to appear in:
Theoretical Advances in Neural Computation and Learning, V. P. Roychowdhury, K. Y. Siu, A. Orlitsky, editors, Kluwer Academic Publishers

Perspectives of Current Research about 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

1 Preliminaries

This paper discusses within the framework of computational learning theory the current state of knowledge and some open problems in three areas of research about learning on feedforward neural nets:

- Neural nets that learn from mistakes
- Bounds for the Vapnik-Chervonenkis dimension of neural nets
- Agnostic PAC-learning of functions on neural nets.

All relevant definitions are given in this paper, and no previous knowledge about computational learning theory or neural nets is required. We refer to [RSO] for further introductory material and survey papers about the complexity of learning on neural nets.

Throughout this paper we consider the following rather general notion of a (feedforward) neural net.

Definition 1.1 *A network architecture (or "neural net") $\mathcal{N}$ is a labeled acyclic directed graph. Its nodes of fan-in 0 (" input nodes"), as well as its nodes of fan-out 0 (" output nodes") are labeled by natural numbers.*

A node g in $\mathcal{N}$ with fan-in $r > 0$ is called a computation node (or gate), and it is labeled by some activation function $\gamma_g : \boldsymbol{R} \rightarrow \boldsymbol{R}$, some polynomial $Q_g(y_1, \ldots, y_r)$, and a subset P_g of the coefficients of this polynomial (if P_g is not separately specified we assume that P_g consists of all coefficients of Q_g).

One says that $\mathcal{N}$ is of order v if all polynomials Q_g in $\mathcal{N}$ are of degree $\leq v$. The coefficients in the sets P_g for the gates g in $\mathcal{N}$ are called the programmable parameters of $\mathcal{N}$.

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) The programmable parameters of a network architecture $\mathcal{N}$ contain the "program" of $\mathcal{N}$. "Learning on $\mathcal{N}$" means that one computes concrete values for the programmable parameters of $\mathcal{N}$ from examples for "desirable" input/output-behaviour. One usually refers to the programmable parameters as "*weights*".

b) 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), and those coefficients of the polynomials Q_g which do not belong to the associated sets P_g. These coefficients could for example encode some "background knowledge" of the neural net.

c) The term "*neural net*" is used in the literature both for a *network architecture* $\mathcal{N}$ in the sense of Definition 1.1, and also for the associated *analog circuit* $\mathcal{N}^{\underline{\alpha}}$ with specific values $\underline{\alpha}$ for its programmable parameters. We will use in this paper the term "network architecture" if we want to stress the former meaning. We will use the term "neural net" if it is not so important (or if it is clear from the context) which of the two interpretations is meant.

d) The definition of a network architecture in Definition 1.1 is quite general. It contains various special cases which are of particular importance.

A gate g with the "*heaviside activation function*" $x \mapsto \mathrm{sgn}(x)$ (where $\mathrm{sgn}(x) := 1$ if $x \geq 0$, else $\mathrm{sgn}(x) = 0$) 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 ([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.

e) For the investigation of learning of "concepts" (i.e. sets) one considers in computational learning theory a framework that consists of a domain X; a class $\mathcal{C} \subseteq 2^X$ of possible target concepts (the "concept class"), and a class $\mathcal{H} \subseteq 2^X$ (the "hypothesis class"). The task of the learner is to compute from given positive and/or negative examples for some unknown target concept $C_T \in \mathcal{C}$ the representation of some hypothesis $H \in \mathcal{H}$ that approximates C_T. A pair $\langle x, 1\rangle$ with $x \in C_T$ is called a *positive example* for C_T, and a pair $\langle x, 0\rangle$ with $x \in X - C_T$ is called a *negative example* for C_T.

For the investigation of concept learning on neural nets one considers a network architecture $\mathcal{N}$ with one boolean output, and one usually defines $\mathcal{H}$ as the class of all sets that are computable by $\mathcal{N}^{\underline{\alpha}}$ for any parameter assignment $\underline{\alpha}$ from a specified range. One considers $\underline{\alpha}$ as a *representation* for the hypothesis $H \in \mathcal{H}$ that is computed by $\mathcal{N}^{\underline{\alpha}}$.

2 Neural Nets that Learn from Mistakes

In this section we consider results about learning on neural nets in the most common model for *on-line learning* (essentially due to Angluin [A] and Littlestone [Li]). In spite of its simplicity, this learning model is quite popular both in computational learning theory and in neural network applications. In fact the famous perceptron learning algorithm (which is closely related to "Hebb's rule") and the backpropagation learning algorithm are typically formulated as learning algorithms for this on-line model.

Consider some arbitrary domain X and some arbitrary concept class $\mathcal{C} \subseteq 2^X$. We assume that the "environment" has fixed some arbitrary "target concept" $C_T \in \mathcal{C}$. In addition it produces some arbitrary sequence $\langle x_i, b_i \rangle_{i \in \mathbf{N}}$ of labeled examples for C_T (i.e. the points $x_i \in X$ are chosen arbitrarily and $b_i = C_T(x_i) \in \{0, 1\}$; note that we identify the set $C_T \subseteq X$ in the usual manner with its characteristic function $\chi_{C_T} : X \to \{0, 1\}$).

We assume that at the beginning of a learning process the learner knows X and $\mathcal{C}$, but not C_T. At step s of the learning process the learner gets to see the point x_s, and he is asked to predict the value of b_s. If this prediction is incorrect, one says that the learner has made a mistake at step s. One can identify the learner in this model with an algorithm A which takes as first argument a finite sequence $\langle x_i, b_i \rangle_{i<s}$ of labeled examples (the previously seen examples) and as second argument some unlabeled point $x \in X$ (the "test point" at step s). The set

$$H_s := \Big\{ x \in X : \, A\Big((\langle x_i, b_i \rangle)_{i<s}, x \Big) = 1 \Big\}$$

is called the *hypothesis* of A at step s. This set may be viewed as the learner's current "model" for the unknown target concept C_T.

The largest possible number of mistakes of algorithm A for arbitrary $C_T \in \mathcal{C}$ and arbitrary sequences $(\langle x_i, C_T(x_i) \rangle)_{i \in \mathbf{N}}$ of labeled examples is denoted by $\mathrm{MB}(\mathcal{C}, A)$ ("mistake bound of algorithm A for concept class $\mathcal{C}$").

A *learning algorithm for $\mathcal{C}$ with hypotheses from $\mathcal{H}$* is in the context of this model an arbitrary function A that assigns to any finite sequence $(\langle x_i, C_T(x_i) \rangle)_{i<s}$ of labeled examples for some $C_T \in \mathcal{C}$ a set $\{x \in X : A((\langle x_i, C_T(x_i) \rangle)_{i<s}, x) = 1\}$ that belongs to $\mathcal{H}$.

One defines for arbitrary "hypothesis classes" $\mathcal{H}$ with $\mathcal{C} \subseteq \mathcal{H} \subseteq 2^X$ the *learning complexity of $\mathcal{C}$ with hypotheses from $\mathcal{H}$* by

$$\begin{aligned}\mathrm{MB}(\mathcal{C},\mathcal{H}) \quad := \quad & \min\{\mathrm{MB}(\mathcal{C},A) : A \text{ is a learning algorithm} \\ & \qquad\qquad \text{for } \mathcal{C} \text{ with hypotheses from } \mathcal{H}\}.\end{aligned}$$

One sets $\mathrm{MB}(\mathcal{C}) \quad := \quad \mathrm{MB}(\mathcal{C},\mathcal{C})$.

Remark 2.1

a) For simplicity the preceding definition of the learning complexity of a concept class does not take into account the *computational complexity* of a learning algorithm A. However it turns out that the concrete learning algorithms A which we will discuss for this model are in fact computationally feasible.

b) *Formally* the environment is required to fix some target concept $C_T \in \mathcal{C}$ at the *beginning* of a learning process. However *practically* at any step of a learning process the environment is still free to choose any $C_T \in \mathcal{C}$ that is consistent with all labeled examples that were so far given to the learner.

c) One can easily show [Li] that for arbitrary $\mathcal{C}$ and $\mathcal{H}$ the learning complexity $\mathrm{MB}(\mathcal{C},\mathcal{H})$ does not change if we define $\mathrm{MB}(\mathcal{C},A)$ in a slightly different manner as the largest possible $t \in \mathbf{N}$ such that for some $C_T \in \mathcal{C}$ and some sequence $(\langle x_i, C_T(x_i)\rangle)_{i\in\mathbf{N}}$ of labeled examples for C_T the learning algorithm A makes a mistake at *each* of the steps $1,\ldots,t-1$. This equivalence is not surprising in view of the fact that we consider in this model a worst case environment, which may just as well wait with the definition of x_s until it has seen (or precomputed) the hypothesis $H_s := \{x \in X : A((\langle x_i, b_i\rangle)_{i<s}, x) = 1\}$ of the learner at step s. If H_s disagrees with C_T, the environment may then choose some x_s so that $\langle x_s, C_T(x_s)\rangle$ provides a *counterexample to hypothesis* H_s (i.e. x_s lies in the symmetric difference $H_s \Delta C_T$ of H_s and C_T).

The preceding observation is useful since it implies that for proving bounds about $\mathrm{MB}(\mathcal{C},\mathcal{H})$ it suffices if we consider only "normalized" learning procedures in which the learner makes a mistake at every single step (until he has "found" C_T). Hence we assume in the following proofs that whenever the current hypothesis H_s of the learner disagrees with C_T then he receives a *counterexample* to H_s at step s of the learning process.

One can also show that it suffices to consider in the mistake bounded model only those learning algorithms which only change their hypothesis at those steps where they have made a mistake.

The variation of the mistake bounded model where the learner makes a mistake at every step is obviously equivalent to Angluin's model [A] for *learning from equivalence queries*. In this interpretation of the on-line learning model one says that the learner poses the *equivalence query* "$H_s = C_T$?" at step s of the learning process, to which he then receives a counterexample $x_s \in H_s \Delta C_T$, or the reply "yes" (in case that $H_s = C_T$).

Our first example for a learning algorithm in this "mistake-bounded" learning model is the famous *perceptron learning algorithm* PLA (which is closely related to Hebb's rule). For that algorithm we consider the network architecture $\mathcal{N}_d$ which consists of a single linear threshold gate $T_{\underline{\alpha}}$ with d real-valued inputs and $d+1$ programmable parameters $\underline{\alpha}$ (called the "weights of $T_{\underline{\alpha}}$", one also refers to α_0 as the "bias" of this gate).

$T_{\underline{\alpha}}$ computes the following function from $\mathbf{R}^d$ into $\{0,1\}$:

$$T_{\underline{\alpha}}(x_1,\ldots,x_d) = \begin{cases} 1, \text{ if } \sum\limits_{i=1}^{d} \alpha_i x_i + \alpha_0 \geq 0 \\ 0, \text{ otherwise.} \end{cases}$$

Remark 2.2

A *perceptron* (see [Ro], [MP]) is a network architecture $\mathcal{N}$ that consists of a linear threshold gate $T_{\underline{\alpha}}$ with d inputs, together with some *fixed* circuit $K : \mathbf{R}^n \to \mathbf{R}^d$ which preprocesses the input for this threshold gate. Through this preprocessing a perceptron can compute a larger variety of interesting functions (e.g. parity on n boolean inputs). However for the analysis of *learning* on a perceptron we may ignore the fixed preprocessing, since only the weights $\underline{\alpha}$ of $T_{\underline{\alpha}}$ are *programmable* parameters of $\mathcal{N}$.

For the sake of notational convenience in the following discussion of the perceptron learning algorithm we extend each input vector $\underline{x} = \langle x_1,\ldots,x_d\rangle$ to a vector $\underline{x}^* := \langle 1, x_1,\ldots,x_d\rangle$. In this way we can view the "*bias*" α_0 also as a *weight* of $T_{\underline{\alpha}}$ (for a dummy input with constant value 1). We then have

$$T_{\underline{\alpha}}(\underline{x}) = 1 \Leftrightarrow \underline{\alpha}\cdot\underline{x}^* \geq 0.$$

We use here the usual notation $\underline{\gamma}\cdot\underline{\delta} := \sum\limits_{i=0}^{d} \gamma_i \cdot \delta_i$ for the inner product of two vectors $\underline{\gamma},\underline{\delta} \in \mathbf{R}^{d+1}$. We write $||\gamma||$ for the L_2- norm $\sqrt{\underline{\gamma}\cdot\underline{\gamma}}$ of $\underline{\gamma}$.

For any set $X \subseteq \mathbf{R}^d$ one defines

$$\text{HALFSPACE}_X^d \;:=\; \Big\{F : X \to \{0,1\} : \; \exists\; \underline{\alpha} \in \mathbf{R}^{d+1}\; \forall\; \underline{x} \in X \Big(F(\underline{x}) = T_{\underline{\alpha}}(\underline{x})\Big)\Big\}.$$

Obviously for the network architecture $\mathcal{N}_d$ that consists of a single linear threshold gate with d inputs this concept class HALFSPACE_X^d is the restriction of all concepts computable on $\mathcal{N}_d$ to the domain X.

The perceptron learning algorithm PLA for this concept class HALFSPACE_X^d is defined as follows. We write $\underline{\alpha(s)}$ for the weight-assignment of the threshold gate after it has processed s examples.

We set $\underline{\alpha(0)} := \underline{0}$ (actually one can start just as well from any other initial weight-assignment).

If $T_{\underline{\alpha(s)}}$ makes a correct prediction for the example $\langle \underline{x}, b\rangle$ presented at step $s+1$ (i.e. $T_{\underline{\alpha(s)}}(\underline{x}) = b$), then one sets $\underline{\alpha(s+1)} := \underline{\alpha(s)}$.

Otherwise, if $b = 1$ (i.e. $\langle \underline{x}, b\rangle$ is a *positive counterexample* to $T_{\underline{\alpha(s)}}$) one sets

$$\underline{\alpha(s+1)} := \underline{\alpha(s)} + \underline{x}^*,$$

and if $b = 0$ (i.e. $\langle \underline{x}, b\rangle$ is a *negative counterexample* to $T_{\underline{\alpha(s)}}$) one sets

$$\underline{\alpha(s+1)} := \underline{\alpha(s)} - \underline{x}^*.$$

It appears quite plausible that with this extremely simple "local" learning rule a clever adversary may cause the learner to make infinitely many mistakes. Therefore the following result had surprised many researchers when it was discovered in the early 60's. It gave some theoretical support for the tremendous excitement about adaptive machines in that period.

Theorem 2.3 *(Perceptron Convergence Theorem; see Rosenblatt [Ro])*
The perceptron learning algorithm PLA satisfies MB(HALFSPACE$_X^d$,PLA)$< \infty$ *for any finite set* $X \subseteq \boldsymbol{R}^d$. ∎

The proof of Theorem 2.3 yields for any concrete $C_T \in$HALFSPACE$_X^d$ an explicit mistake bound of

$$\frac{||\underline{\alpha}||^2 \cdot \delta_{\max}}{\delta_{\min}^2},$$

where $\underline{\alpha} \in \mathbf{R}^{d+1}$ is some arbitrary weight-assignment such that $T_{\underline{\alpha}}(\underline{x}) = C_T(\underline{x})$ for all $\underline{x} \in X$,

$$\begin{aligned} \delta_{\min} &:= \min\{|\underline{\alpha} \cdot \underline{x}^*| : \underline{x} \in X\}, \text{ and} \\ \delta_{\max} &:= \max\{||\underline{x}^*||^2 : \underline{x} \in X\}. \end{aligned}$$

Since X is finite we can always choose $\underline{\alpha}$ such that $\delta_{\min} \geq 1$.

Therefore one can give for the case of boolean inputs (i.e. $X = \{0,1\}^d$) an explicit upper bound for MB(HALFSPACE$_X^d$,PLA):

Theorem 2.4 MB(HALFSPACE$_{\{0,1\}^d}^d$,PLA) $\leq (d+1)^2 \cdot 2^{(d+1)\cdot \log(d+1)}$.

The **proof** of Theorem 2.4 follows immediately from the explicit mistake bound for Theorem 2.3 together with the following very useful estimate:

Theorem 2.5 *(Muroga, Toda, and Takasu [MTT], see also [Mu])*
For all $C \in$ HALFSPACE$_{\{0,1\}^d}^d$ *there exists some* $\underline{\alpha} \in \boldsymbol{Z}^{d+1}$ *whose coordinates have absolute value* $\leq 2^{\frac{d+1}{2}\log(d+1)}$ *such that* $\forall\, \underline{x} \in \{0,1\}^d (C(\underline{x}) = T_{\underline{\alpha}}(\underline{x}))$. ∎

The preceding arguments imply that for the subclass of all $C_T \in$ HALFSPACE$^d_{\{0,1\}^d}$ that can be defined by some $T_{\underline{\alpha}}$ with integer weights $\underline{\alpha}$ of *polynomial size* (with $\delta_{\min} \geq 1$) the perceptron learning algorithm makes at most $O(d^{O(1)})$ mistakes. Unfortunately a trivial counting argument shows that most of the $2^{\Omega(d^2)}$ halfspaces $C \in$ HALFSPACE$^d_{\{0,1\}^d}$ have the property that *every* $\underline{\alpha} \in \mathbf{Z}^{d+1}$ with $C = T_{\underline{\alpha}}$ contains some exponentially large components α_i.

A concrete example for such halfspace is provided by the halfspace of all $\underline{xy} \in \{0,1\}^d$ such that the natural number that is encoded (in binary notation) by the first half $\underline{x}$ of the d input bits is larger than the natural number that is encoded by the second half $\underline{y}$ of the d input bits. An easy inductive argument shows that any threshold function with integer weights that defines this halfspace requires weights of absolute value $\geq 2^{d/4}$. Hastad [H] has recently shown that there exist in fact some $C \in$ HALFSPACE$^d_{\{0,1\}^d}$ that require integer weights of size $2^{\Omega(d \log d)}$.

The preceding lower bounds for the size of integer weights for threshold gates are "bad news" for the analysis of the perceptron learning algorithm, since they imply the same lower bound for the number of mistakes of this algorithms (for some $C_T \in$ HALFSPACE$^d_{\{0,1\}^d}$). One just has to observe that for boolean inputs we have $\underline{\alpha} \in [-s, \ldots, s]^{d+1}$ for the hypothesis $T_{\underline{\alpha}}$ of the perceptron learning algorithm after s mistakes.

On the other hand the following result shows that there also exists a learning algorithm for threshold gates that it is *guaranteed* to converge for *every* target concept $C_T \in$ HALFSPACE$^d_{\{0,1\}^d}$ after at most $O(d^{O(1)})$ mistakes.

Theorem 2.6 *([MT 89], [MT 94]).*
$\binom{d}{2} \leq$ MB(HALFSPACE$^d_{\{0,1\}^d}$) $= O(d^2 \log d)$ *for every* $d \geq 2$.

Furthermore $\binom{d}{2} \cdot n \leq$ MB(HALFSPACE$^d_{X^d_n}$) $= O(d^2(\log d + n))$ *for the larger domain* $X^d_n := \{0, \ldots, 2^n - 1\}^d$.

The upper bounds for these mistake bounds can be achieved by learning algorithms whose number of computation steps are bounded by a polynomial in d and n.

Idea of the **proof of Theorem 2.6:** We only consider the special case $n = 1$ (the general case is similar). For the proof of the *upper bound* we assume for simplicity that $\underline{0} \notin C_T$ for the target concept $C_T \in$ HALFSPACE$^d_{\{0,1\}^d}$. This implies that $\alpha_0 < 0$ for any vector $\underline{\alpha} = \langle \alpha_0, \alpha_1, \ldots, \alpha_d \rangle \in \mathbf{R}^{d+1}$ with $T_{\underline{\alpha}} = C_T$. Hence we can assume w.l.o.g. that $\alpha_0 = -1$, and we can restrict our attention to hypotheses $T_{\underline{\alpha}}$ with $\alpha_0 = -1$. We will view in the following $\underline{\alpha}' := \langle \alpha_1, \ldots, \alpha_n \rangle$ as a code for the halfspace that is computed by $T_{\langle -1, \underline{\alpha}' \rangle}$.

We write V_s for the "version space" of the learning algorithm after s steps. In other words: V_s is the set of all codes $\underline{\alpha}' \in \mathbf{R}^d$ such that $T_{\langle -1, \underline{\alpha}' \rangle}$ is consistent with all labeled examples that the learner has received before step s. Thus we have $V_{s+1} \subseteq V_s$ for all s.

We choose for V_0 a ball around $\underline{0}$ in $\mathbf{R}^d$ with radius $2^{O(d \log d)}$. Thus V_0 has volume $2^{O(d^2 \log d)}$. Theorem 2.5 implies that each possible target concept $C_T \in \mathrm{HALFSPACE}^d_{\{0,1\}^d}$ can be encoded by some vector $\underline{\alpha}' \in V_0$. In fact, since Theorem 2.5 provides for each halfspace a representation $\underline{\alpha}$ that consists of *integers* of bit-length $O(d \log d)$, there exists for each $C_T \in \mathrm{HALFSPACE}^d_{\{0,1\}^d}$ in the *continuous* space V_0 a ball $B_T \subseteq V_0$ of volume $2^{-O(d^2 \log d)}$ such that $C_T = T_{\langle -1, \underline{\alpha}' \rangle}$ for *all* $\underline{\alpha}' \in B_T$. By definition one has $B_T \subseteq V_s$ for all steps s of any learning algorithm for halfspaces.

The strategy of our learning algorithm is to reduce the volume of V_s as quickly as possible. Hence we choose for each step s a hypothesis $T_{\langle -1, \underline{\alpha}'_s \rangle}$ such that *any* counterexample x_s to $T_{\langle -1, \underline{\alpha}'_s \rangle}$ eliminates from V_s not only $\underline{\alpha}'_s$, but a significant fraction of all points $\underline{\alpha}' \in V_s$. This is possible since for *any* counterexample $\langle x_s, b_s \rangle$ to $T_{\langle -1, \underline{\alpha}'_s \rangle}$ the set of all $\underline{\alpha}' \in \mathbf{R}^d$ such that $T_{\langle -1, \underline{\alpha}' \rangle}$ is inconsistent with $\langle x_s, b_s \rangle$ forms a *halfspace* in $\mathbf{R}^d$ that contains $\underline{\alpha}'_s$. For example a negative counterexample $x_s = \langle y_1, \ldots, y_d \rangle$ to $T_{\langle -1, \underline{\alpha}'_s \rangle}$ eliminates from V_s all points in the set $\{\langle \alpha_1, \ldots, \alpha_d \rangle \in \mathbf{R}^d : \sum_{i=1}^{d} \alpha_i y_i - 1 \geq 0\}$. This observation suggests to choose as $\underline{\alpha}'_s$ a point which lies in the "center" of V_s in the sense that any halfspace that contains $\underline{\alpha}'_s$ contains a constant fraction of the volume of V_s. This can be achieved by choosing $\underline{\alpha}'_s$ to be the "volumetric center" of V_s in the sense of Vaidya [Va]. This approach provides a constant $c < 1$ such that

$$\text{volume } (V_{s+1}) \leq c \cdot \text{ volume } (V_s)$$

for all steps s of the learning algorithm with $H_s \neq C_T$. Since $B_T \subseteq V_s$ for all steps s, the number of steps (i.e. mistakes) of this learning algorithm can be bounded by

$$\log_{1/c} \left(\frac{\text{volume } (V_0)}{\text{volume } (B_T)} \right) = \log_{1/c} 2^{O(d^2 \log d)} = O(d^2 \log d).$$

Alternatively one can enclose each version space by a small ellipsoid, and define $\underline{\alpha}'_s$ as the center of that ellipsoid. With this approach (which follows the approach of Khachian's ellipsoid algorithm, see [PS]) one can only achieve that

$$\text{volume } (V_{s+1}) \leq e^{-\frac{1}{5d}} \cdot \text{ volume } (V_s).$$

This approach gives rise to a learning algorithm with a somewhat larger mistake-bound $O(d^3 \log d)$.

The almost optimal *lower bound* of Theorem 2.5 follows by constructing a suitable adversary strategy. According to Littlestone [Li] it suffices to consider only relatively simple adversary strategies which can be represented as binary branching decision trees ("mistake trees" [Li], or "adversary trees" [MT 92]). The construction of such decision tree of depth $\geq \binom{d}{2}$ for $\mathrm{HALFSPACE}^d_{\{0,1\}^d}$ is rather easy (see [MT 94]). ∎

Open problems:

1. *Can one close the gap of "$\log d$" between the upper and lower bound for* MB(HALFSPACE$^d_{\{0,1\}^d}$)?

   [It is quite curious that the same "$\log d$-gap" remains open for the mistake bound for learning axis parallel rectangles over the same domain, see [CM], [Au].]

2. *Does there exist a* <u>*noise-robust*</u> *learning algorithm for* HALFSPACE$^d_{\{0,\ldots,2^n-1\}^d}$ *whose mistake bound is polynomial in d and/or n?*

   [Such noise-robust algorithm has been constructed for learning rectangles in [Au].]

3. *Does there exist a learning algorithm for* HALFSPACE$^d_{\{0,1\}^d}$ *whose mistake bound is polynomial in d, and which is* <u>*local*</u> *in a similar way as the perceptron learning algorithm?*

   [In the perceptron learning algorithm all weights are updated independently from each other, requiring no "global control". Learning algorithms of this type are of particular interest from the point of view of neurobiology.

   Some negative results in this direction are given in section 6 of [MT 94].]

The next goal in an investigation of on-line learning on neural nets is to find efficient on-line learning algorithms for network architectures that consist of *several* linear threshold gates. Perhaps the simplest network architecture of this type consists of an AND of two linear threshold gates. We write IH^2_n for the class of all concepts which can be computed by this network architecture over the 2-dimensional discrete grid $X_n := \{0, \ldots, 2^n - 1\}^2$. The concepts from this class have a very simple geometrical interpretation: they are <u>i</u>ntersections of two <u>h</u>alfspaces (hence the abbreviation IH).

Unfortunately the next result shows that there exists no learning algorithm for this concept class with a polynomial mistake bound (not even if we allow algorithms with unlimited *computational* power).

Theorem 2.7 *([MT 93])*
$\mathrm{MB}(\mathrm{IH}^2_n) = \Omega(2^n)$.

Idea of the **proof:** We construct an adversary strategy which exploits the following obvious fact: any $C \in \mathrm{IH}^2_n$ which contains the set CENTER that consists of the 4 points in the center of the domain $\{0, \ldots, 2^n - 1\}^2$ contains also at least one point from the perimeter of $\{0, \ldots, 2^n - 1\}^2$.

Whenever a hypothesis $H_s \in \mathrm{IH}^2_n$ does not contain all 4 points of the set CENTER, the adversary can give one of these 4 points as a *positive* counterexample. Otherwise he can give (by the preceding observation) some point from the perimeter of $\{0, \ldots, 2^n - 1\}^2$ as a *negative* counterexample.

The key point of this strategy is that there exists a subclass of $\Omega(2^n)$ concepts $C \in \mathrm{IH}_n^2$ that all contain CENTER, but which are pairwise disjoint on the perimeter of $\{0, \ldots, 2^n - 1\}^2$ (consider concepts that are defined by 2 parallel hyperplanes with distance $O(1)$). At most s of the concepts from this subclass have been eliminated as candidates for C_T after s steps of the preceding adversary strategy, no matter which hypotheses $H_s \in \mathrm{IH}_n^2$ have been proposed by the learner. ∎

Open problems:

4. *Consider the class* IH^d *of intersections of 2 halfspaces over the d-dimensional* *__boolean__ domain* $\{0,1\}^d$. *Is* $\mathrm{MB}(\mathrm{IH}^d) = O(d^{O(1)})$?

[Part b) of the remark at the beginning of section 3, in combination with the result of [BR] implies that no learning algorithm with a polynomial bound on its computation time can achieve this mistake bound, unless R=NP.]

5. *Can one prove positive results for on-line learning on a multi-layer threshold circuit* $\mathcal{N}$ *by considering interesting classes* $\mathcal{C}$ *of target concepts which are proper subclasses of the hypothesis class* $\mathcal{H}$ *that is defined by* $\mathcal{N}$?

In spite of the preceding negative results for on-line learning on multi-layer network architectures with linear threshold gates, there exists an on-line learning algorithm for *analog* multi-layer network architectures which has been quite successful in various practical applications: the *backpropagation algorithm.* The success of this learning algorithm has contributed significantly to the excitement about adaptive neural nets during the last decade. However within the framework of computational learning theory no results are known about the performance of this algorithm.

3 Bounds for the Vapnik-Chervonenkis Dimension of Neural Nets

We now turn to the analysis of learning on neural nets in Valiant's model [V] for __p__robably __a__pproximately __c__orrect learning ("PAC-learning"), see also [AB].

Let $\mathcal{N}$ be some arbitrary network architecture with w weights from some weight-space W (e.g. $W = \mathbf{N}, \mathbf{Q}$, or $\mathbf{R}$). If $\mathcal{N}$ has d input-nodes, and if the output gate has range $\{0,1\}$, then $\mathcal{N}$ computes for any weight-assignment $\underline{\alpha} \in W^w$ a function $\mathcal{N}^{\underline{\alpha}}$ from some d-dimensional domain X (e.g. $X = \mathbf{N}^d, \mathbf{Q}^d, \mathbf{R}^d$) into $\{0,1\}$.

In the analysis of learning on neural nets in the PAC-learning model one typically assumes that the network architecture $\mathcal{N}$ defines the *hypothesis class* $\mathcal{H}$ over the domain X. In addition one fixes a class $\mathcal{C} \subseteq 2^X$ of possible target concepts (the "concept class"). The learner is given a parameter $\varepsilon > 0$ ("error parameter") and a parameter $\delta > 0$ ("confidence parameter"). The task of the learner is to determine

a sample-bound $m(\varepsilon, \delta)$ so that he can solve the following problem:
For any distribution D over X, any target concept C_T from the class $\mathcal{C} \subseteq 2^X$, and any sample $S = (\langle x_i, C_T(x_i)\rangle)_{i \leq m}$ of $m \geq m(\varepsilon, \delta)$ labeled examples for C_T with points x_i drawn independently according to D, he can compute from S, ε, and δ the representation of some hypothesis $H \in \mathcal{H}$ (in our case a suitable parameter-assignment $\underline{\alpha}$ for $\mathcal{N}$ so that $H = \mathcal{N}^{\underline{\alpha}}$) such that with probability $\geq 1 - \delta$

$$E_{x \in D}[|H(x) - C_T(x)|] \leq \varepsilon$$

(i.e. $D[\{x \in X : \mathcal{N}^{\underline{\alpha}}(x) \neq C_T(x)\}] \leq \varepsilon$).

If $m(\varepsilon, \delta)$ is bounded by a polynomial in $\frac{1}{\varepsilon}$ and $\frac{1}{\delta}$, and if the representation of H can be computed from S, ε, and δ by an algorithm whose computation time is bounded by a polynomial in $\frac{1}{\varepsilon}, \frac{1}{\delta}$, and the length of S, one says that $\mathcal{C}$ *is efficiently PAC-learnable with hypothesis class* $\mathcal{H}$.

Remark

a) In this section we will always assume that $\mathcal{C} \subseteq \mathcal{H}$ (typically $\mathcal{C} = \mathcal{H}$). In section 4 we will look at a somewhat more realistic scenario where $\mathcal{C} \nsubseteq \mathcal{H}$ is also allowed.

b) Angluin [A] has shown that efficient learnability in the mistake bounded model implies efficient PAC-learnability.

c) It is easy to construct examples of concept classes which show that the converse of the preceding observation does not hold, i.e. efficient PAC-learnability does not imply efficient learnability in the mistake bounded model (e.g. consider the class of singletons over a finite set). There exists however also a natural example for this difference between the two learning models in the context of learning on neural nets. We had shown in Theorem 2.7 that $\mathrm{MB}(\mathrm{IH}_n^2) = \Omega(2^n)$ for the class IH_n^2 of intersections of two halfplanes over $\{0, \ldots, 2^n - 1\}^2$. The VC-dimension of IH_n^2 can be bounded with the help of Theorem 3.3, and therefore one can show with the help of Theorem 3.1 that IH_n^2 is efficiently PAC-learnable (consider all polynomially in $|S|$ many possibilities for partitioning a sample S by a concept from IH_n^2).
   In fact one can even show that IH_n^2 is efficiently learnable in the more demanding model for agnostic PAC-learning that we will discuss in section 4.

One says that a subset T of the domain X is *shattered* by a neural net $\mathcal{N}$ if every function $g : T \to \{0, 1\}$ can be computed on $\mathcal{N}$, i.e.

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

The *Vapnik-Chervonenkis dimension* of $\mathcal{N}$ (abbreviated: VC-dimension($\mathcal{N}$)) is defined as the maximal size of a set $T \subseteq X$ that is shattered by $\mathcal{N}$, i.e.

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

It should be pointed out that the VC-dimension of $\mathcal{N}$ depends in general on the considered domain X and on the chosen weight-space W.

Of course one can define without reference to neural nets more generally for *any* class $\mathcal{H}$ of functions $f : X \to \{0,1\}$ (i.e. subsets of X) the VC-dimension of $\mathcal{H}$ by

$$\text{VC-dimension}(\mathcal{H}) := \\ \max\{|T| : T \subseteq X \text{ and } \forall\, g : T \to \{0,1\}\ \exists\, f \in \mathcal{H}\ \forall\, x \in T\ (g(x) = f(x))\}.$$

Thus our preceding definition of the VC-dimension of a neural net $\mathcal{N}$ is just a special case of this general definition for the class $\mathcal{H} := \{f : X \to \{0,1\} : \exists \underline{\alpha} \in W^w \forall x \in X\ (f(x) = \mathcal{N}^{\underline{\alpha}}(x))\}$.

The importance of the VC-dimension of a neural net $\mathcal{N}$ for PAC-learning on $\mathcal{N}$ arises from the following theorem. This result provides significant information about the *generalization abilities* of a neural net.

Theorem 3.1 *([BEHW])*
Assume that $\mathcal{H} \subseteq 2^X$ satisfies VC-dimension $(\mathcal{H}) < \infty$ and $\mathcal{H}$ is well-behaved (the latter is a rather benign measure-theoretic assumption that is always satisfied if $\mathcal{H}$ is countable; see [BEHW] for details).
Then for

$$m(\varepsilon, \delta) := \max\left(\frac{8 \cdot \text{ VC-dimension}(\mathcal{H})}{\varepsilon} \cdot \log\frac{13}{\varepsilon}\ ,\ \frac{4}{\varepsilon} \cdot \log\frac{2}{\delta}\right)$$

any function A that assigns to a randomly drawn sample S of $m \geq m(\varepsilon, \delta)$ examples $\langle x, b\rangle$ for some target concept $C_T \in \mathcal{H}$ (with x drawn according to some arbitrary distribution D over X) some hypothesis $A(S) \in \mathcal{H}$ that is consistent with S is a PAC-learning algorithm, since we have then $E_{x \in D}[|C_T(x) - A(S)(x)|] \leq \varepsilon$ with probability $\geq 1 - \delta$. ∎

There exists an almost matching *lower bound* for $m(\varepsilon, \delta)$, which shows that no PAC-learner can do his job with substantially fewer examples (not even if he has unlimited computational power). It is shown in [EHKV] that

$$m(\varepsilon, \delta) = \Omega\left(\frac{\text{VC-dimension}(\mathcal{H})}{\varepsilon}\ ,\ \frac{1}{\varepsilon} \cdot \ln\frac{1}{\delta}\right)$$

examples are needed for any nontrivial class $\mathcal{H} \subseteq 2^X$, for *any* PAC-learning algorithm for $\mathcal{H}$.

Theorem 3.1 allows us to divide the task of efficient PAC-learning on a given network architecture $\mathcal{N}$ into two separate tasks:

(i) the proof of a polynomial upper bound (in terms of the "size of $\mathcal{N}$") for the VC-dimension of $\mathcal{N}$, and

(ii) the design of a (probabilistic) polynomial time algorithm which computes for any given sample S some weight-assignment $\underline{\alpha}$ for $\mathcal{N}$ such that $\mathcal{N}^{\underline{\alpha}}$ is consistent with all examples from S (provided there exists such $\underline{\alpha}$).

In fact, it turns out (see [HKLW]) that a polynomial upper bound for the VC-dimension of $\mathcal{N}$ and the existence of an algorithm as in (ii) are sufficient *and necessary* conditions for efficient PAC-learning on $\mathcal{N}$.

It has been shown by Blum and Rivest [BR] that task (ii) is not even feasible for the simplest multi-layer neural net $\mathcal{N}$ with 3 computation nodes. On the other hand it turns out that task (i) can be solved for neural nets with arbitrary architectures for various important types of activation functions. Although these polynomial upper bounds for the VC-dimension cannot be used to prove positive PAC-learning results for neural nets, they are still considered to be quite relevant for practical applications of neural nets. Bounds for the VC-dimension of a neural net $\mathcal{N}$ provide a quantitative relationship between the "apparent error" of a trained neural net $\mathcal{N}^{\underline{\alpha}}$ on a randomly drawn training set S (where the weight-assignment $\underline{\alpha}$ may for example arise from some heuristic learning algorithm such as backpropagation), and the "true error" of $\mathcal{N}^{\underline{\alpha}}$ for new examples drawn from the same distribution. Theorem 3.1 covers a special case of this relationship (for the case where $\mathcal{N}^{\underline{\alpha}}$ has apparent error 0), and the general case is covered by Theorem 4.1 in the next section. This Theorem 4.1 is formulated for the more general setting of neural nets with real valued outputs. The pseudo-dimension of a neural net $\mathcal{N}$ (respectively of its associated function class $\mathcal{F}$) which occurs in Theorem 4.1 coincides with the VC-dimension of $\mathcal{N}$ for the special case of neural nets $\mathcal{N}$ with boolean output, $Y = \{0, 1\}$, and the discrete loss function ℓ_D.

We will present in the next theorems the most important known results regarding upper and lower bounds for the VC-dimension of neural nets.

Theorem 3.2 *(Wenocur and Dudley [WD])*
Assume that the network architecture $\mathcal{N}_d$ consists of a single linear threshold gate with d inputs. Then VC-dimension $(\mathcal{N}_d) = d + 1$ for any domain X with $\{0,1\}^d \subseteq X \subseteq \boldsymbol{R}^d$, and for any weightspace W with $\boldsymbol{Z} \subseteq W \subseteq \boldsymbol{R}$.

Proof: In order to show that VC-dimension $(\mathcal{N}_d) \leq d + 1$ one applies Radon's Theorem (see [E], p. 64). Radon's Theorem states that any set T of $\geq d + 2$ points in $\mathbf{R}^d$ can be partitioned into sets S_0 and S_1 such that the convex hull of S_0 and the convex hull of S_1 intersect. Obviously such sets S_0, S_1 cannot be separated by a hyperplane, and therefore there exists no $\underline{\alpha} \in \mathbf{R}^{d+1}$ such that $\forall\, x \in S_0(\mathcal{N}_d^{\underline{\alpha}}(x) = 0)$ and $\forall\, x \in S_1(\mathcal{N}_d^{\underline{\alpha}}(x) = 1)$. Hence no set $T \subseteq \mathbf{R}^d$ of size $> d + 1$ can be shattered by $\mathcal{N}_d$.

On the other hand it is straightforward to verify that the set $T := \{\underline{0}\} \cup \{\underline{e}_i : i \in \{1, \ldots, d\}\}$ can be shattered by $\mathcal{N}_d$: For any given function $g : T \to \{0, 1\}$ the function $\mathcal{N}_d^{\underline{\alpha}}$ with $\underline{\alpha} = \langle \alpha_0, \alpha_1, \ldots, \alpha_d \rangle$ defined by

$$\alpha_0 = \begin{cases} 0 & , \text{ if } g(\underline{0}) = 1 \\ -1 & , \text{ otherwise} \end{cases}$$

and

$$\alpha_i = \begin{cases} 1 & , \text{ if } g(\underline{e}_i) = 1 \\ -1 & , \text{ otherwise} \end{cases}$$

has the property

$$\forall\, x \in T\Big(g(x) = \mathcal{N}_d^{\underline{\alpha}}(x)\Big). \qquad \blacksquare$$

One important general property of classes $\mathcal{C}$ of bounded VC-dimension is exhibited by Sauer's Lemma (see[BEHW]): It states that for any finite domain X and any $k \in \mathbf{N}$ the class $\mathcal{C}$ that consists of all subsets of X of size up to k is the maximal size class $\mathcal{C} \subseteq 2^X$ with VC-dimension$(\mathcal{C}) \leq k$. Hence we have for any class $\mathcal{C} \subseteq 2^X$ with VC-dimension$(\mathcal{C}) \leq k$ that $|\mathcal{C}| \leq \sum_{i=0}^{k} \binom{|X|}{i} \leq |X|^k + 1$. Applied to the neural net $\mathcal{N}_d$ from Theorem 3.2 we get that for any finite set $X \subseteq \mathbf{R}^d$ there exist at most $|X|^{d+1} + 1$ different ways of partitioning X by halfspaces. This observation is crucial for the estimate of the VC-dimension of multi-layer neural nets in the next theorem.

Theorem 3.3 *(Cover [C 64], [C 68]; see also Baum and Haussler [BH])*
Let $\mathcal{N}$ be an arbitrary network architecture with linear threshold gates that has d input nodes, 1 output node, and w programmable parameters.

Then VC-dimension$(\mathcal{N}) = O(w \cdot \log w)$ for any weightspace $W \subseteq \boldsymbol{R}$ and any domain $X \subseteq \boldsymbol{R}^d$.

Proof: Let $T \subseteq \mathbf{R}^d$ be some arbitrary set of size $m \geq 2$ that is shattered by $\mathcal{N}$. By the preceding remark any gate g in $\mathcal{N}$ can compute at most $|X|^{\text{fan-in}(g)+1} + 1$ different functions from any finite set $X \subseteq \mathbf{R}^{\text{fan-in}(g)}$ into $\{0,1\}$ (fan-in(g) denotes the number of inputs of gate g). Hence $\mathcal{N}$ can compute at most $\prod_{g \text{ gate in } \mathcal{N}} (m^{\text{fan-in}(g)+1} + 1) \leq m^{2w}$ different functions from T into $\{0,1\}$. If T is shattered by $\mathcal{N}$ then $\mathcal{N}$ can compute all 2^m functions from T into $\{0,1\}$. In this case the preceding implies that $2^m \leq m^{2w}$, thus $m = O(w \cdot \log w)$. $\blacksquare$

It is hard to imagine that the VC-dimension of a network of linear threshold gates can be *larger* than the sum of the VC-dimensions of the individual linear threshold gates in the network. Hence on the basis of Theorem 3.2 it has frequently been conjectured that the "true" upper bound in Theorem 3.3 should be $O(w)$. The following result disproves this popular conjecture by showing that the superlinear upper bound of Theorem 3.3 and the related upper bound of Baum and Haussler [BH] are asymptotically optimal. This implies that in a larger neural net an average programmable parameter contributes 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 3.4 *([M 93a] and [M 93c])*
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), 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 programmable parameters of $\mathcal{N}_d$.

The **proof** of Theorem 3.4 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 3.4.. 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

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

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 $\exists\, i \in \{0,\ldots,d-1\}((\text{bin}(i))_b = 1 \wedge g(p,q) = \text{bin}(i))$, 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

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

(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 $\geq 2d + \log d$.

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

Subsequently Sakurai [Sa] has shown that if one allows *real valued* network inputs then the lower bound of Theorem 3.4 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.

In applications of neural nets one usually employs nets with continuous activation functions, because only for multi-layer neural nets with *smooth* activation functions one has found learning algorithms (such as backpropagation) that perform well. In order to estimate the number of training examples that are needed in such applications, it has become of interest to determine bounds for the VC-dimension of neural nets with *continuous activation functions.* In order to get a boolean network output from such net we assume that its *output gate* is still a linear threshold gate.

It turns out that the superlinear lower bound from Theorem 3.4 also holds for nets with the common activation function $\sigma(y) = \frac{1}{1+e^{-y}}$. However it is not clear how sharp this lower bound is for nets with smooth activation functions, since it is much harder to prove upper bounds for the VC-dimension of such neural nets. In particular, it turns out that one cannot expect *any* finite upper bound if one just assumes that the analog activation functions in $\mathcal{N}$ are "very smooth squashing functions". Sontag [S] has shown that for the real-analytic function $\Psi(y) := \frac{1}{\pi}\arctan(y) + \frac{\cos y}{7(1+y^2)} + \frac{1}{2}$ a neural net with 2 real valued inputs, 2 hidden units with activation function Ψ and a linear threshold gate as output gate has *infinite* VC-dimension. Note that this function Ψ is strictly increasing and has limits $1, 0$ at $\pm\infty$ (hence it is a "squashing function"). For the case of neural nets with *d boolean* inputs Sontag constructed activation functions with the same analytic properties as the function Ψ, such that the neural net with the same architecture as above has the maximal possible VC-dimension 2^d.

In view of the preceding results it is clear that in order to prove significant upper bounds for the VC-dimension of an analog neural net one has to exploit rather specific properties of its activation functions, such as the structure of their explicit definitions.

The first upper bound for the VC-dimension of a neural net whose gates employ the activation function $\sigma(y) = \frac{1}{1+e^{-y}}$ is due to Macintyre and Sontag. By using a sophisticated result from mathematical logic (order-minimality of the elementary theory L of real numbers with the basic algebraic operations and exponentiation) they have shown:

Theorem 3.5 *(Macintyre and Sontag [MS])*
Let $\mathcal{N}$ be any feedforward neural net with arbitrary activation functions that are definable in the theory L (such as $\sigma(y) = \frac{1}{1+e^{-y}}$), and a linear threshold gate as output gate. Then the VC-dimension of $\mathcal{N}$ (for arbitrary real inputs and arbitrary real weights) is finite. ∎

In addition, for neural nets $\mathcal{N}$ with *discrete* inputs from $\{-K, \ldots, K\}^d$, one layer of hidden units with activation function σ, and a linear threshold gate at the output it has been shown by Bartlett and Williamson that the VC-dimension of $\mathcal{N}$ is bounded by $O(w \log(wK))$, where w is the number of weights in $\mathcal{N}$ (see their related Theorem 4.3 in the next section).

It was shown in [M 93a] that analog neural nets of arbitrary constant depth with d boolean inputs, boolean output, and polynomially in d many gates with piecewise polynomial activation functions and arbitrary real weights, can be simulated by polynomial size neural nets that consist entirely of linear threshold gates. Hence a polynomial upper bound for the VC-dimension of such neural nets follows immediately from Theorem 3.3. Subsequently Goldberg and Jerrum have shown that with the help of Milnor's theorem from algebraic geometry one can prove directly a polynomial upper bound for arbitrary polynomial size neural nets with piecewise polynomial activation functions (in fact their argument also applies to the case of piecewise rational activation functions).

Theorem 3.6 *(Goldberg and Jerrum [GoJ])*
Let $\mathcal{N}$ be any neural net with piecewise polynomial activation functions (with $O(1)$ pieces each), arbitrary real inputs and weights, and boolean output. Then the VC-dimension of $\mathcal{N}$ is at most $O(w^2)$, where w is the total number of weights in $\mathcal{N}$.

We will sketch a **proof** of the corresponding bound for the pseudo-dimension of such neural nets in the next section (Theorem 4.4).

Open problems:

6. *Is the VC-dimension of every network architecture of depth 2 with boolean inputs, linear threshold gates and w programmable parameters bounded by $O(w)$?*
[Theorem 3.2 shows that the answer to the corresponding question for depth 1 is positive, and Theorem 3.4 shows that the answer is negative for any depth $d \geq 3$.]

7. *Consider any network architecture $\mathcal{N}$ with linear threshold gates*

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

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*

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

[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.]

8. *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 "$< \infty$" (see Theorem 3.5) and the best known lower bound is $\Omega(w \log w)$ (see Theorem 3.4). For π the best known upper bound is $O(w^2)$ (see Theorem 3.6) and the best known lower bound is $\Omega(w \log w)$ (see Theorem 3.4)].

9. *Is efficient PAC-learning possible for the hypothesis class $\mathcal{H}$ defined by network architectures of linear threshold gates of depth 2 and some "interesting" class $\mathcal{C} \subsetneq \mathcal{H}$ of target concepts?*

[[KV] have shown that if $\mathcal{C}$ contains all concepts computable by polynomial size threshold circuits of a certain fixed depth larger than 2, then $\mathcal{C}$ is not PAC-learnable with any "reasonable" hypothesis class $\mathcal{H}$. Hence for a positive learning result it is crucial to limit the "computational power" of $\mathcal{C}$.]

4 Agnostic PAC-Learning of Functions on Neural Nets

The previously discussed learning models are not suitable for the analysis of learning on neural nets in the context of real world learning problems, because they are based on an unrealistic assumption. Consider for example the numerous datasets for real world classification problems that are used in machine learning as benchmark problems for empirical comparisons of heuristic learning algorithms (see e.g. [Ho], [WK]). These datasets have in common that one cannot assume that the "examples" are generated by a target concept C_T of a specific structure (e.g. a specific neural net $\mathcal{N}^{\underline{\alpha}}$), as required by the previously discussed learning models. Hence one is forced to drop the assumption that $\mathcal{C} \subseteq \mathcal{H}$.

Apparently the only learning model that is applicable to real world learning problems is Haussler's extension [Ha] of the PAC-learning model, the model for *agnostic PAC-learning* (this notion is due to Kearns, Schapire, and Sellie [KSS]). In this model one makes no a-priori assumption about any "target concept" which generates the examples $\langle x, y\rangle \in X \times Y$. Instead, one allows arbitrary distributions A of examples $\langle x, y\rangle$ from $X \times Y$, for which one does not even require that there exists *any* function $F : X \to Y$ such that $F(x) = y$ for all examples $\langle x, y\rangle$. Thus one allows in particular that the same $x \in X$ may occur in different examples $\langle x, y\rangle$ and $\langle x, y'\rangle$ with $y \neq y'$ (i.e. the examples may be noisy; the probabilistic concepts of [KS] occur as a special case). Another important improvement of Haussler's model is that it does not require that $Y = \{0, 1\}$. Hence we can also analyze in this model the complexity of learning real valued functions on neural nets where not only the outputs of intermediate gates, but also the outputs of the output gates are non-boolean. This is an important step, since there exists some evidence that the specific strength of adaptive neural nets (in contrast to other quite successful learning methods from applied machine learning such as decision tree induction (see [WK]) lies in areas such as process control, where some unknown smooth function has to be approximated by the neural net. It should also be noted that the backpropagation learning algorithm is often applied in this more general framework.

In the following definition of Haussler's model for agnostic PAC-learning we consider 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) 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

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

An example for a positive result of this type is provided by Theorem 4.7, where $\mathcal{T}$ is the class of functions that are computable on a smaller neural net $\mathcal{N}$ (whereas $\mathcal{H}$ is the class of functions computable on a larger neural net $\tilde{\mathcal{N}}$ whose architecture "supports" the learning algorithm that is used).

Definition: *Let X ("domain") and Y ("range") be arbitrary sets, and let $\ell : Y \times Y \to \boldsymbol{R}^+$ be some arbitrary function ("loss function"). Let $\mathcal{A}$ be some arbitrary class of distributions on $X \times Y$, and let $\mathcal{T}$ and $\mathcal{H}$ be classes of functions from X into Y (e.g. classes of functions computable on some fixed network architectures).*

The task of the learner is to determine a sample-bound $m(\varepsilon, \delta)$ so that he can solve the following problem for any given $\varepsilon, \delta > 0$:
For any distribution $A \in \mathcal{A}$ and any given sample $S = (\langle x_i, y_i\rangle)_{i\leq m}$ of $m \geq m(\varepsilon, \delta)$ points from $X \times Y$ that are drawn independently according to A, he is supposed to compute from S, ε, and δ, the representation of some hypothesis $H \in \mathcal{H}$ (e.g. a parameter assignment $\underline{\alpha}$ for some network architecture) so that with probability $\geq 1 - \delta$:

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

If the sample-bound $m(\varepsilon, \delta)$ can be bounded by a polynomial in $\frac{1}{\varepsilon}$ and $\frac{1}{\delta}$, and if the representation of such hypothesis H can be computed from S, ε and δ by an algorithm whose computation time is bounded by a polynomial in $\frac{1}{\varepsilon}, \frac{1}{\delta}$, and the length of S, we say that $\underline{\mathcal{T}\text{ is efficiently PAC-learnable by }\mathcal{H}\text{ assuming }\mathcal{A}}$. In the special case where $\mathcal{T} = \mathcal{H}$ one says that $\underline{\mathcal{H}\text{ is efficiently PAC-learnable assuming }\mathcal{A}}$.

In lack of a better terminology we will refer to this learning model as *agnostic PAC-learning*. Previously [KSS] had used this term only for a special case of this learning model (where $Y = \{0,1\}$, $\mathcal{T} = \mathcal{H}$, ℓ is the discrete loss function, and $\mathcal{A}$ is such that for each sample there exists some concept consistent with the sample). One should note that we are considering here a much more general framework.

Since we assume that the learner has no a-priori knowledge about the actual distribution $A \in \mathcal{A}$, and since he gets to see only a finite sample S from A, he cannot measure the *true error* of any hypothesis $H \in \mathcal{H}$. Instead, he can only measure the *apparent error* of a hypothesis on the random sample S that is drawn according to A. Therefore the learner needs to know how large he should make the sample size $m(\varepsilon, \delta)$ so that with high probability for every possible hypothesis $H \in \mathcal{H}$ the apparent error of H on S is close to the true error of H. It turns out that one can give such bounds for the required sample size in turns of the *pseudo-dimension* of $\mathcal{H}$. This notion provides a useful generalization of the notion of a VC-dimension for classes of hypotheses with non-boolean output.

In order to define the pseudo-dimension of a neural net $\mathcal{N}$ one has to specify a loss function ℓ that is used to measure for any example $\langle x, y\rangle \in X \times Y$ the deviation $\ell(\mathcal{N}^{\underline{\alpha}}(x), y)$ of the prediction $\mathcal{N}^{\underline{\alpha}}(x)$ of the neural net from the target value y. Popular choices for ℓ are $\ell(z,y) = |z-y|$, $\ell(z,y) = (z-y)^2$, or the discrete loss function $\ell_D : X \times Y \to \{0,1\}$ with $\ell_D(z,y) = 0 \Leftrightarrow z = y$.

Definition: *The pseudo-dimension* $dim_P^\ell(\mathcal{N})$ *of* $\mathcal{N}$ *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* $\mathcal{N}$ *in the sense that*

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

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

If the size m of a training set $S = (\langle x_i, y_i\rangle_{i \leq m})$ (which is randomly drawn according to some arbitrary distribution A over $X \times Y$) is relatively large in comparison with the pseudo-dimension of $\mathcal{N}$ then the "apparent error" $\frac{1}{m}\sum_{i=1}^{m} \ell(\mathcal{N}^{\underline{\alpha}}(x_i), y_i)$ of $\mathcal{N}^{\underline{\alpha}}$ is (with high probability) close to the "true error" $E_{\langle x,y\rangle \in A}[\ell(\mathcal{N}^{\underline{\alpha}}(x), y)]$ of $\mathcal{N}^{\underline{\alpha}}$, provided that the range of the values $\ell(\mathcal{N}^{\underline{\alpha}}(x), y)$ is bounded. This relationship is made more precise by the following result.

Theorem 4.1 *(Pollard [P]; Haussler [Ha])*
Assume that the class $\mathcal{F}_{\mathcal{N},\ell}$ of all functions $\langle x,y\rangle \mapsto \ell(\mathcal{N}^{\underline{\alpha}}(x),y)$ with $\underline{\alpha} \in W^w$ is a permissible class of functions from $X \times Y$ into some arbitrary bounded interval $[0,r]$ (the "permissibility" of $\mathcal{F}_{\mathcal{N},\ell}$ is a somewhat technical measurability assumption, which is always satisfied if the weightspace W is countable, e.g. for $W \subseteq \boldsymbol{Q}$).

Then for any distribution A over $X \times Y$ and any sample $S = (\langle x_i, y_i\rangle)_{i\le m}$ of m "training-examples" (which are drawn independently according to distribution A) one has for any given $\varepsilon, \delta > 0$ and any sample-size $m \ge \frac{64r^2}{\varepsilon^2}(2\cdot\ dim_P^\ell(\mathcal{N})\ln\frac{16er}{\varepsilon} + \ln\frac{8}{\delta})$ that with probability $\ge 1-\delta$:

$$\forall\, \underline{\alpha} \in W^w \Big(\Big|\frac{1}{m}\sum_{i=1}^m \ell(\mathcal{N}^{\underline{\alpha}}(x_i),y_i) - E_{\langle x,y\rangle\in A}[\ell(\mathcal{N}^{\underline{\alpha}}(x),y)]\Big| \le \varepsilon\Big).$$

∎

Open problem:

10. *Can one relax the lower bound for m in Theorem 4.1?*

Remark 4.2

a) The preceding result provides a tool for analyzing a rather serious problem in experiments with learning on neural nets $\mathcal{N}$: the effect of "overfitting". Overfitting occurs when the training of the neural net on some finite sample S (e.g. via backprop) yields some hypothesis $H = \mathcal{N}^{\underline{\alpha}}$ whose *apparent error* on the training set S is quite small, but which performs badly on new data (i.e. H has a rather large *true error*). This effect tends to occur if S is relatively small compared with the complexity (i.e. the number w of programmable parameters) of the neural net $\mathcal{N}$. The preceding Theorem 4.1 gives an explicit bound for a sufficient size of the sample (in terms of the pseudo-dimension of $\mathcal{N}$) so that overfitting is unlikely to occur.

b) Theorem 4.1 allows us to reduce agnostic PAC-learning on a neural net $\mathcal{N}$ to the *finite* optimization problem of computing for a sample $S = (\langle x_i, y_i\rangle)_{i\le m}$ of sufficiently large size m a weight-setting $\underline{\alpha}$ for $\mathcal{N}$ that minimizes the apparent error of $\mathcal{N}^{\underline{\alpha}}$ on S.

For example, for $m \ge \frac{64r^2}{(\varepsilon/3)^2}(2\cdot\ \dim_P^\ell(\mathcal{N})\ln\frac{16er}{\varepsilon/3} + \ln\frac{8}{\delta})$, where e is the base of the natural logarithm, it suffices to compute for a random sample $S = (\langle x_i, y_i\rangle)_{i\le m}$ which is drawn according to some arbitrary distribution A over $\mathbf{R}^d \times Y$ some $\underline{\alpha} \in \mathbf{Q}^w$ such that

$$\Big|\frac{1}{m}\sum_{i=1}^m \ell(\mathcal{N}^{\underline{\alpha}}(x_i),y_i) - \inf_{\underline{\alpha}'\in\mathbf{Q}^w} \frac{1}{m}\sum_{i=1}^m \ell(\mathcal{N}^{\underline{\alpha}'}(x_i),y_i)\Big| \le \frac{\varepsilon}{3}.$$

By Theorem 4.1 we then have with probability $\ge 1-\delta$ (with regard to the random drawing of S) that

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

c) Haussler [Ha] has developed in addition a somewhat different method for bounding the sample size for agnostic PAC-learning of functions on neural nets. He has shown that it suffices to bound the "*capacity*" of such neural nets, and he has proven good bounds for the capacity of neural nets with Lipschitz-bounded activation functions (such as σ) if the weights are bounded.

For neural nets $\mathcal{N}$ with the sigmoid activation function $\sigma(y) = \frac{1}{1+e^{-y}}$ the only known upper bounds for the pseudo-dimension are given by a corresponding generalization of Theorem 3.5 (Macintyre and Sontag [MS]), and by the following result:

Theorem 4.3 *(Bartlett and Williamson[BW])*
Let $\mathcal{N}$ be a neural net of depth 2. Assume that the gates on level 1 use the activation function σ (alternatively these gates may compute a radial basis function $\underline{y} = \langle y_1, \ldots, y_m \rangle \mapsto e^{-||\underline{y}-\underline{c}||}$ with "weights" $\underline{c} \in \boldsymbol{R}^m$), and that the output gate on level 2 outputs a weighted sum of its inputs.

Then for discrete inputs from $\{-K, \ldots, K\}^d$ the pseudo-dimension of $\mathcal{N}$ is at most $8w \log_2(11 \cdot wK)$, where w denotes the total number of weights in $\mathcal{N}$.

The **proof** uses an exponential parameter transformation in order to transform the function that is computed by $\mathcal{N}$ into one that is polynomial in its parameters. One can then apply Milnor's Theorem in a similar fashion as in Theorems 3.6 and 4.4. ∎

For neural nets $\mathcal{N}$ with *piecewise polynomial activation functions* one can give the following upper bound.

Theorem 4.4 *Let $\mathcal{N}$ be an arbitrary network architecture of order $O(1)$ with w real valued weights, arbitrary piecewise polynomial (or piecewise rational) activation functions that consist of $O(1)$ pieces of degree $O(1)$, and real-valued network inputs and outputs.*

Assume that the loss function ℓ is defined by $\ell(z, y) = ||z - y||_p$ for some $p \in \{1, 2, \ldots, \}$.

Then $\dim_P^\ell(\mathcal{N}) = O(w^2)$.

Proof: The proof proceeds analogously as that of Theorem 3.6. Set $D := \dim_P^\ell(\mathbf{N})$, let d be the number of input nodes, and let l be the number of output nodes of $\mathcal{N}$. Then there are values $(\langle \underline{x}_i, \underline{y}_i, z_i \rangle)_{i=1,\ldots,D} \in (\mathbf{R}^{d+l+1})^D$ 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 $\leq (2vr)^w$ (v is a bound for the order of $\mathcal{N}$, see Definition 1.1, and r is the maximal degree of a polynomial or rational piece of an activation function in $\mathcal{N}$). The disjunctions arise here from using different combinations of pieces from the activation functions. The conjunctions consist of all associated comparisons with thresholds of the activation functions. The factor 2 in the degree bound arises only in the case of piecewise rational activation functions.

By definition one has that $\Phi_i(\underline{\alpha}_b)$ 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 $\leq K := D \cdot q^w \cdot 2^l \cdot (2w + l + 1)$ atomic formulas that occur in the D formulas $\Phi_1, \ldots, \Phi_D$ has different truth values for $\underline{\alpha}_b, \underline{\alpha}_{\tilde{b}}$ (where q is a bound for the maximal number of polynomial or rational pieces in an activation function of $\mathcal{N}$).

On the other hand, each of the $\leq K$ atomic formulas is a polynomial inequality of degree $\leq (2vr)^w$ (where r is the maximal degree of a polynomial or rational piece of an activation function in $\mathcal{N}$). Hence a theorem of Milnor [Mi] (see also [R]) implies that the number of different combinations of truth assignments to these atomic formulas that can be realized by different $\underline{\alpha} \in \mathbf{R}^w$ is bounded by $(K \cdot (2vr)^w)^{O(w)}$. Thus we have $2^D \leq (K \cdot (2vr)^w)^{O(w)}$, which implies by the definition of K that $D = O(w) \cdot (\log D + w \log q)$. This yields the desired estimate $D = O(w^2)$ for $q = O(1)$. ∎

Open problem:

11. *Can the bound of Theorem 4.4 be improved to $O(w \log w)$?*
[Theorem 3.4 provides a lower bound of $\Omega(w \log w)$ for certain network architectures of this type.]

Agnostic PAC-learning can be reduced to a finite optimization problem according to Remark 4.2 b). However it turns out that the resulting finite optimization problem is computationally quite difficult for most concrete learning problems. The following result shows that any efficient algorithm for agnostic PAC-learning with hypothesis class $\mathcal{H}$ can (for some choices of $\mathcal{A}, \ell$, and $\mathcal{T}$) be used to solve efficiently the associated "minimizing disagreement problem" for $\mathcal{H}$.

Theorem 4.5 *(Kearns, Schapire and Sellie [KSS])*
Let X be some arbitrary set and $Y := \{0, 1\}$. Let $\mathcal{A}$ be the class of all distributions on $X \times Y$, and let ℓ_D be the discrete loss function (i.e. $\ell_D(z, y) = 0$ if $z = y$ and $\ell(z, y) = 1$ if $z \neq y$). Assume that $\mathcal{H}$ is efficiently PAC-learnable assuming $\mathcal{A}$ (with respect to loss function ℓ_D). Then there exists a randomized polynomial time algorithm MD which solves the following minimizing disagreement problem associated with $\mathcal{H}$:

Given any finite sequence $(\langle x_i, y_i \rangle)_{i \leq m}$ of elements of $X \times Y$ (possibly with repetitions and contradictory labels for the same $x \in X$) the algorithm MD with input

$(\langle x_i, y_i\rangle)_{i\le m}$ *computes some* $H \in \mathcal{H}$ *which satisfies with probability* $\geq \frac{3}{4}$*:*

$$\Big|\{i \le m : H(x_i) \ne y_i\}\Big| = \inf_{H' \in \mathcal{H}} \Big|\{i \le m : H'(x_i) \ne y_i\}\Big|.$$

Proof: Let $S \subseteq X \times Y$ be the multi-set which is generated by the sequence $(\langle x_i, y_i\rangle)_{i\le m}$. Let A be the uniform distribution over S, and define $\varepsilon := \frac{1}{m+1}$ and $\delta := \frac{1}{4}$. Let LEARN be any efficient PAC-learning algorithm for $\mathcal{H}$ assuming $\mathcal{A}$ (with loss function ℓ_D).

We construct the randomized algorithm MD as follows. MD draws (internally) a sample S' of $m\left(\frac{1}{m+1}, \frac{1}{4}\right)$ examples from S according to the uniform distribution U_S over S, where $m(\varepsilon, \delta)$ is the polynomially bounded sample-complexity of LEARN. By assumption the algorithm LEARN computes from this sample S' some $H \in \mathcal{H}$ such that with probability $\geq \frac{3}{4}$ (with regard to the drawing of S') the following holds:

$$E_{\langle x,y\rangle \in U_S}\Big[|H(x) - y|\Big] \le \inf_{H' \in \mathcal{H}} E_{\langle x,y\rangle \in U_S}\Big[|H'(x) - y|\Big] + \frac{1}{m+1}.$$

Since every point in S has a weight of at least $\frac{1}{m}$ under the uniform distribution U_S over S, the hypothesis H cannot afford to misclassify any example more than neccessary in order to achieve this performance. Hence H is a solution of the associated minimizing disagreement problem. ■

Obviously in the special case $Y = \{0,1\}$ efficient *agnostic* PAC-learning is at least as difficult as efficient learning in the regular PAC-model. In fact, it is strictly more difficult as the following arguments show. The "minimizing disagreement problem" for a hypothesis space $\mathcal{H}$ contains as a subproblem the task to find some $H \in \mathcal{H}$ that is consistent with the sample S, provided there exists such $H \in \mathcal{H}$. This subproblem captures the computational demands for PAC-learning with hypothesis class $\mathcal{H}$ in the standard PAC-model. It turns out that for various important classes $\mathcal{H}$ this subproblem is computationally feasible (and hence efficient PAC-learning with $\mathcal{H}$ is possible in the standard PAC-model), whereas the minimizing disagreement problem for $\mathcal{H}$ is computationally "hard" (and hence efficient agnostic PAC-learning with hypothesis class $\mathcal{H}$ is impossible according to Theorem 4.5).

The following result shows that the class $\mathcal{H}$ of halfspaces over $\{0,1\}^d$ is an example for a class that is efficiently learnable in the standard PAC-model, but not in the model for agnostic PAC-learning.

Theorem 4.6 *(Hoeffgen, Simon and Van Horn [HSV])*
For all $d \in \boldsymbol{N}$ *let* $\mathcal{A}_d$ *be the class of all distributions on* $\{0,1\}^d \times \{0,1\}$*. Then* $R \ne NP$ *implies that* $\mathcal{H} = (\mathrm{HALFSPACE}^d_{\{0,1\}^d})_{d\in\mathbf{N}}$ *is not efficiently PAC-learnable assuming* $\mathcal{A} = (\mathcal{A}_d)_{d\in\boldsymbol{N}}$*.*

Proof: It is shown in [HSV] that it is NP-hard to decide for an arbitrary multi-set S of positive and negative examples from some domain $\{0,1\}^d$ (i.e. $S \subseteq \{0,1\}^d \times \{0,1\}$) and an arbitrary given $k \in \mathbf{N}$, whether there exists some $H \in \mathrm{HALFSPACE}^d_{\{0,1\}^d}$ that misclassifies at most k of the examples in S.

On the other hand it follows from Theorem 4.5 that an efficient PAC-learning algorithm for $\mathcal{H}$ assuming $\mathcal{A}$ would give rise to a randomized polynomial time algorithm for this NP-complete decision problem. ∎

Since the preceding result shows that agnostic PAC-learning is not even possible for a perceptron, the question arises whether one can prove *anything* positive for agnostic PAC-learning on neural nets. One characteristic feature of all the well-known *negative* results about PAC-learning on neural nets (see [BR], [J], [KV], and Theorem 4.6) 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 4.7. 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.

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 4.7 *([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

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

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

The **proof** of Theorem 4.7 is mathematically quite involved, and we can give here only an outline. It 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 pseudo-dimension and Theorem 4.1 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.

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

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 Theorem 4.1 one has for $m \geq m(\frac{1}{\varepsilon}, \frac{1}{\delta})$, $K := \frac{\sqrt{257}}{8}$, and any distribution A over $\mathbf{Q}_n^k \times (\mathbf{Q}_n \cap [b_1, b_2])^l$

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

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

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 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

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

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 4.8

a) 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 $\sum\limits_{i=1}^{m} ||h_j(\underline{x}_i) - \underline{y}_i||_1$ 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* $\tilde{w}$.

b) The proof of Theorem 4.7 uses an idea that promises to bear further fruits: Rather than insisting on designing an efficient learning algorithm for *every* neural net, one designs 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 fast learning algorithms.

Open problems:

12. *Does Theorem 4.7 also hold for $\tilde{\mathcal{N}} := \mathcal{N}$?*

13. *Can one improve the time bound of the learning algorithm in Theorem 4.7 to $O(2^{O(\tilde{w})} \cdot \text{poly}(\frac{1}{\varepsilon}, \frac{1}{\delta}))$?*

[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].]

14. *Can one extend Theorem 4.7 to network architectures $\mathcal{N}$ with the sigmoid activation function?*

15. *Can one prove further positive results for agnostic PAC-learning by exploiting the observation that various real world classification problems give rise to distributions A over $X \times Y$ that have special structural properties (e.g. for $Y = \{0, 1\}$ one often has that very simple hypotheses can "predict" the labels of examples quite well, see [Ho])?*

5 Conclusion

Several of the existing results about learning on neural nets are negative results. However one should not interpret these results as saying that efficient learning on neural nets is impossible. In fact, efficient learning on neural networks is a reality, both on existing artificial neural nets and even more on neural systems of living organisms. Hence one should view these negative results as useful hints, which guide us towards a better understanding of the essential mechanisms of learning on neural nets, and towards the development of more adequate theoretical models.

Acknowledgements

I would like to thank Ian Parberry for his suggestion to write up these notes, and Giancarlo Mauri for giving me the opportunity to lecture from a draft of these notes at the "1993 Advanced School on Computational Learning and Cryptography" of the European Association for Theoretical Computer Science. I also would like to thank Michael Schmitt for his helpful comments.

References

[A] D. Angluin, "Queries and concept learning", *Machine Learning*, vol. 2, 1988, 319 - 342

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

[Au] P. Auer, "On-line learning of rectangles in noisy environments", *Proc. of the 6th Annual ACM Conference on Computational Learning Theory*, 1993, 253 - 261

[BW] P. L. Bartlett, R. C. Williamson, "The VC-dimension and pseudodimension of two-layer neural networks with discrete inputs", preprint (1993)

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

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

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

[CM] Z. Chen, W. Maass, "On-line learning of rectangles", *Proc. of the 5th Annual ACM Workshop on Computational Learning Theory* 1992, 16 - 28

[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

[E] H. Edelsbrunner, "Algorithms in Combinatorial Geometry", *EATCS Monographs on Theoretical Computer Science*, vol. 10, Springer (Berlin, New York), 1987

[EHKV] A. Ehrenfeucht, D. Haussler, M. Kearns, L. Valiant, "A general lower bound on the number of examples needed for learning", *Information and Computation*, vol. 82, 1989, 247 - 261

[G] S. I. Gallant, "Neural Network Learning and Expert Systems", *MIT Press* (Cambridge, 1993)

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

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

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

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

[Ho] R. C. Holte, "Very simple classification rules perform well on most commonly used datasets", *Machine Learning*, vol. 11, 1993, 63 - 91

[J] J. S. Judd, "Neural Network Design and the Complexity of Learning", *MIT-Press* (Cambridge, 1990)

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

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

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

[Li] N. Littlestone, "Learning quickly when irrelevant attributes abound: a new linear-threshold algorithm", *Machine Learning*, vol. 2, 1988, 285 - 318

[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

[MT 89] W. Maass, G. Turan, " On the complexity of learning from counterexamples" (extended abstract), *Proc. of the 30th Annual IEEE Symp. on Foundations of Computer Science*, 1989, 262 - 267

[MT 92] W. Maass, G. Turan, "Lower bounds and separation results for on-line learning models", *Machine Learning*, vol. 9, 1992, 107 - 145

[MT 93] W. Maass, G. Turan, "Algorithms and lower bounds for on-line learning of geometrical concepts", to appear in *Machine Learning*

[MT 94] W. Maass, G. Turan, "How fast can a threshold gate learn?", in: *Computational Learning Theory and Natural Learning Systems: Constraints and Prospects,* G. Drastal, S. J. Hanson and R. Rivest eds., *MIT Press*, to appear

[MS] 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

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

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

[MTT] S. Muroga, I. Todo, S. Takasu, "Theory of majority decision elements", *J. Franklin Inst.*, vol. 271, 1961, 376 - 418

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

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

[RSO] V. P. Roychowdhury, K. Y. Siu, A. Orlitsky, "Advances in Neural Computation", *Kluwer Academic Publishers*, to appear

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

[P] D. Pollard, "Empirical Processes: Theory and Applications", *NSF-CBMS Regional Conference Series in Probability and Statistics*, vol. 2, 1990

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

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

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

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

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

[Va] P. M. Vaidya, "A new algorithm for minimizing convex functions over convex sets", *Proc. of the 30th Annual IEEE Symp. on Foundations of Computer Science*, 1989, 338 - 343

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

[WK] S. M. Weiss, C. A. Kulikowski, "Computer Systems that Learn", *Morgan Kaufmann* (San Mateo, 1991)

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