On the Complexity of Learning on 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

The goal of this paper is to present in a heuristic manner the proof ideas for two recent results on the complexity of learning on neural nets:

- Superlinear lower bounds for the VC-dimension of neural nets
- Agnostic PAC-learning on small analog neural nets.

We present here a slightly edited and expanded version of the transparencies from an invited talk at Euro-COLT '93. The complete proofs are technically rather involved, and they will appear elsewhere ([M 93c], [M 93b]). We refer to [ORS] for further introductory material and overviews regarding the complexity of learning on neural nets.

1 SUPERLINEAR LOWER BOUNDS FOR THE VC-DIMENSION OF NEURAL NETS

The results of [BEHW] (see also [AB]) imply that the key parameter for determining the sample-complexity for PAC-learning on a neural net $\mathcal{N}$ is its Vapnik-Chervonenkis dimension ("VC-dimension"). This parameter is defined as follows.

Definition 1. *Consider a neural net $\mathcal{N}$ with domain X (e.g. $X = \{0,1\}^d$, $\boldsymbol{Q}^d$, $\boldsymbol{R}^d$), boolean output, and w "weights" from W (e.g. $W = \boldsymbol{N}, \boldsymbol{Q}, \boldsymbol{R}$). A set $T \subseteq X$ is <u>shattered by $\mathcal{N}$</u> if*

$$\forall\, f : T \rightarrow \{0,1\}\ \exists\, \underline{\alpha} \in W^w\ \forall\, \underline{x} \in T(f(\underline{x}) = \mathcal{N}^{\underline{\alpha}}(\underline{x})).$$

$$\textit{VC-dimension}\ (\mathcal{N}) := \sup\{|T| : T \subseteq X \textit{ is shattered by } \mathcal{N}\}.$$

We consider in this section primarily neural nets $\mathcal{N}$ that consist of linear threshold gates. One often refers to the programmable parameters of $\mathcal{N}$ (i.e. the weights and biases of its gates) simply as the *weights of* $\mathcal{N}$.

The VC-dimension of a single *linear threshold gate* of fan-in w is equal to $w+1$ ([WD]). Tom Cover had shown in 1964 ([C 64], see also [C 68]) that *any neural net* consisting of linear threshold gates with altogether w *weights* has VC-dimension $O(w \cdot \log w)$. On the other hand the best known *lower bound* for the VC-dimension of a neural net with w weights was w (which is trivially achieved by a neural net with just one unit, by [WD]). This gave rise to the following interesting question:

Can the VC-dimension of a neural net be substantially larger than the sum of the VC-dimensions of its units? (Note that for a neural net that consists of linear threshold gates the sum of the VC-dimensions of its units is equal to its total number w of weights.)

The following result shows that the superlinear upper bound of Cover is in fact asymptotically optimal. This implies that in a larger neural net a single weight contributes (on average) more than a constant to the VC-dimension of the neural net. In fact, its average contribution can be as large as $\Omega(\log w)$, and hence increase with the total size of the neural net. Therefore one may interpret the following result as mathematical evidence for a certain type of "connectionism thesis": that a network of neuron-like elements is more than just the sum of its components.

Theorem 2. *([M 93c], see also [M 93a])*
Assume that $(\mathcal{N}_d)_{d\in\mathbf{N}}$ *is a sequence of neural nets of depth* ≥ 3, *where* $\mathcal{N}_d$ *has* d *boolean input nodes and* $O(d)$ *gates.*

Furthermore assume that $\mathcal{N}_d$ *has* $\Omega(d)$ *gates on the first hidden layer, and at least* $4\log d$ *gates on the second hidden layer. We also assume that* $\mathcal{N}_d$ *is fully connected between any two successive layers (hence* $\mathcal{N}_d$ *has* $\Theta(d^2)$ *programmable parameters or "weights"), and that the gates of* $\mathcal{N}_d$ *are linear threshold gates (or gates with the sigmoid activation function* $\sigma(y)=\frac{1}{1+e^{-y}}$, *with round-off at the network output).*

Then VC-dimension$(\mathcal{N}_d)=\Theta(d^2\cdot\log d)$, *hence VC-dimension*$(\mathcal{N}_d)=\Theta(w\log w)$ *in terms of the number* w *of weights of* $\mathcal{N}_d$.

The **proof** of Theorem 2 proceeds by constructing a particular sequence $(\mathcal{M}_d)_{d\in\mathbf{N}}$ of neural nets with superlinear VC-dimension. It is easy to show that these nets $(\mathcal{M}_d)_{d\in\mathbf{N}}$ can be embedded into arbitrary given nets $(\mathcal{N}_d)_{d\in\mathbf{N}}$ with the properties from Theorem 2. This implies that the $\mathcal{N}_d$ also have superlinear VC-dimension.

Assume that d is some arbitrary power of 2. We construct a neural net $\mathcal{M}_d$ of depth 3 with $2d+\log d$ input nodes and $\leq 17d^2$ edges such that VC-dimension $(\mathcal{M}_d)\geq d^2\cdot\log d$. This construction uses methods due to Neciporuk [N] and Lupanov [L].

We construct $\mathcal{M}_d$ so that it shatters the set

$$T := \{\underline{e}_p \underline{e}_q \underline{\tilde{e}}_m : p, q \in \{1, \ldots, d\}, m \in \{1, \ldots, \log d\}\} \subseteq \{0,1\}^{2d+\log d},$$

where $\underline{e}_p, \underline{e}_q$ denote unit vectors of length d and $\underline{\tilde{e}}_m$ denotes a unit vector of length $\log d$ (thus every $\underline{x} \in T$ contains exactly three "1", one in each of the three blocks of length d, d, and $\log d$).

Fix some arbitrary map $F : T \to \{0,1\}$. We construct a neural net $\mathcal{M}_d$ that computes F in such a way that only the values of the weights $w_{i,q}$ in $\mathcal{M}_d$ (and not the architecture of $\mathcal{M}_d$) depend on this particular function F. One encodes F by a function $g : \{1, \ldots, d\}^2 \to \{0,1\}^{\log d}$ by setting

$$g(p,q) := \langle F(\underline{e}_p \underline{e}_q \underline{\tilde{e}}_1), \ldots, F(\underline{e}_p \underline{e}_q \underline{\tilde{e}}_{\log d})\rangle.$$

For simplicity we first assume that $g(\cdot, q)$ is 1-1 for every $q \in \{1, \ldots, d\}$. Then $g(\cdot, q)$ is invertible and we can define for $q \in \{1, \ldots, d\}$ and $i \in \{0, \ldots, d-1\}$ the weights $w_{i,q}$ by

$$w_{i,q} = p :\Leftrightarrow g(p,q) = \text{bin}(i),$$

where $\text{bin}(i) \in \{0,1\}^{\log d}$ denotes the binary representation of $i \in \{0, \ldots, d-1\}$.

In order to illustrate the construction principle of $\mathcal{M}_d$ we first assume that some $b \in \{1, \ldots, \log d\}$ *has been fixed.* By definition of g one has

$$F(\underline{e}_p \underline{e}_q \underline{\tilde{e}}_b) = 1 \Leftrightarrow (g(p,q))_b = 1 \Leftrightarrow$$
$$\exists\, i \in \{0, \ldots, d-1\}((\text{bin}(i))_b = 1 \wedge g(p,q) = \text{bin}(i)),$$

where $(\underline{x})_b$ denotes the b-th bit of any bit-string $\underline{x}$. The network $\mathcal{M}_d$ employs linear threshold gates G_i^+, G_i^- on level 1, which are defined by the condition

$$G_i^+(\underline{e}_p, \underline{e}_q) = 1 \Leftrightarrow \sum_{r=1}^{d} r \cdot (\underline{e}_p)_r \geq \sum_{r=1}^{d} w_{i,r} \cdot (\underline{e}_q)_r$$
$$G_i^-(\underline{e}_p, \underline{e}_q) = 1 \Leftrightarrow \sum_{r=1}^{d} r \cdot (\underline{e}_p)_r \leq \sum_{r=1}^{d} w_{i,r} \cdot (\underline{e}_q)_r.$$

The term $(\underline{e}_p)_r$ has value 1 if and only if $p = r$, hence $\sum_{r=1}^{d} r \cdot (\underline{e}_p)_r = p$ and $\sum_{r=1}^{d} w_{i,r} \cdot (\underline{e}_q)_r = w_{i,q}$. It is obvious that for any values of p, q, i at least one of the two gates G_i^+, G_i^- gives output 1 for input $\underline{e}_p, \underline{e}_q$. Furthermore *both* gates give output 1 for input $\underline{e}_p, \underline{e}_q$ if and only if $w_{i,q} = p$, i.e. $g(p,q) = \text{bin}(i)$. Hence a threshold gate on level 2 of $\mathcal{M}_d$ that outputs 1 whenever

$$\sum_{\substack{i=0 \\ \text{with } (\text{bin}(i))_b = 1}}^{d-1} G_i^+(\underline{e}_p, \underline{e}_q) + G_i^-(\underline{e}_p, \underline{e}_q) \geq \frac{d}{2} + 1$$

can be used to check whether $\exists\, i \in \{0,\ldots,d-1\}((\mathrm{bin}(i))_b = 1 \wedge g(p,q) = \mathrm{bin}(i))$, which is equivalent to $F(\underline{e}_p \underline{e}_q \underline{\tilde{e}}_b) = 1$.

In the general case when b *is a variable*, one uses for each possible value $b \in \{1,\ldots,\log d\}$ a separate circuit of depth 2 as described before, which simultaneously checks whether $b = m$ for the last block $\underline{\tilde{e}}_m$ of the input $\underline{e}_p \underline{e}_q \underline{\tilde{e}}_m$. This yields a circuit of depth 3 that gives output 1 if and only if $F(\underline{e}_p \underline{e}_q \underline{\tilde{e}}_m) = 1$.

Finally we have to remove the simplifying assumption that $g(\cdot, q)$ is 1-1 for every $q \in \{1,\ldots,d\}$. According to [N], [L] there exist for *any* function $g : \{1,\ldots,d\}^2 \to \{0,1\}^{\log d}$ four auxiliary functions $g_1, g_2, g_3, g_4 : \{1,\ldots,d\}^2 \to \{0,1\}^{\log d}$ such that $g_j(\cdot, q)$ is 1-1 for every $q \in \{1,\ldots,d\}$ and every $j \in \{1,\ldots,4\}$, and such that

$$g(p,q) = \begin{cases} g_1(p,q) \oplus g_2(p,q)\ , & \text{if} \quad p \le d/2 \\ g_3(p,q) \oplus g_4(p,q)\ , & \text{if} \quad p > d/2 \end{cases}$$

(where $\oplus$ denotes a bitwise EXCLUSIVE OR). One can construct in the previously described way for $j = 1,\ldots,4$ separate threshold circuits of depth 3 that check whether $(g_j(p,q))_b = 1$ (respectively whether $(g_j(p,q))_b = 0$), using the fact that $g_j(\cdot, q)$ is 1-1 for every $q \in \{1,\ldots,d\}$. It is not very difficult to combine these circuits into a single network of depth 3 that checks whether $(g(p,q))_m = 1$, i.e. whether $F(\underline{e}_p \underline{e}_q \underline{\tilde{e}}_m) = 1$.

It is obvious from the construction that the architecture of the resulting network $\mathcal{M}_d$ is independent of the specific function $F : T \to \{0,1\}$. Hence $\mathcal{M}_d$ has VC-dimension $\ge 2d + \log d$.

We refer to [M 93c] for further details. ∎

Subsequently Sakurai [S] has shown that if one allows *real valued* network inputs then the lower bound of Theorem 2 can be extended to certain neural nets of depth 2. In addition he has shown that for the case of real valued inputs one can determine exactly the constant factor in these bounds.

Open problems:

1. *Is the VC-dimension of every neural net of depth 2 with boolean inputs, linear threshold gates and w weights bounded by $O(w)$?*

   [The result of [WD] shows that the answer to the corresponding question for depth 1 is positive, and Theorem 2 shows that the answer is negative for any depth $d \ge 3$.]

2. *Consider any neural net $\mathcal{N}$ with linear threshold gates*

$$\langle y_1, \ldots, y_m \rangle \mapsto \mathrm{sgn}\ \left(\sum_{i=1}^{m} \alpha_i y_i + \alpha_0 \right) .$$

Can the VC-dimension of $\mathcal{N}$ become larger if we replace at the hidden nodes of $\mathcal{N}$ the "heaviside activation function" sgn *by some common smooth activation function such as $\sigma(y) = \frac{1}{1+e^{-y}}$, or*

$$\pi(y) = \begin{cases} 0\,, & \text{if } y < 0 \\ y\,, & \text{if } 0 \le y \le 1 \\ 1\,, & \text{if } y > 1 \end{cases} \qquad ?$$

[This problem is open both for the case of boolean and for the case of real valued network inputs. It is demonstrated in [MSS] that certain neural nets can compute more boolean functions if one replaces their heaviside activation functions by σ or π.]

3. *Can one close the gaps between the best known upper bounds and the best known lower bounds for the VC-dimension of neural nets with w weights, activation functions σ or π, and boolean network output?*

[For σ the best known upper bound is "$< \infty$" ([MS]) and the best known lower bound is $\Omega(w \log w)$ (see Theorem 2). For π the best known upper bound is $O(w^2)$ ([GJ]) and the best known lower bound is $\Omega(w \log w)$ (see Theorem 2)].

2 AGNOSTIC PAC-LEARNING ON ANALOG NEURAL NETS

Our goal is to prove a *positive* learning result for *multi-layer analog* neural nets in a *realistic model for learning.* To achieve this goal, one has to overcome the following **obstacles**:

(a) The usual model of PAC-learning on a neural net is *not* a "*realistic*" learning model. It

- $*$ assumes that there is an "ideal" weight-setting $\underline{\alpha}^*$ s.t. $\mathcal{N}^{\underline{\alpha}^*}$ has true error $= 0$ (i.e. $\mathcal{N}^{\underline{\alpha}^*}(\underline{x}) = \underline{y}$ for *all* examples $\langle \underline{x}, \underline{y} \rangle$).
- $*$ allows *very little noise* in examples
- $*$ applies only to *binary output*

(b) There exist very strong *negative results* for PAC-learning on neural nets.

(c) Training of a multi-layer *analog* neural net requires to solve a *highly nonlinear* optimization problem.

Before we describe possibilities for overcoming these obstacles, we first give a precise definition of the neural network model that we consider.

Definition 3. *A* <u>*network architecture*</u> *(or "neural net") $\mathcal{N}$ is a labeled acyclic directed graph. Its nodes of fan-in 0 ("* <u>*input nodes*</u>*"), as well as its nodes of fan-out 0 ("* <u>*output nodes*</u>*") are labeled by natural numbers.*

A node g in $\mathcal{N}$ with fan-in $r > 0$ is called a <u>*computation node*</u> *(or* <u>*gate*</u>*), and it is labeled by some activation function $\gamma_g : \boldsymbol{R} \to \boldsymbol{R}$ and some polynomial $Q_g(y_1, \ldots, y_r)$.*

One says that $\mathcal{N}$ is of <u>*order*</u> *v if all polynomials Q_g in $\mathcal{N}$ are of degree $\leq v$. The coefficients in the polynomials $\boldsymbol{Q}_g$ for the gates g in $\mathcal{N}$ are called the* <u>*programmable parameters (or "weights") of $\mathcal{N}$*</u>*.*

Assume that $\mathcal{N}$ has w programmable parameters, that some numbering of these has been fixed, and that values for all non-programmable parameters have been assigned. Furthermore assume that $\mathcal{N}$ has d input nodes and l output nodes. Then each assignment $\underline{\alpha} \in \boldsymbol{R}^w$ of reals to the programmable parameters in $\mathcal{N}$ defines an analog circuit $\mathcal{N}^{\underline{\alpha}}$, which computes a function $\underline{x} \mapsto \mathcal{N}^{\underline{\alpha}}(\underline{x})$ from $\boldsymbol{R}^d$ into $\boldsymbol{R}^l$ in the following way: Assume that some input $\underline{x} \in \boldsymbol{R}^d$ has been assigned to the input nodes of $\mathcal{N}$. If a gate g in $\mathcal{N}$ has r immediate predecessors which output $y_1, \ldots, y_r \in \boldsymbol{R}$, then g outputs $\gamma_g(Q_g(y_1, \ldots, y_r))$.

Remark:

a) Apart from the programmable parameters, a network architecture $\mathcal{N}$ also contains some fixed parameters (which one should view as "hardwired"). These are the parameters that occur in the definitions of the activation functions γ_g (e.g. thresholds between linear pieces for a piecewise linear activation function).

b) The preceding definition of a network architecture is quite general. It contains various special cases which are of particular importance.

A gate g with the "*heaviside activation function*" $x \mapsto \operatorname{sgn}(x)$ (where $\operatorname{sgn}(x) := 1$ if $x \geq 0$, else $\operatorname{sgn}(x) = 0$) and a polynomial Q_g of degree ≤ 1 (i.e. Q_g is simply a "weighted sum") is called a *linear threshold gate*. If all gates in $\mathcal{N}$ are linear threshold gates one refers to $\mathcal{N}$ as a *threshold circuit* (with variable weights).

If $\mathcal{N}$ employs some activation functions with non-boolean output, one refers to $\mathcal{N}$ as an *analog neural net*. Most experiments with learning on neural nets are carried out on analog neural nets with smooth activation functions such as the *sigmoid function* $\sigma(y) = \frac{1}{1+e^{-y}}$ or some piecewise polynomial approximation to this function. The reason is that most heuristic learning algorithms for multi-layer neural nets require that the network output is differentiable as a function of the programmable parameters of the neural net. Another important advantage of analog neural nets is that they can be used to "learn" *real-valued* functions.

In addition it has recently been shown that certain *boolean* functions can be computed more efficiently on an analog neural net (see

[MSS]).

Often one only considers neural nets of order 1, where all polynomials Q_g are simply weighted sums. But neural nets of high order allow us to implement radial basis functions, and they also provide a somewhat better model of real neurons.

We now describe a possible way to overcome the previously mentioned three obstacles.

Ad obstacle (a): ["find realistic learning model"]
One can use Haussler's model for "agnostic PAC-learning of functions" ([H]). This model

* requires no assumption that an "ideal" weight setting exists with true error $= 0$
* allows arbitrary noise in examples
* allows real-valued "outputs"
* replaces the unrealistic goal [true error of $\mathcal{N}^{\underline{\alpha}}$] $\leq \varepsilon$ by

$$[\text{true error of } \mathcal{N}^{\underline{\alpha}}] \leq \varepsilon + \inf_{G \in \mathcal{T}} [\text{ true error of } G]$$

for a suitable "benchmark class" $\mathcal{T}$.

In the definition of Haussler's model for agnostic PAC-learning one considers for a fixed domain X and a fixed range Y a class $\mathcal{A}$ of distributions on $X \times Y$ (not on X !). Compared with the regular PAC-model this class $\mathcal{A}$ simultaneously plays the role of the class of distributions D on the domain, *and* of the class $\mathcal{C}$ of target concepts. The only class that plays the same role as in the standard definition of PAC-learning is the class $\mathcal{H} \subseteq Y^X$ of hypotheses. This class is determined by the learning approach of the learner, e.g. by a specific neural network architecture.

Obviously in this generalized framework the way in which the quality of a hypothesis $H \in \mathcal{H}$ is evaluated has to change, since we no longer assume that there exists a target concept (or target function) in $\mathcal{H}$ which is consistent with all or at least most examples in a given random sample. Therefore one now compares the performance of each $H \in \mathcal{H}$ with that of the best $H' \in \mathcal{H}$, or (with an eye towards feasibility) with that of the best $G \in \mathcal{T}$ from some specified "touchstone class" $\mathcal{T} \subseteq \mathcal{H}$ (see [KSS]).

This framework is adequate for real world learning situations, where some dataset $S = (\langle x_i, y_i \rangle)_{i \leq m}$ with finitely many points from $X \times Y$ is given to the learner, without any guarantee that any hypothesis $H \in \mathcal{H}$ performs well on S. For example for an application in medicine (where one would like to support the physician by an automated diagnosis system) the sample S may consist of records from a large number of previous patients. In the agnostic PAC-learning model one assumes that S results from independent drawings of elements from $X \times Y$ with regard to some unknown distribution $A \in \mathcal{A}$.

Each possible hypothesis $H \in \mathcal{H}$ is a function from X to Y. Its performance on a sample S ("the *apparent error* on S") is measured by the term $\frac{1}{m}\sum_{i=1}^{m} \ell\Big(H(x_i), y_i\Big)$, for some suitable loss function $\ell : Y \times Y \to \mathbf{R}^+$. For example in the case $Y = \mathbf{R}$ one might choose $\ell(z, y) := |z - y|$, in which case $\frac{1}{m}\sum_{i=1}^{m} \ell\Big(H(x_i), y_i\Big)$ measures the average vertical distance of the datapoints $\langle x_i, y_i\rangle \in S$ from the graph of the function H.

The real goal of the learner is to minimize for the underlying distribution $A \in \mathcal{A}$ the socalled "*true error*" of his hypothesis, i.e. the term $E_{\langle x,y\rangle \in A}[\ell(H(x), y)]$. This term measures the average prediction loss of hypothesis H on new datapoints $\langle x, y\rangle$ that are generated by the same (unknown) distribution A on $X \times Y$.

For many learning problems it is impossible to bring the true error of the best hypothesis $H \in \mathcal{H}$ close to 0 (for example it may be the case for some distribution A that *every* function $F : X \to Y$ has a true error $E_{\langle x,y\rangle \in A}[\ell(F(x), y)] \geq \frac{1}{4}$ because A is very "noisy"). Hence the best one can hope for is to find a hypothesis $H \in \mathcal{H}$, whose true error is close to that of the *best* hypothesis $H' \in \mathcal{H}$, i.e. close to

$$\inf_{H' \in \mathcal{H}} \; E_{\langle x,y\rangle \in A}[\ell(H'(x), y)].$$

However even this goal is often unattainable because of computational difficulties. Therefore it makes sense to consider in addition a "touchstone class" $\mathcal{T} \subseteq \mathcal{H}$, which provides a more modest benchmark for evaluating the quality of hypotheses $H \in \mathcal{H}$. The goal of the learner is then to find a hypothesis $H \in \mathcal{H}$ whose true error is close to

$$\inf_{G \in \mathcal{T}} \; E_{\langle x,y\rangle \in A}[\ell(G(x), y)].$$

Ad obstacle (b): ["How to overcome negative results for PAC-learning"] One characteristic feature of all the well-known *negative* results about PAC-learning on neural nets (see [BR], [J], [KV], [AB], [HSV]) is that one transfers to neural nets a type of asymptotic analysis that has become customary in the analysis of algorithms for digital computation. One assumes in these negative results that the number w of programmable parameters in $\mathcal{N}$ goes to infinity. However this analysis is not quite adequate for many applications of neural nets, where one considers a relatively small *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). Hence it is also of interest to find out whether efficient PAC-learning is possible for a neural net with e.g. $w := 20$ programmable parameters.

For a network architecture $\mathcal{N}$ with *boolean* inputs and outputs and $w = O(1)$ programmable parameters the question whether $\mathcal{N}$ can PAC-learn has a trivial positive answer since $\mathcal{N}$ can only compute $O(1)$ different functions. However for the case of rational or real inputs (and/or outputs) such $\mathcal{N}$ can compute infinitely many different function, and the nontrivial asymptotic question arises (in the regular and in the agnostic PAC-learning model) whether there exists an efficient learning algorithm for $\mathcal{N}$ that allows it to learn any target function with arbitrarily small true error if sufficiently many training examples are provided. Obviously the preceding negative results leave open the question whether there exists for a *fixed* neural net $\tilde{\mathcal{N}}$ a PAC-learning algorithm whose computation time can be bounded by a polynomial in $\frac{1}{\varepsilon}, \frac{1}{\delta}$, in the maximal bit-length n of its d input numbers from $\mathbf{Q}$, and in the bound s for the allowed bit-length of weights in $\tilde{\mathcal{N}}$ (but where the size of $\tilde{\mathcal{N}}$ may occur in the exponent of the time bound). A positive result in this direction is given by the following Theorem. We use there as "touchstone class" the class of all functions computable on a given neural net $\mathcal{N}$ with rational weights of bit-length s.

Ad obstacle (c): ["How to solve efficiently the resulting nonlinear optimization problem"]
It turns out that even in order to optimize the weights for a *fixed* multi-layer neural net $\tilde{\mathcal{N}}$ one has to solve a *nonlinear* optimization problem. This problem arises through the composition of the functions that are computed by units on successive layers of $\tilde{\mathcal{N}}$, and it occurs even if for example each unit computes a linear or piecewise linear function. A solution of this problem can be achieved by focussing on those network architectures $\tilde{\mathcal{N}}$ for which one can transform the nonlinear optimization problem for the weights of $\tilde{\mathcal{N}}$ to a *linear optimization* problem for a *transformed set of parameters*. This method is outlined in step (3) of the subsequent sketch of the proof of Theorem 4.

Let $\mathbf{Q}_n$ be the set of rational numbers that can be written as quotients of integers with bit-length $\leq n$. We write $||z||_1$ for the L_1-norm of a vector $z \in \mathbf{R}^l$.

Theorem 4. *([M 93b])*
Let $B \subseteq \boldsymbol{R}$ be an arbitrary bounded set. Let $\mathcal{N}$ be some arbitrary high order network architecture with d inputs and l outputs. We assume that all activation functions of gates in $\mathcal{N}$ are piecewise polynomial with architectural parameters from $\boldsymbol{Q}$.

Then one can construct an associated first order network architecture $\tilde{\mathcal{N}}$ with linear threshold gates and gates with activation functions from the class $\{x \mapsto x, x \mapsto x^2\}$, as well as a polynomial $m(\frac{1}{\varepsilon}, \frac{1}{\delta})$ and a learning algorithm $LEARN_{\tilde{\mathcal{N}}}$ such that for any given $s, n \in \boldsymbol{N}$ and any distribution A over $\boldsymbol{Q}_n^d \times (\boldsymbol{Q}_n \cap B)^l$ the following holds:
For any sample $S = (\langle \underline{x}_i, \underline{y}_i \rangle)_{i=1,\ldots,m}$ of $m \geq m(\frac{1}{\varepsilon}, \frac{1}{\delta})$ examples that are independently drawn according to A the algorithm $LEARN_{\tilde{\mathcal{N}}}$ computes from S, 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

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

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

The **proof** of Theorem 4 consists of three steps:

(1) Construction of the auxiliary neural net $\tilde{\mathcal{N}}$.

(2) Reducing the optimization of weights in $\tilde{\mathcal{N}}$ for a given distribution A to a *finite* nonlinear optimization problem.

(3) Reducing the resulting finite nonlinear optimization problem to a family of finite *linear* optimization problems.

Details to step (1): We use the same construction as in [M 93a]. If the activation functions γ_g in $\mathcal{N}$ are piecewise linear and all computation nodes in $\mathcal{N}$ have fan-out ≤ 1 (this occurs for example if $\mathcal{N}$ has just one hidden layer and only one output) then one can set $\tilde{\mathcal{N}} := \mathcal{N}$. If the γ_g are piecewise linear but not all computation nodes in $\mathcal{N}$ have fan-out ≤ 1 one defines $\tilde{\mathcal{N}}$ as the tree of the same depth as $\mathcal{N}$, where subcircuits of computation nodes with fan-out $m > 1$ are duplicated m times. The activation functions remain unchanged.

If the activation functions γ_g are piecewise polynomial but not piecewise linear, one has to apply a rather complex construction which is described in detail in the journal version of [M 93b]. In any case $\tilde{\mathcal{N}}$ has the property that all functions that are computable on $\mathcal{N}$ can also be computed on $\tilde{\mathcal{N}}$, the depth of $\tilde{\mathcal{N}}$ is bounded by a constant, and the size of $\tilde{\mathcal{N}}$ is bounded by a polynomial in the size of $\mathcal{N}$ (provided that the depth and order of $\mathcal{N}$, as well as the number and degrees of the polynomial pieces of the γ_g are bounded by a constant).

Details to step (2): With the help of the notion of a *pseudo-dimension* of a neural net one can reduce the desired optimization of weights in $\tilde{\mathcal{N}}$ (with regard to an arbitrary given distribution A of examples $\langle \underline{x}, \underline{y} \rangle$) to a *finite* optimization problem.

Definition 5. *The pseudo-dimension* $dim_P^\ell(\tilde{\mathcal{N}})$ *of* $\tilde{\mathcal{N}}$ *with respect to the loss function* ℓ *is defined as the maximal size of a set* $T \subseteq X \times Y$ *which is shattered by* $\tilde{\mathcal{N}}$ *in the sense that*

$$\exists\, t : T \to \boldsymbol{R}\, \forall\, g : T \to \{0,1\}\, \exists\, \underline{\alpha} \in W^w\ \forall\, \langle x, y \rangle \in T$$
$$(\ell(\tilde{\mathcal{N}}^{\underline{\alpha}}(x), y) \geq t(\langle x, y \rangle) \ \Leftrightarrow\ g(\langle x, y \rangle) = 1).$$

Note that in the special case if $Y = \{0,1\}$, if the network $\tilde{\mathcal{N}}$ outputs only boolean values, and if ℓ is the discrete loss function ℓ_D, then the pseudo-dimension $\dim_P^\ell(\tilde{\mathcal{N}})$ coincides with the VC-dimension of $\tilde{\mathcal{N}}$.

One can show that the pseudo-dimension of the here considered neural net $\tilde{\mathcal{N}}$ is bounded by using an argument from [GJ] (see [M 93b]).

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

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

$$m\left(\frac{1}{\varepsilon}, \frac{1}{\delta}\right) := \frac{257 \cdot r^2}{\varepsilon^2}\left(2 \cdot \dim_P^l(\tilde{\mathcal{N}}) \cdot ln\frac{33er}{\varepsilon} + ln\frac{8}{\delta}\right).$$

By the uniform convergence theorem of Haussler[H] one has for $m \geq m(\frac{1}{\varepsilon}, \frac{1}{\delta})$, $K := \frac{\sqrt{257}}{8}$, and any distribution A over $\mathbf{Q}_n^k \times (\mathbf{Q}_n \cap [b_1, b_2])^l$

$$(1) \quad Pr_{S \in A^m}[\{\forall f \in \mathcal{F} : |(\tfrac{1}{m} \sum_{\langle \underline{x}, \underline{y} \rangle \in S} f(\underline{x}, \underline{y})) - E_{\langle \underline{x}, \underline{y} \rangle \in A}[f(\underline{x}, \underline{y})]| \leq \tfrac{\varepsilon}{K}\}] \geq 1 - \delta,$$

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 $\text{LEARN}_{\tilde{\mathcal{N}}}$ that computes for any $m \in \mathbf{N}$, any sample

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

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

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

This suffices for the proof of Theorem 4, 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 $m \geq m(\frac{1}{\varepsilon}, \frac{1}{\delta})$, with probability $\geq 1-\delta$ (with respect to the random drawing of $S \in A^m$) the algorithm LEARN$_{\tilde{\mathcal{N}}}$ outputs for inputs S and s an assignment $\underline{\tilde{\alpha}}$ of rational numbers to the parameters in $\tilde{\mathcal{N}}$ such that

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

Details to step (3): The computation of weights $\underline{\tilde{\alpha}}$ that satisfy (2) is nontrivial, since this amounts to solving a *nonlinear* optimization problem. This holds even if each activation function in $\tilde{\mathcal{N}}$ is piecewise *linear*, because even then the weights from successive layers are multiplied with each other.

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. 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. Furthermore one can easily compute from $\underline{\alpha}$ values for $\underline{\beta}$ and $\underline{c}$ so that $\forall \underline{x} \in \mathbf{R}^d \left(\tilde{\mathcal{N}}^{\underline{\alpha}}(\underline{x}) = \hat{\mathcal{N}}[\underline{c}]^{\underline{\beta}}(\underline{x})\right)$.

We outline this parameter transformation from $\underline{\alpha}$ to $\underline{c}, \underline{\beta}$ in the simpler case where all activation functions in $\mathcal{N}$ (hence also in $\tilde{\mathcal{N}}$) are piecewise linear. Consider the gate function γ of an arbitrary gate g in $\tilde{\mathcal{N}}$. Since γ is piecewise linear, there are fixed parameters $t_1 < \cdots < t_k, a_0, \ldots, a_k, b_0, \ldots,$

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

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

At the end of this proof we will also need the fact that the previously described parameter transformation can be inverted, i.e. one can compute from $\underline{c}, \underline{\beta}$ an equivalent weight assignment $\underline{\alpha}$ for $\tilde{\mathcal{N}}$ (with the *original* activation functions γ).

We now describe how the algorithm $\text{LEARN}_{\tilde{\mathcal{N}}}$ computes for any given sample $S = (\langle \underline{x_i}, \underline{y_i} \rangle)_{i=1,\ldots,m} \in (\mathbf{Q}_n^k \times (\mathbf{Q}_n \cap [b_1, b_2])^l)^m$ 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 $S = (\langle \underline{x_i}, \underline{y_i} \rangle)_{i=1,\ldots,m}$ 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. Each system L_j reflects one possibility for employing specific linear pieces of the activation functions in $\hat{\mathcal{N}}$ for specific network inputs $\underline{x_1}, \ldots, \underline{x_m}$, and for employing different combinations of weights from $\{-1, 0, 1\}$ for edges between computation nodes.

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 L_j contains only $O(1)$ variables.

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 S 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. We expand the system L_j of linear inequalities to a linear programming problem LP_j in canonical form by adding the optimization requirement

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

The algorithm LEARN$_{\tilde{\mathcal{N}}}$ 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$_{\tilde{\mathcal{N}}}$ computes $\dfrac{1}{m}\sum\limits_{i=1}^{m} ||h_j(\underline{x_i}) - \underline{y_i}||_1$ 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$_{\tilde{\mathcal{N}}}$ employs the previously mentioned 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$_{\tilde{\mathcal{N}}}$.

We refer to the journal version of [M 93b] for the verification that this weight assignment $\underline{\tilde{\alpha}}$ has the desired properties, and for the construction in the more general case where the activation functions of $\mathcal{N}$ are piecewise *polynomial.* ∎

Remark: The algorithm LEARN$_{\tilde{\mathcal{N}}}$ 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 "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 $\underline{\tilde{\alpha}}$ for $\tilde{\mathcal{N}}$ that correspond to the best ones of these solutions h_j. If one views the number $\tilde{w}$ of weights in $\tilde{\mathcal{N}}$ no longer as a constant, one sees that the number of processores that are needed is simply exponential in $\tilde{w}^2$, but that the *parallel computation time* is polynomial in m *and* $\tilde{w}$.

Open problems:

4. *Does Theorem 4 also hold for* $\tilde{\mathcal{N}} := \mathcal{N}$*?*

5. *Can one improve the time bound of the learning algorithm in Theorem 4 to* $O(2^{O(\tilde{w})} \cdot \text{poly}(\frac{1}{\varepsilon}, \frac{1}{\delta}))$*(where* $\tilde{w}$ *is the number of programmable parameters in* $\tilde{\mathcal{N}}$*)?*

[The time bound of $\text{LEARN}_{\tilde{\mathcal{N}}}$ contains a factor of the form $2^{O(\tilde{w}^2)}$, see the journal version of [M 93b].]

6. *Can one extend Theorem 4 to network architectures* $\mathcal{N}$ *with the sigmoid activation function?*

Bibliography

[AB] M. Anthony, N. Biggs, "Computational Learning Theory", *Cambridge University Press*, 1992

[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 64] T. M. Cover, "Geometrical and statistical properties of linear threshold devices", Stanford PH. D. Thesis 1964, *Stanford SEL Technical Report No. 6107-1*, May 1964

[C 68] T. M. Cover, "Capacity problems for linear machines", in: *Pattern Recognition*, L. Kanal ed., *Thompson Book Co.*, 1968, 283 - 289

[GJ] P. Goldberg, M. Jerrum, "Bounding the Vapnik-Chervonenkis dimension of concept classes parameterized by real numbers", *Proc. of the 6th Annual ACM Conference on Computational Learning Theory*, 1993, 361 - 369

[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

[HSV] K. U. Hoeffgen, H. U. Simon, K. S. Van Horn, "Robust trainability of single neurons", to appear

[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

[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

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

[M 93a] W. Maass, "Bounds for the computational power and learning complexity of analog neural nets (Extended Abstract)", *Proc. of the 25th Annual ACM Symposium on the Theory of Computing*, 1993, 335 - 344

[M 93b] W. Maass, "Agnostic PAC-Learning of Functions on Analog Neural Nets", an extended abstract appears in the *Proc. of the 7th Annual IEEE Conference on Neural Information Processing Systems 1993*; the full paper appears in *Neural Computation.*

[M 93c] W. Maass, "Neural Nets with Superlinear VC-Dimension", to appear in *Neural Computation.*

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

[MS] M. MacIntyre, E. D. Sontag, "Finiteness results for sigmoidal neural networks", *Proc. of the 25th Annual ACM Symposium on the Theory of Computing*, 1993, 325 - 334

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

[ORS] A. Orlitzky, V. Roychowdhury, K. S. Siu, "Advances in Neural Computation", *Kluwer Academic Publishers*, to appear in 1994

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

[S] A. Sakurai, "Tighter bounds of the VC-dimension of three layer networks", *Proc. of WCNN '93*, vol. 3, 540 - 543

[WD] R. S. Wenocur, R. M. Dudley, "Some special Vapnik-Chervonenkis classes", *Discrete Math.*, vol. 33, 1981, 313 - 318