respectively approximate, those functions or distributions that are potentially to be learned. One then goes to a somewhat larger neural net $\tilde{\mathcal{N}}$ that can simulate $\mathcal{N}$ and that has the previously mentioned special structure that 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 recipe that is frequently recommended by practitioners as a way to reduce the chance that backpropagation gets stuck in local minima: to carry out such training on a neural net that has

somewhat more units than necessary for computing the desired target functions (Rumelhart and McClelland 1986; Lippmann 1987).

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 there exist hardly any positive results for learning with interesting concept classes and function classes as hypotheses 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) lead to negative results in the here considered refinement of the PAC learning model (Kearns *et al.* 1992). Hence it is a rather noteworthy fact that function classes that are defined by arbitrarily complex analog neural nets yield positive results in this refined version of the PAC model.

One should note, however, that the asymptotic analysis that we use here for the investigation of learning on neural nets is *orthogonal* to that which underlies the quoted negative result for agnostic PAC learning with monomials (one assumes there that the number of input variables goes to infinity). Hence one should not interpret our result as saying that learning with hypotheses defined by analog neural nets is easier than learning with monomials (or other boolean formulas) as hypotheses. Our result shows that learning with a *fixed* number of *numerical* inputs is provably feasible on a multilayer neural net, whereas boolean formulas such as monomials are not suitable for dealing with numerical inputs, and it makes no sense to carry out an asymptotic analysis of learning with a fixed number of *boolean* inputs (since there exist then only *finitely* many different hypotheses).

Definition 1.1. A *network architecture* (or "neural net") $\mathcal{N}$ of order v with k input nodes and l output nodes is a labeled acyclic directed graph $\langle V, E\rangle$. It has k nodes with fan-in 0 ("input nodes") that are labeled by $1, \ldots, k$, and l nodes with fan-out 0 ("output nodes") that are labeled by $1, \ldots, l$. Each node g of fan-in $r > 0$ is called a computation node (or gate), and is labeled by some activation function $\gamma^g : \mathbf{R} \to \mathbf{R}$ and some polynomial $I^g(y_1, \ldots, y_r)$ of degree $\leq v$. We assume that the ranges of activation functions of output nodes in $\mathcal{N}$ are bounded.

The coefficients of all polynomials $I^g(y_1, \ldots, y_r)$ 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 $y_1, \ldots, y_r \in \mathbf{R}$, then g outputs $\gamma^g[I^g(y_1, \ldots, y_r)]$.

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 $t_1, \ldots, t_s \in \mathbf{R}$ and polynomials $P_0, \ldots, P_s$ such that $t_1 < \cdots < t_s$ and for each $i \in \{0, \ldots, s\} : t_i \leq x < t_{i+1} \Rightarrow \gamma(x) = P_i(x)$ (we set $t_0 := -\infty$ and $t_{s+1} := \infty$).

We refer to $t_1, \ldots, t_s$ together with all coefficients in the polynomials $P_0, \ldots, P_s$ as the *parameters of* γ. If the polynomials $P_0, \ldots, P_s$ 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 linear activation functions there exists another network architecture $\tilde{\mathcal{N}}$ that not only can 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 (i.e., the *computation nodes* of $\tilde{\mathcal{N}}$ form a tree, but there is no restriction on the fan-out of *input nodes*), 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 v with fan-out $m > 1$ is replaced by m nodes with fan-out 1, which all use the same activation function as v and which all get the same input as v. It is obvious that for this classical construction from circuit theory (Savage 1976) the depth of $\tilde{\mathcal{N}}$ is the same as the depth of $\mathcal{N}$. 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 $\sum_{i=0}^{\text{depth}(\mathcal{N})-1} \text{size}(\mathcal{N})^i \leq \text{size}(\mathcal{N})^{\text{depth}(\mathcal{N})}/(\text{size}(\mathcal{N}) - 1)$ 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 network architecture of first order (i.e., $v := 1$) 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 $m(1/\varepsilon, 1/\delta)$ 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 $m \geq m(1/\varepsilon, 1/\delta)$ 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 $\langle \underline{x}, \mathcal{N}^{\underline{\alpha}_T}(\underline{x}) + \text{noise}\rangle$, 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 (Valiant 1984). This refined version of the PAC model (essentially

due to Haussler 1992) 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 that explains the empirical data (i.e., the "sample").
- It allows for arbitrary "noise" in the sample (however, it does not attempt to remove the "noise"; instead it models the distribution including the "noise").
- 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 (and 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 Kearns *et al.* 1992). This provides the motivation for the following definition, which slightly generalizes those in Haussler (1992) and Kearns *et al.* (1992).

Definition 2.2. Let $\mathcal{A} = \cup_{n \in \mathbf{N}} \mathcal{A}_n$ 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 $\mathcal{T}$ *is efficiently learnable by* $\mathcal{H}$ *assuming* $\mathcal{A}$ if there is an algorithm LEARN and a function $m(\varepsilon, \delta, s, n)$ that is bounded by a polynomial in $1/\varepsilon$, $1/\delta$, 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 $\mathcal{H} = \cup_{s\in\mathbf{N}}\mathcal{T}_s$ we say that $\mathcal{T}$ *is properly efficiently learnable assuming* $\mathcal{A}$.

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 (Valiant 1984): 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 $C_T \in \cup_{s\in\mathbf{N}}\mathcal{C}_s$ for which

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

Then by definition $(\mathcal{T}_s)_{s\in\mathbf{N}}$ 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 Valiant (1984) (see Haussler *et al.* 1991 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 Kearns *et al.* (1992) (consider only functions with values in $\{0,1\}$ and only distributions over $\mathbf{Q}^k \times \{0,1\}$ in our preceding definition), and the model for PAC learning of probabilistic concepts from Kearns and Schapire (1990).

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 $f : \mathbf{R}^k \to \mathbf{R}^l$ 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 $\cup_{n\in\mathbf{N}}\mathcal{A}_n$. 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 $\cup_{n\in\mathbf{N}}\mathcal{A}_n$ in the sense of Definition 2.2.

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. In the definition of the VC dimension of a class $\mathcal{F}$ of $0-1$ valued functions (i.e., concepts) one says that a set S is "shattered by $\mathcal{F}$" if $\forall b \in \{0,1\}^S \ \exists f \in \mathcal{F} \ \forall x \in S \ [f(x) = b(x)]$. However, for a class $\mathcal{F}$ of *real-valued* functions f (which need not assume the values 0 or 1) one has to define in a different way that a set S is *shattered* by $\mathcal{F}$: one allows here that arbitrary "thresholds" $h(x)$ are assigned to the elements x of S. Then one can reduce the notion of "shattering" for real valued functions to that for boolean-valued functions by rounding for any $f \in \mathcal{F}$ the value $f(x)$ to 1 if $f(x) \geq h(x)$, and to 0 if $f(x) < h(x)$. Analogously as in the definition of the VC dimension one defines the pseudo-dimension of a class $\mathcal{F}$ of real valued functions as the size of the largest set S that is shattered by $\mathcal{F}$. In this way one arrives at the following definition.

Definition 2.4. (see Haussler 1992). 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

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

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 $\mathcal{F}$ (see Maass 1995a, 1995b for a survey of related results and open problems).

We will give in the following Theorem 2.5 for any network architecture $\mathcal{N}$ an upper bound for the pseudo-dimension of the class $\mathcal{F}$ of all functions f of the form

$$\langle \underline{x}, \underline{y} \rangle \mapsto ||\mathcal{N}^{\underline{\alpha}}(\underline{x}) - \underline{y}||_p$$

for arbitrary assignments $\underline{\alpha}$ to the programmable parameters of $\mathcal{N}$. Such a bound (for the network architecture $\tilde{\mathcal{N}}$) will be essential for the proof of Theorem 2.1, since it allows us to bound with the help of "uniform convergence results" due to Pollard (1990) and Haussler (1992) [see the subsequent inequality 2.1] the number of random examples that are needed to train $\tilde{\mathcal{N}}$. Thereby one can reduce the computation of a suitable assignment $\tilde{\underline{\alpha}}$ to the programmable parameters of $\tilde{\mathcal{N}}$ to a *finite* optimization problem. Or in other words: instead of minimizing the "true error" of $\tilde{\mathcal{N}}^{\tilde{\underline{\alpha}}}$ it can be shown to be enough to minimize the "apparent error" of $\tilde{\mathcal{N}}^{\tilde{\underline{\alpha}}}$ on a "sufficiently large" training set, where "sufficiently large" is specified by the bound $m(1/\varepsilon, 1/\delta)$ in terms of the pseudo-dimension of the associated function class $\mathcal{F}$ at the beginning of the subsequent proof of Theorem 2.1.

Theorem 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*

$$\mathcal{F} := \left\{ f : \mathbf{R}^{k+l} \to \mathbf{R} \ : \exists \underline{\alpha} \in \mathbf{R}^w \ \forall \underline{x} \in \mathbf{R}^k \ \forall \underline{y} \in \mathbf{R}^l [f(\underline{x}, \underline{y}) = ||\mathcal{N}^{\underline{\alpha}}(\underline{x}) - \underline{y}||_p] \right\}.$$

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 $\leq (2vd)^w$. Each conjunction in this DNF formula Φ_i describes one "guess" regarding which of the $\leq q$ polynomial pieces of each activation function γ^g of gates g in $\mathcal{N}$ are used in the computation of $\mathcal{N}^{\underline{\alpha}}(\underline{x}_i)$ for the considered (fixed) network input $\underline{x}_i$. Obviously there are at most $q^{(\text{number of gates in } \mathcal{N})} \leq q^w$ different "guesses" of this type possible. In addition each of these conjunctions also describes a "guess" regarding which of the l output gates of $\mathcal{N}$ yield in the computation of $\mathcal{N}^{\underline{\alpha}}(\underline{x}_i)$, a number which is larger or equal to the corresponding component of the fixed "target output" $\underline{y}_i$. There are 2^l different possibilities for that. Thus altogether Φ_i consists of at most $q^w \cdot 2^l$ conjunctions.

The atomic formulas of each of these $\leq q^w \cdot 2^l$ conjunctions of Φ_i consist of all associated comparisons with thresholds of the activation functions. More precisely, one has in each conjunction of Φ_i for each gate g in $\mathcal{N}$ two atomic formulas that compare the value of the term $I^g(y_1, \ldots, y_r)$ (this is the term to which the activation function γ^g of gate g is applied for the presently considered network input $\underline{x}_i$) with two consecutive thresholds of the activation function γ^g. These two thresholds are the boundaries of that interval in the domain of the piecewise polynomial function γ^g where γ^g is defined as that polynomial piece that is "guessed" in this conjunction, and that is used in other atomic formulas of the same conjunction of Φ_i to specify the arguments of activation function of *subsequent* gates in $\mathcal{N}$ (for the same network input $\underline{x}_i$). In addition for each output gate of $\mathcal{N}$ one has an atomic formula that expresses that the output value of that gate is above (respectively below) the corresponding coordinate of the "target output" $\underline{y}_i$, as specified by the "guess" that is associated with this conjunction. Thus altogether each conjunction of Φ_i expresses that its associated collection of "guesses" is consistent with the actual definitions of the activation functions in $\mathcal{N}$. One exploits here that for the considered computation of $\mathcal{N}(\underline{x}_i)$ the actual input to each activation function γ^g can be written as a polynomial in terms of the coordinates of $\underline{\alpha}$ (and various constants, such as the architectural parameters of $\mathcal{N}$ and

the coordinates of $\underline{x}_i$), provided one "knows" which pieces of the activation functions of preceding gates were used for this computation. The factor 2 in the degree bound arises only in the case of piecewise rational activation functions.

By definition one has $\Phi_i(\underline{\alpha}_b)$ is true if and only if $b(i) = 1$ 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 $i \in \{1, \ldots, D\}$ so that $\Phi_i(\underline{\alpha}_b)$ and $\Phi_i(\underline{\alpha}_{\tilde{b}})$ have different truth values. 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 $\Phi_1, \ldots, \Phi_D$ 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 (1964) (see also Renegar 1992) 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 Maass (1992, 1993) (for bounded depth) and Goldberg and Jerrum (1993) (for unbounded depth). The preceding proof generalizes the argument from Goldberg and Jerrum (1993).

Proof of Theorem 2.1. We associate with $\mathcal{N}$ another network architecture $\tilde{\mathcal{N}}$ as defined before Theorem 2.1. Assume that $\mathcal{N}$ has w weights, and let $\tilde{w}$ be the number of weights in $\tilde{\mathcal{N}}$. By construction any function that is computable by $\mathcal{N}$ can also be computed by $\tilde{\mathcal{N}}$.

We first reduce with the help of Theorem 2.5 the computation of appropriate weights for $\tilde{\mathcal{N}}$ to a *finite* optimization problem. 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

$$\begin{aligned} b &:= l \cdot (b_2 - b_1), \text{ and} \\ \mathcal{F} &:= \{f : \mathbf{R}^k \times [b_1, b_2]^l \to [0, b] : \\ &\quad \exists \underline{\alpha} \in \mathbf{R}^{\tilde{w}} \, \forall \underline{x} \in \mathbf{R}^k \, \forall \underline{y} \in [b_1, b_2]^l \, [f(\underline{x}, \underline{y}) = ||\tilde{\mathcal{N}}^{\underline{\alpha}}(\underline{x}) - \underline{y}||_1]\} \end{aligned}$$

The preceding Theorem 2.5 implies that the pseudo-dimension $\dim_P(\mathcal{F})$ of this class $\mathcal{F}$ is finite. Therefore one can derive a finite upper bound for the minimum size of a training set for the considered learning problem in the following way. Assume that parameters $\varepsilon, \delta \in (0,1)$ with $\varepsilon \leq b$ 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

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

By Corollary 2 of Theorem 7 in Haussler (1992) one has for $m \geq m(1/\varepsilon, 1/\delta)$, $K := \sqrt{257}/8 \in (2,3)$, and any distribution A over $\mathbf{Q}_n^k \times (\mathbf{Q}_n \cap [b_1, b_2])^l$

$$\Pr_{\zeta \in A^m}\left[\left\{\exists f \in \mathcal{F} : \left|\left(\frac{1}{m}\sum_{\langle \underline{x},\underline{y}\rangle \in \zeta} f(\underline{x},\underline{y})\right) - E_{\langle \underline{x},\underline{y}\rangle \in A}[f(\underline{x},\underline{y})]\right| > \frac{\varepsilon}{K}\right\}\right] \leq \delta \quad (2.1)$$

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 $\tilde{\underline{\alpha}}$ of rational numbers to the parameters in $\tilde{\mathcal{N}}$ such that the function $\tilde{h}$ that is computed by $\tilde{\mathcal{N}}^{\tilde{\underline{\alpha}}}$ satisfies

$$\frac{1}{m}\sum_{i=1}^{m} ||\tilde{h}(\underline{x}_i) - \underline{y}_i||_1 \leq \left(1 - \frac{2}{K}\right)\varepsilon + \inf_{\underline{\alpha} \in \mathbf{Q}_s^w} \frac{1}{m}\sum_{i=1}^{m} ||\mathcal{N}^{\underline{\alpha}}(\underline{x}_i) - \underline{y}_i||_1 \quad (2.2)$$

It suffices for the proof of Theorem 2.1 to solve this finite optimization problem, since 2.1 and 2.2 (together with the fact that the function $\langle \underline{x}, \underline{y}\rangle \mapsto ||\mathcal{N}^{\underline{\alpha}}(\underline{x}) - \underline{y}||_1$ is in $\mathcal{F}$ for every $\underline{\alpha} \in \mathbf{R}^w$) imply that, for any distribution A over $\mathbf{Q}_n^k \times (\mathbf{Q}_n \cap [b_1, b_2])^l$ and any $m \geq m(1/\varepsilon, 1/\delta)$, with probability $\geq 1 - \delta$ (with respect to the random drawing of $\zeta \in A^m$) the algorithm LEARN outputs for inputs ζ and s an assignment $\tilde{\underline{\alpha}}$ of rational numbers to the parameters in $\tilde{\mathcal{N}}$ such that

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

We have now reduced the problem of computing appropriate weigths for $\tilde{\mathcal{N}}$ to the *finite* optimization problem 2.2 for the algorithm LEARN. However it turns out that this finite optimization problem is highly nonlinear, and hence has no readily available algorithmic solution. In the remainder of this proof we show how this finite nonlinear optimization problem can be reduced to linear programming. More precisely, 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 $\tilde{\underline{\alpha}}$ for the programmable parameters in $\tilde{\mathcal{N}}$ so that $\tilde{\mathcal{N}}^{\tilde{\underline{\alpha}}}$ satisfies 2.2. The reduction of the computation of $\tilde{\underline{\alpha}}$ 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 (and since each activation function is applied in the here considered case $v := 1$ to a term that is *linear* in the weights of the respective gate). 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 Maass (1993) 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 $\hat{\mathcal{N}}[\underline{c}]$. By this, all variables that appear in the inequalities 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 inequalities.

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 $\tilde{\mathcal{N}}$ 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 $\tilde{\underline{c}}, \tilde{\underline{\beta}}$ to the parameters in $\hat{\mathcal{N}}$ with $c > 0$ for all c in $\tilde{\underline{c}}$ an assignment $\tilde{\underline{\alpha}}$ to the programmable parameters in $\tilde{\mathcal{N}}$ such that $\forall \underline{x} \in \mathbf{R}^k [\tilde{\mathcal{N}}^{\tilde{\underline{\alpha}}}(\underline{x}) = \hat{\mathcal{N}}[\tilde{\underline{c}}]^{\tilde{\underline{\beta}}}(\underline{x})]$. This backward transformation is also defined by recursion. Consider some gate g on level 1 in $\hat{\mathcal{N}}$ that uses (for the new parameter assignment $\tilde{\underline{c}}$) 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 $\hat{\mathcal{N}}[\tilde{\underline{c}}]^{\tilde{\underline{\beta}}}$ by $\alpha_1/c, \ldots, \alpha_k/c, \alpha_0/c$; 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 $\hat{\mathcal{N}}[\tilde{\underline{c}}]^{\tilde{\underline{\beta}}}$ (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 $\tilde{\underline{\beta}}$, and that $c' > 0$ is the scaling parameter of g' in $\hat{\mathcal{N}}[\tilde{\underline{c}}]^{\tilde{\underline{\beta}}}$. Then we assign the new

weights $\alpha_1'/c', \ldots, \alpha_q'/c'$ and the new bias α_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 2.2. 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 consists 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^g 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^g 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 in the last paragraph of this proof. 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 $2^{\tilde{w}'} \leq 2^{\tilde{w}}$ different architectures, where $\tilde{w}'$ is the number of edges between computation nodes of $\hat{\mathcal{N}}$ [thus $\tilde{w}' \leq$ (number of gates in $\tilde{\mathcal{N}} =: \tilde{v}$), and $\tilde{w}$ 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 $2^{\tilde{v}} = O(1)$ ways of assigning the weight 0 to certain edges between computation nodes in $\tilde{\mathcal{N}}$, and correspondingly there are at most $2^{\tilde{v}}$ 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 $\hat{\mathcal{N}}$. There-

fore 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 $t \leq s^g$ and $s^g + 2^{-\rho} \leq t'$ 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 to describe for this network input $\underline{x}_i$ and for all subsequent gates g' the contribution of gate g to 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^g, 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 $H_1 := \{\underline{x} \in \mathbf{R}^k : \underline{x} \cdot \hat{\underline{\alpha}} + 2^{-\rho} \leq t\}$ and $H_2 := \{\underline{x} \in \mathbf{R}^k : \underline{x} \cdot \hat{\underline{\alpha}} \geq t\}$, where the weights $\hat{\underline{\alpha}}$ for gate g are chosen in such a way that $\underline{x}_i \cdot \hat{\underline{\alpha}} + 2^{-\rho} = t$ for all $i \in S_1$ and $\underline{x}_i \cdot \hat{\underline{\alpha}} = t$ 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 $\leq 2^k \cdot \binom{m}{k}$ 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 $\hat{\underline{\alpha}}$ 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 $\{\underline{x}_1, \ldots, \underline{x}_m\}$ that 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 rather 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. Even if in each of the $2^{k'}$ subsets one only has to consider two possibilities for the hyperplane that is defined by threshold t' of g', one arrives altogether at $2^{2^{k'}}$ possibilities for this hyperplane. Thus the straightforward estimate for the number of different systems L_j yields an upper bound that is *double-exponential* in the size of $\hat{\mathcal{N}}$. 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.

To get a better estimate for the number of systems L_j we exploit that the input to the considered gate g' is not only piecewise linear in the coordinates of the input, but also piecewise linear in the weights for gates on level 1 and in the scaling factors of preceding gates. Hence we now view these weights and scaling factors as *variables* in the expression that describes the input to gate g'. The number of these variables can be bounded by the number $\tilde{w}$ of variables in $\hat{\mathcal{N}}$. The coefficients of these variables in the input to gate g' consist of the coordinates of the network input and of the fixed parameters a_i, b_i of the linear pieces $x \mapsto a_i x + b_i$ of the activation functions of preceding gates. We may assume that one has fixed for each of the preceding gates *which* linear piece of *each* activation function is applied for *each* of the m fixed network inputs. Hence each of these m network inputs defines a unique vector of $\tilde{w}$ coefficients for the $\tilde{w}$ "new variables" in the input to gate g'. The set of these m coefficient vectors from $\mathbf{R}^{\tilde{w}}$ can be partitioned in at most $O(m^{\tilde{w}})$ different ways by a pair of hyperplanes in $\mathbf{R}^{\tilde{w}}$ with distance $2^{-\rho}$. Hence we have to consider only $O(m^{\tilde{w}})$ possibilities for the choice of the subset of the m network inputs for which the input to gate g' is less than t'. We would like to emphasize that we consider here the weights on level 1 and the scaling factors as *variables*, and we examine the effect on the values of the input to gate g' for the considered m network inputs if we change these variables. It will be justified in the last paragraph of the proof of Theorem 2.1 that we may assume that the input to gate g' has at least distance $2^{-\rho}$ from t' if its value lies below this threshold t'.

The preceding argument yields an upper bound of $m^{O(\tilde{w}\cdot\tilde{v})}$ for the number of linear systems L_j that arise by considering all possibilities for using different linear pieces of the activation functions for the m fixed

network inputs (where $\tilde{v}$ denotes the number of computation nodes in $\hat{\mathcal{N}}$). In addition one has to consider $3^{\tilde{v}}$ different choices of weights from $\{-1,0,1\}$ for the gates on level >1 in $\hat{\mathcal{N}}$. Thus altogether at most $m^{O(\tilde{w}\cdot\tilde{v})}$ different systems L_j of linear inequalities have to be considered.

Hence the algorithm LEARN generates for each of the polynomially in m many partitions of $\underline{x}_1,\ldots,\underline{x}_m$ that 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 (if the size of $\mathcal{N}$ can be viewed as a constant).

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 (it should be noted that it is essential that these $\geq m$ additional variables were not yet present in our preceding consideration, since otherwise we would have arrived at *exponentially* in m many systems of linear inequalities L_j). 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

$$t_j^\nu(\underline{x}_i) \leq (\underline{y}_i)_\nu + u_i^\nu - v_i^\nu, \quad t_j^\nu(\underline{x}_i) \geq (\underline{y}_i)_\nu + u_i^\nu - v_i^\nu,$$
$$u_i^\nu \geq 0, \qquad v_i^\nu \geq 0$$

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 $\underline{y}_i$ that 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 Papadimitriou and Steiglitz 1982) to compute in altogether polynomially in m, s, and n many steps an optimal solution for each of the linear programming problems $LP_1,\ldots,LP_{p(m)}$. Note that we assume that s is sufficiently large so that all architectural parameters of $\mathcal{N}$ (respectively $\hat{\mathcal{N}}$) are from $\mathbf{Q}_s$.

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 $(1/m)\sum_{i=1}^m ||h_j(\underline{x}_i) - \underline{y}_i||_1$ for $j=1,\ldots,p(m)$. Let $\tilde{j}$ be that index for which

this expression has a minimal value. Let $\tilde{\underline{c}}, \tilde{\underline{\beta}}$ be the associated optimal solution of LP_j (i.e., $\hat{\mathcal{N}}[\tilde{\underline{c}}]^{\tilde{\underline{\beta}}}$ computes h_j). LEARN employs the previously described backward transformation from $\tilde{\underline{c}}, \tilde{\underline{\beta}}$ into values $\tilde{\underline{\alpha}}$ for the programmable parameters of $\tilde{\mathcal{N}}$ such that $\forall \underline{x} \in \mathbf{R}^k[\tilde{\mathcal{N}}^{\tilde{\underline{\alpha}}}(\underline{x}) = \hat{\mathcal{N}}[\tilde{\underline{c}}]^{\tilde{\underline{\beta}}}(\underline{x})]$. These values $\tilde{\underline{\alpha}}$ are given as output of the algorithm LEARN.

We will show that $\tilde{h} := h_{\tilde{j}}$ satisfies condition 2.2, i.e.,

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

In fact we will show that $\tilde{h}$ satisfies the stronger inequality, where $[1 - (2/K)] \cdot \varepsilon$ is replaced by 0. Fix some $\underline{\alpha}' \in \mathbf{Q}_s^w$ with

$$\frac{1}{m}\sum_{i=1}^{m} ||\mathcal{N}^{\underline{\alpha}'}(\underline{x}_i) - \underline{y}_i||_1 = \inf_{\underline{\alpha} \in \mathbf{Q}_s^w} \frac{1}{m}\sum_{i=1}^{m} ||\mathcal{N}^{\underline{\alpha}}(\underline{x}_i) - \underline{y}_i||_1 \tag{2.3}$$

Let $\tilde{\underline{\alpha}'}$ consist of corresponding values from $\mathbf{Q}_s$ such that $\forall \underline{x} \in \mathbf{R}^k[\mathcal{N}^{\underline{\alpha}'}(\underline{x}) = \tilde{\mathcal{N}}^{\tilde{\underline{\alpha}'}}(\underline{x})]$. According to the previously described construction one can transform $\tilde{\underline{\alpha}'}$ into parameters $\underline{c}, \underline{\beta}$ from $\mathbf{Q}_{s \cdot \mathrm{depth}(\tilde{\mathcal{N}})}$ such that $\forall \underline{x} \in \mathbf{R}^k[\tilde{\mathcal{N}}^{\tilde{\underline{\alpha}'}}(\underline{x}) = \hat{\mathcal{N}}[\underline{c}]^{\underline{\beta}}(\underline{x})]$. 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 $\tilde{\underline{\alpha}'}$, 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 $\rho := 2[s \cdot \mathrm{size}(\mathcal{N})]^{\mathrm{depth}(\mathcal{N})-1} \cdot [s^2 \cdot \mathrm{depth}(\mathcal{N}) \cdot (k+2) \cdot n]$. 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 $\tilde{\underline{\alpha}'}$ 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 $\sum_{i=1}^{m} \sum_{\substack{\nu \text{ output} \\ \text{node in } \hat{\mathcal{N}}}} (u_i^\nu + v_i^\nu)$ 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 2.2 follows from 2.3. 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. The numbers of processors that

are needed can be bounded by $m^{O(\tilde{w}\cdot\tilde{v})} \cdot \text{poly}(n,s)$. 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 "competitive 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 $\tilde{\alpha}$ 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 competitive 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 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 interesting heuristic learning algorithm. Such a "scaled-down" version of LEARN is no longer guaranteed to find probably an approximately optimal weight setting in the strict sense of the PAC learning model. However, it may provide satisfactory performance for a real world learning problem in case that not only a single one, but a certain fraction of all linear programming problems LP_j, yields for this learning problem a satisfactory solution. One may compare this heuristic consideration with the somewhat analogous situation for backpropagation, where one hopes that for a certain fraction of randomly chosen initial settings of the weights one is reasonably close to a global minimum of the error function.

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 $y \cdot z = \frac{1}{2}[(y+z)^2 - y^2 - z^2]$ 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 nonlinearity 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 that 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 that was introduced for a different purpose (it had been introduced to get an a priori bound for the size of weights in the proof of Theorem 3.1 in Maass 1993; see Maass 1995c 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 Maass (1993) by adding after each output gate of that network a subcircuit that computes the function

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

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 proceed 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 from $\mathbf{Q}_s$ 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 and thresholds, respectively) can be replaced by the stronger statement "$s_1 + 2^{-\rho} \leq s_2$." 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 (Judd 1990; Blum and Rivest 1988; Kearns and Valiant 1989).

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. 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 and approximation of complicated real valued functions, such as they occur, for example, in process control.

The proofs of the main theorems of this paper (Theorems 2.1 and 3.1) employ rather sophisticated results from statistics and algebraic geometry to provide a bound not just for the *apparent error* (i.e., the error on the training set) of the trained neural net, but also for its *true error* (i.e., its error on *new* examples from the same distribution). In addition, these positive learning results employ rather nontrivial variable transformation techniques to reduce the nonlinear optimization problem for the weights of the considered multilayer neural nets to a family of linear programming problems. The new learning algorithm LEARN that we introduce solves all of these linear programming problems, and then takes their best solution to compute the desired assignment of weights for the trained neural net.

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.

Acknowledgments

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

References

Baum, E. B., and Haussler, D. 1989. What size net gives valid generalization? *Neural Comp.* **1**, 151–160.

Blum, A., and Rivest, R. L. 1988. Training a 3-node neural network is NP-complete. In *Proceedings of the 1988 Workshop on Computational Learning Theory*, pp. 9–18. Morgan Kaufmann, San Mateo, CA.

Blumer, A., Ehrenfeucht, A., Haussler, D., and Warmuth, M. K. 1989. Learnability and the Vapnik-Chervonenkis dimension. *J. ACM* **36**(4), 929–965.

Cover, T. M. 1968. Capacity problems for linear machines. In *Pattern Recognition*, L. Kanal, ed., pp. 283–289. Thompson Book Co., Washington, DC.

Goldberg, P., and Jerrum, M. 1993. Bounding the Vapnik-Chervonenkis dimension of concept classes parameterized by real numbers. *Proc. of the 6th Annual ACM Conference on Computational Learning Theory* 361–369. ACM-Press, New York, NY.

Haussler, D. 1992. Decision theoretic generalizations of the PAC model for neural nets and other learning applications. *Inform. Comp.* **100**, 78–150.

Haussler, D., Kearns, M., Littlestone, N., and Schapire, R. E. 1991. Equivalence of models for polynomial learnability. *Inform. Comp.* **95**, 129–161.

Judd, J. S. 1990. *Neural Network Design and the Complexity of Learning*. MIT Press, Cambridge, MA.

Kearns, M., and Schapire, R. E. 1990. Efficient distribution free learning of probabilistic concepts. *Proc. 31st IEEE Symp. Foundations Comp. Sci.* 382–391.

Kearns, M., and Valiant, L. 1989. Cryptographic limitations on learning boolean formulae and finite automata. *Proc. 21st ACM Symp. Theory Comp.* 433–444.

Kearns, M. J., Schapire, R. E., and Sellie, L. M. 1992. Toward efficient agnostic learning. *Proc. 5th ACM Workshop Comp. Learning Theory* 341–352.

Lippmann, R. P. 1987. An introduction to computing with neural nets. *IEEE ASSP Mag.* 4–22.

Maass, W. 1992. *Bounds for the Computational Power and Learning Complexity of Analog Neural Nets*. IIG-Report 349, Technische Universität Graz.

Maass, W. 1993. Bounds for the computational power and learning complexity of analog neural nets (extended abstract). *Proc. 25th ACM Symp. Theory Comput.* 335–344.

Maass, W. 1994. Agnostic PAC-learning of functions on analog neural nets (extended abstract). In *Advances in Neural Information Processing Systems*, Vol. 6, pp. 311–318. Morgan Kaufmann, San Mateo, CA.

Maass, W. 1995a. Perspectives of current research about the complexity of learning on neural nets. In *Theoretical Advances in Neural Computation and Learning*, 295–336, V. P. Roychowdhury, K. Y. Siu, and A. Orlitsky, eds. Kluwer Academic Publishers, Boston.

Maass, W. 1995b. Vapnik-Chervonenkis Dimension of Neural Nets. In *Handbook of Brain Theory and Neural Networks*, M. A. Arbib, ed. MIT Press, Cambridge, MA (in press).

Maass, W. 1995c. Computing on analog neural nets with arbitrary real weights. In *Theoretical Advances in Neural Computation and Learning*, V. P. Roychowdhury, K. Y. Siu, and A. Orlitsky, eds., pp. 153–172. Kluwer Academic Publishers, Boston, MA.

Milnor, J. 1964. On the Betti numbers of real varieties. *Proc. Am. Math. Soc.* **15**, 275–280.

Papadimitriou, C. H., and Steiglitz, K. 1982. *Combinatorial Optimization: Algorithms and Complexity*. Prentice Hall, Englewood Cliffs, NJ.

Pollard, D. 1990. Empirical Processes: Theory and Applications. *NSF-CBMS Regional Conf. Ser. Prob. Statist. 2.*

Renegar, J. 1992. On the computational complexity and geometry of the first order theory of the reals, Part I. *J. Symbolic Comp.* **13**, 255–299.

Rumelhart, D. E., and McClelland, J. L. 1986. *Parallel Distributed Processing: Exploration in the Microstructure of Cognition, Vol. 1: Foundations.* MIT Press, Cambridge, MA.

Savage, J. E. 1976. *The Complexity of Computing*. Wiley, New York.

Valiant, L. G. 1984. A theory of the learnable. *Commun. ACM* **27**, 1134–1142.

Received May 14, 1994; accepted November 9, 1994.