appears in: COMPUTATIONAL LEARNING THEORY
AND NATURAL LEARNING SYSTEMS,
Drastal, Hanson, Rivest, eds., MIT-Press

How fast can a threshold gate learn?

WOLFGANG MAASS[1] and GYÖRGY TURÁN[2]

Abstract

This paper addresses a rather old problem in the theory of machine learning: how fast can a threshold gate (or equivalently: a perceptron) learn from its errors? With the well-known perceptron learning algorithm (Hebb's rule) a threshold gate makes at most finitely many errors. However the number of errors is in general exponential in the number d of input variables of the threshold gate. We show in this paper that there is a different on-line learning algorithm for threshold gates, for which the number of errors and the total number of computation steps is polynomial in d. We prove that the error bound of this new learning algorithm is close to the theoretical optimum. We also show that the new learning algorithm can be used for efficient training of threshold gates whose input variables range over a larger domain, and for threshold gates with multiple responses (for example discrete approximations to sigmoid threshold gates).

Finally, we examine in this paper the speed of algorithms for threshold gates that are *distributed* in the sense that they do not require a global control. It is shown that all distributed learning algorithms of a similar type as Hebb's rule and Littlestone's Winnow rules are inherently slow.

---

[1]Institutes for Information Processing Graz, Technische Universitaet Graz, Klosterwiesgasse 32, A-8010 Graz, Austria. e-mail: maass@iicm.tu-graz.ac.at. Written under partial support by NSF grant CCR 890 3398.
[2]Department of Mathematics, Statistics and Computer Science, University of Illinois at Chicago, M/C 249, POB 4348, Chicago, IL 60680. e-mail: U11557@uicvm.bitnet.
Automata Theory Research Group of the Hungarian Academy of Sciences, Aradi tér 1, Szeged 6720, Hungary. Partially supported by OTKA-501.

1. Introduction.

A *threshold gate* G with weights $w_1, \ldots, w_d \in \mathbf{R}$ and threshold $t \in \mathbf{R}$ computes the following function from $\{0,1\}^d$ to $\{0,1\}$: for inputs $x_1, \ldots, x_d \in \{0,1\}$ it outputs 1 if $\sum_{i=1}^{d} w_i x_i \geq t$, else it outputs 0. Any function that can be computed by such a gate G is called a threshold function. For any $\mathbf{w} = (w_1, \ldots, w_d) \in \mathbf{R}^d$ and $t \in \mathbf{R}$ one also refers to

$$H_{\mathbf{w},t} := \left\{ (x_1, \ldots, x_d) \in \{0,1\}^d \mid \sum_{i=1}^{d} w_i x_i \geq t \right\}$$

as a *halfspace over* $\{0,1\}^d$. This is motivated by the fact that $H_{\mathbf{w},t} = F_{\mathbf{w},t} \cap \{0,1\}^d$, where

$$F_{\mathbf{w},t} = \left\{ (x_1, \ldots, x_d) \in \mathbf{R}^d \mid \sum_{i=1}^{d} w_i x_i \geq t \right\}$$

is a halfspace over $\mathbf{R}^d$. Thus the notions of a threshold gate, a threshold function and a halfspace over $\{0,1\}^d$ are equivalent and will be used interchangeably. The class of all halfspaces over $\{0,1\}^d$ is denoted by $\mathrm{HALFSPACE}_2^d$.

A threshold gate may be viewed as a simple mathematical model of the computational abilities of a neuron (McCulloch and Pitts (1943), Rosenblatt (1962), Minsky and Papert (1988), Hopfield (1982)) and it forms the basic building block of neural networks (Rumelhart and McClelland (1986)). Particular attention has been given to the question whether it is necessary to "program" a threshold gate by explicitly providing the parameters $\mathbf{w}$ and t, or whether the threshold gate can automatically *learn* these parameters from its own *errors* in a feasible number of steps.

This question can be made more precise by introducing some standard notions from computational learning theory (Angluin (1988)).

The learning task is to identify an unknown *target* halfspace H_T from $\mathrm{HALFSPACE}_2^d$. A *learning algorithm* (or *learner*) A proposes *hypotheses* H from $\mathrm{HALFSPACE}_2^d$. If the current hypothesis H is incorrect, i.e. $H \neq H_T$, then the learner receives a *counterexample* $\mathbf{x}$ from $(H_T \backslash H) \cup (H \backslash H_T)$. In other words, $\mathbf{x}$ is an input which is processed incorrectly by the threshold gate with the current weights and the current threshold. After such an *error* the values of the parameters are changed according to the learning algorithm A, to obtain a new hypothesis. The new hypothesis may also depend on the previous counterexamples, i.e. on the "history" of the learning process.

The learning complexity $LC(A)$ of the learning algorithm A is the largest number of errors that may occur before A identifies the target halfspace, for any target halfspace and any choice of the counterexamples. $LC(A)$ is also called the *mistake bound* of A (Littlestone (1988)). The *learning complexity* $LC(\mathrm{HALFSPACE}_2^d)$ of $\mathrm{HALFSPACE}_2^d$ is

$$LC(\mathrm{HALFSPACE}_2^d) := \min\{ LC(A) \mid A \text{ is a learning algorithm for } \mathrm{HALFSPACE}_2^d \}.$$

One can now rephrase the question whether a threshold gate can learn within a feasible number of steps as the question whether $LC(\text{HALFSPACE}_2^d)$ is bounded by a polynomial in d.

This learning problem has been studied in particular in the context of *perceptrons* (Rosenblatt (1962), Minsky and Papert (1988), Nilsson (1965)). These are circuits consisting of several gates where only the last gate is a threshold gate with variable weights and threshold.

Rosenblatt has shown that the learning complexity of the so-called perceptron algorithm (or Hebb's rule) for learning a halfspace over $\{0,1\}^d$ is finite (Rosenblatt (1962), Minsky and Papert (1988)). However the learning complexity of this algorithm is clearly not polynomial in d (see Section 6). Littlestone (1988) proposed other learning algorithms for threshold gates (Winnow 1, Winnow 2) but it has remained open whether these can learn all monotone target halfspaces in HALFSPACE_2^d with a polynomial number of mistakes (as the Winnow algorithms only produce nonnegative weights they cannot learn non-monotone target halfspaces).

In this paper we show that $LC(\text{HALFSPACE}_2^d)$ is polynomial in d.

More generally, we consider threshold gates with d inputs from $\{0,\ldots,n-1\}$. This gives a somewhate more realistic model (see the discussion in Hampson and Wolper (1990)) and leads to interesting learning problems even in the case when $d=2$. A threshold gate with d inputs from $\{0,\ldots,n-1\}$ accepts the set

$$H_{\mathbf{w},t} := \left\{(x_1,\ldots,x_d) \in \{0,\ldots,n-1\}^d \mid \sum_{i=1}^{d} w_i x_i \geq t\right\} = F_{\mathbf{w},t} \cap \{0,\ldots,n-1\}^d,$$

called a *halfspace over* $\{0,\ldots,n-1\}^d$. The class of all halfspaces over $\{0,\ldots,n-1\}^d$ is denoted by HALFSPACE_n^d. The learning complexity $LC(\text{HALFSPACE}_n^d)$ is defined analogously to $LC(\text{HALFSPACE}_2^d)$ above. In Section 3 we show that

$$LC(\text{HALFSPACE}_n^d) = O\big(d^2(\log d + \log n)\big),$$

i.e. there is an algorithm for learning a halfspace over $\{0,\ldots,n-1\}^d$ which makes at most $O\big(d^2(\log d + \log n)\big)$ errors. Furthermore, the number of computation steps executed by the algorithm is polynomial in d and $\log n$.

The new learning algorithms presented in this paper are based on the existence of efficient algorithms in *convex optimization* for finding a point in a convex body given by a *separation oracle*. The latter algorithms can be used in particular to find a point in a polytope, thus as a special case one gets those linear programming algorithms which, instead of using an explicit representation of the input in the form of a list of the faces, access their input through a separation oracle. Both the *ellipsoid method* (Khachian (1979),

see Grötschel, Lovász and Schrijver (1988)) and *Vaidya's algorithm* (Vaidya (1989)) fit into this framework. Grötschel, Lovász and Schrijver (1988) give a general theory, presenting several applications of this class of algorithms.

We observe that in fact *every* algorithm for finding a point in a convex body (given by a separation oracle) gives rise to a halfspace learning algorithm. The $O\big(d^2(\log d + \log n)\big)$ learning algorithm is obtained by using the algorithm of Vaidya (1989).

It is substantially easier to design an efficient learning algorithm for halfspaces in Valiant's PAC-model (Valiant (1984)) for batch learning. In this model one assumes that the examples are drawn according to an arbitrary time-invariant distribution over the underlying domain, and it is sufficient to output an approximation to the target concept. It turns out that for a learning algorithm in the PAC-learning model it is sufficient to output any hypothesis that is consistent with the given set of examples (Blumer, Ehrenfeucht, Haussler and Warmuth (1989)). Hence one can design a polynomial time computable learning algorithm for halfspaces in the PAC-learning model in the following way: one uses a computationally feasible algorithm for linear programming for the given set of positive and negative examples in order to compute some halfspace that is consistent with these examples. It is easy to see that this method is too weak to yield a learning algorithm with polynomial error bound in the model considered here for on-line learning. An on-line learning algorithm for halfspaces that always outputs hypotheses that are consistent with all previously seen counterexamples may make up to 2^d errors in the LC-model considered here. Hence a different approach is needed.

We also exhibit in this paper an efficient learning algorithm for threshold gates that have more than two output values. A *multithreshold gate* G with weights $w_1, \ldots, w_d \in \mathbf{R}$ and s different thresholds $t_1 \leq \ldots \leq t_s$ ($t_1, \ldots, t_s \in \mathbf{R}, s \in \mathbf{N}$) is assumed to compute the following function $f_G : \{0, \ldots, n-1\}^d \to \{0, \ldots, s\}$:

$$f_G(x_1, \ldots, x_d) = \begin{cases} \max\left\{ j \mid \sum_{i=1}^{d} w_i x_i \geq t_j \right\} & , \text{ if this set is not empty,} \\ 0 & , \text{ otherwise.} \end{cases}$$

We write $\text{MULTITHRESHOLD}_n^{d,s}$ for the class of all functions computable by such multithreshold gates, for arbitrary weights and thresholds from $\mathbf{R}$. Note that $\text{MULTITHRESHOLD}_n^{d,s}$ contains various discrete approximations to the frequently considered *sigmoid* continuous threshold functions (Rumelhart and McClelland (1986)). Another motivation for the investigation of multithreshold gates is the desire to explore simple models for the (very complicated) information processing capabilities of a neuron in a natural neural system. In a first approximation one may view the current firing rate of a neuron as its current output (see Rumelhart and McClelland (1986), Schwartz (1990)). The firing rates of neurons are known to change between a few and several hundred firings per

second. Hence a multithreshold gate provides a somewhat better computational model for a neuron than a gate that has only two different output signals. Muroga (1971), Olafsson and Abu-Mostafa (1988) have previously investigated non-monotone multilevel threshold functions. Multithreshold automata have been studied by Goles and Martinez (1981).

After defining a suitable extension of the learning model to the case of learning functions, we give in Theorem 4.1 an algorithm for learning multithreshold gates. The number of hypotheses required and the total amount of computation time of this learning algorithm are both bounded by a polynomial of d, s and $\log n$.

We consider in Section 4 also the problem of learning a halfspace over an *arbitrary* finite set $X \subseteq \mathbf{R}^d$, i.e. learning a concept from the class

$$\text{HALFSPACE}_X^d := \{C \subseteq X \mid \text{for some } \mathbf{w} \in \mathbf{R}^d, t \in \mathbf{R} \text{ it holds that } C = X \cap F_{\mathbf{w},t}\},$$

where $F_{\mathbf{w},t}$ is a halfspace in $\mathbf{R}^d$ as defined above. Using a similar approach as in the previous learning algorithms and the existence of a centerpoint for every finite set in $\mathbf{R}^d$ (Yaglom and Boltyanskii (1961), see Edelsbrunner (1987)), we present in Theorem 4.3 a learning algorithm that makes at most polynomial in d and $\log |X|$ many errors. This algorithm does not appear to have a computationally efficient implementation.

We also discuss in this paper various lower bounds for the error bounds that can be achieved by arbitrary on-line learning algorithms for threshold gates. It is shown that the learning complexity of *every* algorithm learning a halfspace over $\{0, \ldots, n-1\}^d$ is $\Omega(d^2 \log n)$ (without any assumption on the computational feasibility of the algorithm). Thus the upper bound mentioned above is optimal up to a factor of $\log d$. The lower bound in fact applies to a larger class of learning algorithms, where a hypothesis may be *any* subset of $\{0, \ldots, n-1\}^d$, and not just a halfspace. Recent results of Littlestone (see Maass (1991)) imply that this lower bound remains valid even if the learner is allowed to use randomization and the environment is assumed to be oblivious in presenting the examples.

One of the simplest threshold circuits consisting of more than one gate is the conjunction of two threshold gates, with two inputs from $\{0, \ldots, n-1\}$. Sets accepted by such circuits correspond to intersections of two halfplanes in $\{0, \ldots, n-1\}^2$. It is shown in Theorem 5.6 that *every* learning algorithm for this class requires $\Omega(n)$ hypotheses in the worst case. This complements the negative results of Blum and Rivest (1988) for the case of threshold gates with n Boolean input variables.

The perceptron learning algorithm (Hebb's rule) differs from our new learning algorithm for threshold gates insofar as it is a *distributed* algorithm: each weight can be thought of being controlled by a separate processor. We show that *every* distributed halfspace learning algorithm which satisfies a condition called *boundedness* (see Section 6) is inherently slow. This condition is satisfied by all known distributed learning algorithms

for threshold gates. In particular, it follows that the learning complexity of the Winnow algorithms of Littlestone (1988) is exponential in d.

This paper is organized as follows. In Section 2 we give a formal definition of the learning model considered. Section 3 contains the new learning algorithms for threshold gates. In Section 4 we discuss learning a threshold gate with several outputs, and learning a halfspace over an arbitrary set. Section 5 contains the lower bound results. Distributed learning algorithms are considered in Section 6. Open problems are discussed in Section 7. The Appendix contains some technical details, and the description of the convex optimization algorithms used in Section 3.

This paper provides complete proofs for a number of results that were announced in Maass and Turán (1989), (1990 a). Complete proofs of other results from these papers are given in Maass and Turán (1990 b), (1990 c).

2. Definitions.

In this section we describe the general on-line learning model of Angluin (1988), which generalizes the classical learning models for perceptrons and neural networks. Littlestone (1988) introduced an equivalent version of this model.

A learning problem is specified by a *domain* X and a *concept class* $\mathcal{C} \subseteq 2^X$. The sets $C \in \mathcal{C}$ are called *concepts*. In this paper X will always be a finite subset of $\mathbf{R}^d$.

The goal of the *learner* (or *learning algorithm*) is to identify an unknown *target concept* $C_T \in \mathcal{C}$, fixed in the beginning of the learning process by the *environment*. The learner proposes *hypotheses* $H \in \mathcal{C}$. If $H = C_T$ then the environment responds "yes". Otherwise, it responds with a *counterexample* x from the symmetric difference $H \triangle C_T := (C_T \backslash H) \cup (H \backslash C_T)$. If $x \in C_T \backslash H$ then it is called a *positive* counterexample, if $x \in H \backslash C_T$ then it is called a *negative* counterexample.

A learning algorithm for $\mathcal{C}$ is any mapping A which produces hypotheses

$$H_{i+1}^A := A(H_1^A, \ldots, H_i^A, x_1, \ldots, x_i)$$

from $\mathcal{C}$ which may depend on the previous hypotheses H_j^A and the counterexamples $x_j \in H_j^A \triangle C_T$ received. As in this paper we only consider deterministic learning algorithms, the previous hypotheses may be suppressed as arguments of A.

The *learning complexity* of a learning algorithm A is defined by

$$\begin{aligned} LC(A) := \max \{ i \in \mathbf{N} \mid & \text{ there is some } C_T \in \mathcal{C} \text{ and some choice of} \\ & \text{counterexamples } x_j \in H_j^A \triangle C_T \text{ for} \\ & j = 1, \ldots, i-1 \text{ such that } H_i^A \neq C_T \}. \end{aligned}$$

Note that in the definition of $LC(A)$ the amount of computation performed by A to determine the next hypothesis is not taken into consideration. The attention is focused on the amount of interaction between the learner and the environment. As it will be noted later on, the learning algorithms presented in this paper (with one exception) have the additional property that they are *computationally feasible* as well, in the sense that the required number of computation steps is bounded by a polynomial function of the input parameters.

The *learning complexity* $LC(\mathcal{C})$ of the concept class $\mathcal{C}$ is defined by

$$LC(\mathcal{C}) := \min\{LC(A) \mid A \text{ is a learning algorithm for } \mathcal{C}\}.$$

In Sections 3 and 4 we consider the domain $\{0,\ldots,n-1\}^d$ and the concept class

$$\text{HALFSPACE}_n^d := \{C \subseteq \{0,\ldots,n-1\}^d \mid \text{for some } \mathbf{w} \in \mathbf{R}^d, t \in \mathbf{R} \text{ it holds that } C = H_{\mathbf{w},t}\},$$

where for $\mathbf{w} = (w_1,\ldots,w_d) \in \mathbf{R}^d$, $t \in \mathbf{R}$

$$H_{\mathbf{w},t} := \left\{(x_1,\ldots,x_d) \in \{0,\ldots,n-1\}^d \mid \sum_{i=1}^{d} w_i x_i \geq t\right\}.$$

The definition of the other learning problems discussed will be given in the corresponding sections of the paper.

3. A polynomial time on-line learning algorithm for threshold gates.

In this section we present learning algorithms for threshold gates with d inputs from $\{0,\ldots,n-1\}$, or equivalently, algorithms for learning halfspaces over $\{0,\ldots,n-1\}^d$.

There are several learning algorithms in artificial intelligence which proceed by updating the *version space*, i.e. the set of concepts not ruled out by previous counterexamples (Mitchell (1977), Cohen and Feigenbaum (1982)). The learning algorithms to be presented proceed similarly by maintaining an *approximation* of the version space. Typically, the "real" version space is slightly enlarged to obtain a more tractable representation. The goal of a fast learning algorithm is to find a new hypothesis with the property that *every* counterexample to this hypothesis eliminates a *large* part of the version space. This guarantees that the version space shrinks fast, and the target concept is identified in few learning steps.

In the case of learning halfspaces we represent the considered concepts $C \in \text{HALFSPACE}_n^d$ by suitable points in $\mathbf{R}^d$. More precisely, we represent C by a point

$\mathbf{w} \in \mathbf{R}^d$ such that $C = F_{\mathbf{w},1} \cap \{0,\ldots,n-1\}^d$, where $F_{\mathbf{w},1} = \{\mathbf{x} \in \mathbf{R}^d \mid \sum_{i=1}^{d} x_i w_i \geq 1\}$. Hence we may view the version space as a subset of $\mathbf{R}^d$, and we can exploit its *geometrical structure*. The key property of the chosen representation of the version space is the following: any counterexample to some hypothesis $C \in \text{HALFSPACE}_n^d$ (represented by some point $\mathbf{w} \in \mathbf{R}^d$) does not just eliminate the point $\mathbf{w}$, but a whole halfspace in $\mathbf{R}^d$ that contains the point $\mathbf{w}$. Hence in order to guarantee fast learning, it is sufficient to choose as next hypothesis some $C \in \text{HALFSPACE}_n^d$ such that its representation $\mathbf{w}$ lies in the *center* of the remaining version space in $\mathbf{R}^d$. Different algorithms are obtained by using different notions of a center. In each case the essential property of the center $\mathbf{w} \in \mathbf{R}^d$ of the version space is that *every* halfspace that contains $\mathbf{w}$ contains a large portion of the version space.

This strategy is closely related to the *ellipsoid method* for *linear programming*, or more generally, for *convex optimization* (Khachian (1979), see Grötschel, Lovász and Schrijver (1988), Schrijver (1986)). It may be viewed as an extension of the paradigm of binary search to higher dimensions. The learning algorithm for halfspaces that is induced by the ellipsoid method maintains a d-dimensional ellipsoid which contains the current version space. In a learning step the algorithm poses that halfspace which is represented by the center of the ellipsoid as the next hypothesis. It turns out that a counterexample to this hypothesis eliminates a whole half-ellipsoid from the version space. The remaining half-ellipsoid is included in a new ellipsoid (having smaller volume than the previous one), and the algorithm proceeds to the next learning step. In order to show that this process identifies the target concept $C_T \in \text{HALFSPACE}_n^d$ after polynomially many iterations, one shows that the version space contains a small ball $B_T \subseteq \mathbf{R}^d$ such that *every* point in B_T is a representation for C_T. Obviously the described algorithm will identify C_T at the latest when the volume of the current ellipsoid is as small as the volume of B_T.

More generally, it holds that every convex optimization algorithm which can be formulated in a certain *oracle* model (to be defined below) can be used directly as a halfspace learning algorithm. Therefore we first formulate the convex feasibility problem in the oracle model and describe the reduction of halfspace learning to this problem. Having this reduction, one can simply "plug in" any convex feasibility algorithm which uses the oracle model, to obtain a learning algorithm for halfspaces. In particular, the convex feasibility algorithm of Vaidya (1989) leads to the halfspace learning algorithm which is optimal up to a factor of $\log d$.

The goal of the convex feasibility problem is to find a point in an unknown convex body $P \subseteq \mathbf{R}^d$ which has a *guarantee* r, and is given by a *separation oracle*.

A guarantee $r \in \mathbf{N}$ for the convex body P is a number such that the *volume* of P within the ball of radius r around $\mathbf{0}$ is at least r^{-d}. Having a guarantee means that it is

known in advance that the unknown convex body is "not too small".

Information about P can be obtained through the separation oracle. The oracle can answer queries of the form "$\mathbf{y} \in P$?", where $\mathbf{y} \in \mathbf{R}^d$. If $\mathbf{y} \in P$ then the response to the query is "yes", and the problem is solved. Otherwise the response is a halfspace $F = \{\mathbf{x} : \mathbf{c}^T\mathbf{x} \geq b\}$ such that $P \subseteq F$ but $\mathbf{c}^T\mathbf{y} \leq b$ (such a halfspace separates $\mathbf{y}$ from P, hence the name of the oracle). Furthermore, it is assumed that there is a polynomial p such that if $\mathbf{y}$ is m bits long then the response $(\mathbf{c}, b)$ of the oracle is not longer than $p(m, \log r, d)$ bits. In the case of separation oracles that arise in the learning problem considered here one has $p(m, \log r, d) = m + \log r + d$.

We will reduce the learning of halfspaces to the following problem in combinatorial optimization.

Convex feasibility problem: given a separation oracle and a guarantee r for an unknown convex body P, find a point in P.

We consider algorithms for the convex feasibility problem for which the number of oracle queries and the total number of computation steps performed are both bounded by functions of d and $\log r$. The *query complexity* $q(d, \log r)$ of such an algorithm is the number of queries required in the worst case. The *time complexity* $t(d, \log r)$ is the number of computation steps performed in the worst case.

Theorem 3.1. Assume that there is an algorithm A^* solving the convex feasibility problem with query complexity $q(d, \log r)$ and time complexity $t(d, \log r)$. Then there is a learning algorithm A for HALFSPACE_n^d such that $LC(A) \leq q\big(d, 4d(\log d + \log n + 3)\big) + 1$ and the total amount of computation performed by A is at most $t\big(d, 4d(\log d + \log n + 3)\big) + q\big(d, 4d(\log d + \log n + 3)\big) \cdot p(d, \log n)$ for some polynomial p.

Proof. Let the first hypothesis of A be $\{0, \ldots, n-1\}^d$. If a counterexample is received then this must be a negative counterexample $\mathbf{x}^* = (x_1^*, \ldots, x_d^*)$. This point will be considered the origin by transforming the domain to $U = \underset{i=1}{\overset{d}{\times}} \{-x_i^*, \ldots, n-1-x_i^*\}$. The subsequent steps of the learning algorithm will be described as learning a halfspace over this domain. It is straightforward to translate the hypotheses and the counterexamples between the two domains, contributing the overhead $q\big(d, 4d(\log d + \log n + 3)\big) \cdot p(d, \log n)$ to the total number of computation steps.

Let $C_T = \left\{\mathbf{x} \in \{0, \ldots, n-1\}^d \mid \sum_{i=1}^{d} w_i x_i \geq t\right\}$ be the target concept. Because of the preceding first step of the learning algorithm and the subsequent transformation of the coordinate system we have $\mathbf{0} \notin C_T$. This implies that $t > 0$. Hence by multiplying with a positive constant it may be assumed that $t = 1$, i.e. $C_T = F_{\mathbf{w},1} \cap U$. Define

$$\text{SOL}_{C_T} := \{\mathbf{u} \in \mathbf{R}^d \mid C_T = F_{\mathbf{u},1} \cap U\}.$$

If $\mathbf{x} \in C_T$ (resp. $\mathbf{x} \notin C_T$) then every $\mathbf{u} \in \mathrm{SOL}_{C_T}$ must satisfy $\mathbf{u}^T\mathbf{x} = \sum_{i=1}^{d} u_i x_i \geq 1$ (resp. $\mathbf{u}^T\mathbf{x} < 1$). Conversely, if $\mathbf{u}$ satisfies these conditions for every $\mathbf{x} \in U$ then it belongs to SOL_{C_T}. Hence

$$\mathrm{SOL}_{C_T} = \bigcap_{\mathbf{x} \in C_T} F_{\mathbf{x},1} \cap \bigcap_{\mathbf{x} \in U \setminus C_T} \overline{F_{\mathbf{x},1}}.$$

Thus SOL_{C_T} is the intersection of n^d halfspaces (the halfspaces corresponding to $U \setminus C_T$ are open). In particular, SOL_{C_T} is convex. Hence A^* can be used to find a point in SOL_{C_T}.

We need some standard bounds for linear inequalities to get a guarantee for SOL_{C_T}. The proof of the following lemma is given in the Appendix for completeness.

Lemma 3.2. $2^{4d(\log d + \log n + 3)}$ is a guarantee for SOL_{C_T}.

Proof. See Lemma A2 in the Appendix. □

The learning algorithm A proceeds as follows. It simulates A^* by presenting the hypothesis $F_{\mathbf{y},1} \cap U$ whenever A^* asks a query "$\mathbf{y} \in \mathrm{SOL}_{C_T}$?". Hence to prove the theorem it is sufficient to show that a response to a query "$\mathbf{y} \in \mathrm{SOL}_{C_T}$?" of algorithm A^* can be obtained from the counterexample $\mathbf{z}$ received after presenting the hypothesis $F_{\mathbf{y},1} \cap U$. Of course, if this hypothesis is correct, the learning process terminates.

Case 1. : $\mathbf{z}$ is a positive counterexample.

This means that $\mathbf{y}^T\mathbf{z} < 1$ but $\mathbf{z} \in C_T$. Hence for every $\mathbf{u} \in \mathrm{SOL}_{C_T}$ it holds that $\mathbf{u}^T\mathbf{z} \geq 1$. Thus $F_{\mathbf{z},1}$ is a separating halfspace for $\mathbf{y}$.

Case 2. : $\mathbf{z}$ is a negative counterexample.

In this case $\mathbf{y}^T\mathbf{z} \geq 1$ but $\mathbf{z} \notin C_T$. Hence for every $\mathbf{u} \in \mathrm{SOL}_{C_T}$ it holds that $\mathbf{u}^T\mathbf{z} < 1$. Thus $F_{-\mathbf{z},-1}$ is a separating halfspace for $\mathbf{y}$.

Therefore A can indeed simulate A^*, and the bounds for the complexity of A^* directly imply the claimed bounds for the complexity of A, taking also into account the overhead mentioned at the beginning of the proof. □

Now we turn to the informal description of algorithms for the convex feasibility problem.

Algorithm 1: the ellipsoid method (Khachian (1979), Grötschel, Lovász and Schrijver (1988)).

The algorithm maintains a d-dimensional ellipsoid containing the unknown convex body. Initially the ellipsoid is the ball of radius r around the origin. The next query to the

separation oracle is the center of the ellipsoid. The response to the query determines a halfspace. The intersection of this halfspace and the ellipsoid forms a half-ellipsoid containing the convex body. This half-ellipsoid is then included in an ellipsoid. As calculations are performed with finite precision, it is necessary to slightly enlarge this ellipsoid to compensate for rounding errors. This slightly enlarged ellipsoid is used in the next iteration. The volume of the new ellipsoid is smaller by a factor $e^{-\frac{1}{5d}}$ than the volume of the original ellipsoid. Hence as the volume of the initial ball is at most $(2r)^d$ and the volume of the unknown convex body within this ball is at least r^{-d}, the number of iterations necessary is at most $10d^2 \ln 2r$. In one iteration one has to perform $O(d^3)$ arithmetic operations assuming that the standard algorithm is used for multiplying matrices. It is sufficient to do the calculations with a precision of $O(d^2 \log r)$ bits. Hence, again assuming that the standard algorithms are used for the multiplication and division of numbers, the time complexity of the algorithm is $O\big(d^9(\log r)^3\big)$.

Applying Theorem 3.1 with the ellipsoid method one obtains a halfspace learning algorithm with learning complexity $O\big(d^3(\log d + \log n)\big)$ and with time complexity polynomial in d and $\log n$.

We refer to the Appendix for further details.

Algorithm 2: Vaidya's algorithm (Vaidya (1989)).

The algorithm maintains a full-dimensional polytope P defined by some of the halfspaces obtained as responses to previous queries. Thus P always contains the unknown convex body. The next query to the separation oracle is an *approximation* to the so-called *volumetric center* of P. Also, in order to prevent P from becoming too complicated, some of the inequalities defining P may be dropped.

The volume of P decreases by a constant factor on the average, where the constant is independent of d. Therefore the query complexity of the algorithm is $O(d \log r)$, and so one obtains a speedup d in query complexity compared to the ellipsoid method. The time complexity of the algorithm is given in Vaidya (1989) in the unit cost model, i.e. each arithmetic operation is counted as one step. The number of arithmetic operations needed in one iteration is $O(d^3)$, again assuming that we use standard matrix multiplication. The precision required is polynomial in d and $\log r$ (Vaidya (1990)). Hence the time complexity of the algorithm is also polynomial in d and $\log r$.

We again refer to the Appendix for further details.

Thus we obtain the following upper bound for the complexity of learning a threshold gate.

Theorem 3.3. $LC(\text{HALFSPACE}_n^d) = O\big(d^2(\log d + \log n)\big)$. The upper bound is achieved by a learning algorithm for which the total number of computation steps is polynomial in d and $\log n$.

Proof. Apply Theorem 2.1 with the convex feasibility algorithm of Vaidya (1989). □

4. Adaptive threshold gates with graded response and real valued inputs.

First we consider learning a function computed by a multithreshold gate, i.e. identifying a *target function* from the class $\text{MULTITHRESHOLD}_n^{d,s}$ defined in the introduction. We recall that a function $f : \{0,\dots,n-1\}^d \to \{0,\dots,s\}$ is in this class if there are $\mathbf{w} = (w_1,\dots,w_d) \in \mathbf{R}^d$ and $\mathbf{t} = (t_1,\dots,t_s) \in \mathbf{R}^s$ with $t_1 \le \dots \le t_s$ such that for every $\mathbf{x} \in \{0,\dots,n-1\}^d$ it holds that

$$f(\mathbf{x}) = \begin{cases} \max\left\{ j \mid \sum_{i=1}^{d} w_i x_i \ge t_j \right\} & \text{, if this set is not empty,} \\ 0 & \text{, otherwise.} \end{cases}$$

As the original learning model is defined for learning sets, one has to specify an extension of this model to the case of learning functions. We distinguish three different extensions M_1, M_2 and M_3, which differ in the type of feedback information that the learner receives when he makes an error. Analogously as before, the learner makes an *error* when for a hypothesis f he encounters an input $\mathbf{x}$, for which $f(\mathbf{x}) \neq f_T(\mathbf{x})$, where f is the current *hypothesis*, i.e. the function computed by the gate with the current values of the parameters $\mathbf{w}$ and $\mathbf{t}$, and f_T is the target function.

In the weakest model M_1 the learner only receives the point $\mathbf{x}$ and the information that $f(\mathbf{x}) \neq f_T(\mathbf{x})$.

In the intermediate model M_2 the learner receives a pair $(\mathbf{x}, a)$, where $a \in \{0,1\}$. If $a = 0$ then $f(\mathbf{x}) > f_T(\mathbf{x})$, i.e. $f(\mathbf{x})$ is *too high*. If $a = 1$ then $f(\mathbf{x}) < f_T(\mathbf{x})$, i.e. $f(\mathbf{x})$ is *too low*.

In the strongest model M_3 the learner receives the pair $\big(\mathbf{x}, f(\mathbf{x})\big)$, i.e. he is told the correct output for $\mathbf{x}$.

The other aspects of these models are the same as before. In particular, a learning algorithm is a function which produces a new hypothesis, i.e. new parameter values $\mathbf{w}'$ and $\mathbf{t}'$, in dependence of the total information received from the previous errors. The learning complexity of an algorithm is the number of errors it can make in the worst case, before identifying the target function f_T. The learning complexity $LC(\text{MULTITHRESHOLD}_n^{d,s})$ of the class of multithreshold functions is the minimum of the learning complexities of learning algorithms for this class.

Note that in the case $s = 1$ each model coincides with the original one.

The question we consider is whether there is a learning algorithm for MULTITHRESHOLD$_n^{d,s}$ which has learning complexity polynomial in d, s and $\log n$.

It turns out that the first model M_1 is too weak for the existence of such an algorithm, even if both d and s are assumed to be constants ($d = s = 2$). A lower bound demonstrating this will be presented in Section 5 (Theorem 5.7). On the other hand, it is possible to give such algorithm in the *intermediate* model M_2 (which appears to be more realistic than the strongest model M_3). This follows from the observation that every halfspace learning algorithm can be used to learn multithreshold functions, and the results of the previous section. A reduction from multithreshold symmetric automata to binary threshold symmetric automata was previously given by Goles and Martinez (1981).

Theorem 4.1. $LC(\text{MULTITHRESHOLD}_n^{d,s}) \leq LC(\text{HALFSPACE}_n^{d+s})$.

Proof. First we define a representation for multithreshold functions which is slightly more general than the one given above. For $\mathbf{u} := (w_1, \ldots, w_d, t_1, \ldots, t_s, b) \in \mathbf{R}^{d+s+1}$ we also write $\mathbf{u} = (\mathbf{w}, \mathbf{t}, b)$. Define $f_{\mathbf{u}} : \{0, \ldots, n-1\}^d \to \{0, \ldots, s\}$ by

$$f_{\mathbf{u}}(\mathbf{x}) = \begin{cases} \max\left\{ j \mid \sum_{i=1}^{d} w_i x_i \geq t_j + b \right\} & , \text{ if this set is not empty,} \\ 0 & , \text{ otherwise.} \end{cases}$$

Clearly $f_{\mathbf{u}}$ is a multithreshold function with thresholds $t'_1 \leq \ldots \leq t'_s$ defined by $t'_j := \min\{t_k + b \mid j \leq k \leq s\}$ for $j = 1, \ldots, s$. For each $\mathbf{u} = (\mathbf{w}, \mathbf{t}, b)$ we associate with $f_{\mathbf{u}}$ the halfspace $C = H_{(\mathbf{w}, -\mathbf{t}), b}$ from HALFSPACE$_n^{d+s}$, i.e.

$$C = \left\{ (\mathbf{x}, \mathbf{y}) \in \{0, \ldots, n-1\}^{d+s} \mid \sum_{i=1}^{d} w_i x_i - \sum_{j=1}^{s} t_j y_j \geq b \right\}.$$

In order to prove the theorem it is sufficient to show that for every learning algorithm A^* for HALFSPACE$_n^{d+s}$ there is a learning algorithm A for MULTITHRESHOLD$_n^{d,s}$ simulating A^*, with $LC(A) \leq LC(A^*)$. Assume that a target function $f_T \in$ MULTITHRESHOLD$_n^{d,s}$ has been fixed. Let $\widetilde{\mathbf{w}} = (\widetilde{w}_1, \ldots, \widetilde{w}_d)$ and $\widetilde{\mathbf{t}} = (\widetilde{t}_1, \ldots, \widetilde{t}_s)$ be parameters with $\widetilde{t}_1 \leq \ldots \leq \widetilde{t}_s$ such that

$$f_T(\mathbf{x}) = \begin{cases} \max\left\{ j \mid \sum_{i=1}^{d} \widetilde{w}_i x_i \geq \widetilde{t}_j + b \right\} & , \text{ if this set is not empty,} \\ 0 & , \text{ otherwise.} \end{cases}$$

We construct a learning algorithm A for MULTITHRESHOLD$_n^{d,s}$ that simulates the given learning algorithm A^* for HALFSPACE$_n^{d+s}$ in a learning process for a corresponding target concept $C_T = H_{(\widetilde{\mathbf{w}}, -\widetilde{\mathbf{t}}), 0} \in$ HALFSPACE$_n^{d+s}$. If A^* presents a hypothesis $H_{(\mathbf{w}, -\mathbf{t}), b}$, then

A presents the hypothesis $f_{\mathbf{u}}$, for $\mathbf{u} = (\mathbf{w}, \mathbf{t}, b)$. (To be precise, A outputs the standard representation $(\mathbf{w}, \mathbf{t}')$ for $f_{\mathbf{u}}$ with $t'_j := \min\{t_k + b | j \leq k \leq s\}$. But this is not relevant for the following.)

Now assume that A receives a pair $(\mathbf{x}, 0)$, i.e., it gets a vector $\mathbf{x}$ such that $j := f_{\mathbf{u}}(\mathbf{x})$ is too high. Since $f_{\mathbf{u}}(\mathbf{x}) = j$ one has $\sum_{i=1}^{d} w_i x_i \geq t_j + b$. This implies that $(\mathbf{x}, \mathbf{e}_j) \in H_{(\mathbf{w},-\mathbf{t}),b}$, where $\mathbf{e}_j := (0, \ldots, 0, 1, 0, \ldots, 0)$ is the j-th unit vector in $\mathbf{R}^s$. Since $f_T(\mathbf{x}) < j$ it holds that $\sum_{i=1}^{d} \widetilde{w}_i x_i < \widetilde{t}_j$, hence $(\mathbf{x}, \mathbf{e}_j) \notin C_T$. Thus $(\mathbf{x}, \mathbf{e}_j)$ is a negative counterexample for the hypothesis $H_{(\mathbf{w},-\mathbf{t}),b}$ of A^*.

If A receives as response a pair $(\mathbf{x}, 1)$, then $j := f_{\mathbf{u}}(\mathbf{x})$ is too low. This implies that $\sum_{i=1}^{d} w_i x_i < t_{j+1} + b$, thus $(\mathbf{x}, \mathbf{e}_{j+1}) \notin H_{(\mathbf{w},-\mathbf{t}),b}$. On the other hand $f_T(\mathbf{x}) > j$ implies that $(\mathbf{x}, \mathbf{e}_{j+1}) \in C_T$ (since $\widetilde{t}_1 \leq \ldots \leq \widetilde{t}_s$). Hence in this case $(\mathbf{x}, \mathbf{e}_{j+1})$ is a positive counterexample for $H_{(\mathbf{w},-\mathbf{t}),b}$.

Thus every hypothesis of A^* is translated to a hypothesis of A, and every counterexample for the hypothesis of A is translated to a counterexample for the hypothesis of A^*. Hence A can receive at most $LC(A^*)$ counterexamples, and thus $LC(A) \leq LC(A^*)$, proving the theorem. □

Corollary 4.2. $LC(\text{MULTITHRESHOLD}_n^{d,s}) = O\big((d+s)^2\big(\log(d+s) + \log n\big)\big)$. The upper bound is achieved by a learning algorithm for which the total number of computation steps is polynomial in d, s and $\log n$.

Proof. This follows directly from Theorems 3.3 and 4.1. Note that in the proof of Theorem 4.1 the computational overhead needed by A compared to A^* is clearly polynomial in d, s and $\log n$. □

Now we turn to another extension of halfspace learning. Instead of the domain $\{0, \ldots, n-1\}^d$ we now consider as domain an arbitrary finite subset $X \subseteq \mathbf{R}^d$ and the concept class

$$\text{HALFSPACE}_X^d := \big\{C \subseteq X \mid \text{for some } \mathbf{w} \in \mathbf{R}^d, t \in \mathbf{R} \text{ it holds that } C = X \cap F_{\mathbf{w},t}\big\}$$

of halfspaces over X. Learning a concept from this class corresponds to learning a threshold gate with d real valued input variables with inputs ranging over X.

How many hypotheses are needed to learn a halfspace over X? If we neither know anything about the number of digits needed to represent the elements of X, nor have any other information ensuring that X is not "too degenerate", then the approach of the previous section cannot be used. Without the guarantee r there is no upper bound on the number of iterations required before finding a point in the solution set. Nevertheless we show that by a different combinatorial argument one can give a learning algorithm of

learning complexity $O(d^2 \log |X|)$ for every $X \subseteq \mathbf{R}^d$. For $X = \{0, \ldots, n-1\}^d$ the learning complexity of this algorithm is comparable to the previous algorithms (the upper bound is slightly better than that of the ellipsoid method). On the other hand the algorithm seems to be inefficient computationally.

Theorem 4.3. For every $X \subseteq \mathbf{R}^d$ it holds that

$$LC(\text{HALFSPACE}_X^d) = O(d^2 \log |X|).$$

Proof. The proof is based on the notion of a centerpoint. A point $\mathbf{w} \in \mathbf{R}^d$ is called a *centerpoint* of a finite set $Y \subseteq \mathbf{R}^d$ if every open halfspace not containing $\mathbf{w}$ contains at most $\frac{d}{d+1}|Y|$ points from Y.

Lemma 4.4. (Yaglom and Boltyanskii (1961), see Edelsbrunner (1987)). Every finite set has a centerpoint. □

Now let $X = \{\mathbf{x}_1, \ldots, \mathbf{x}_m\}$. The first hypothesis of the learning algorithm is again X. If a counterexample is received then this must be a negative counterexample, which will considered to be the origin. Thus we may assume that the target concept is of the form $C_T = X \cap F_{\widetilde{\mathbf{w}},1}$ with $\widetilde{\mathbf{w}} \in \mathbf{R}^d$. The second hypothesis is $\emptyset$. If a counterexample is received then this must be a positive counterexample $\mathbf{x}^*$.

Consider the hyperplanes E_i defined by $\mathbf{x}_i^T\mathbf{y} = 1$ in $\mathbf{R}^d$ for $i = 1, \ldots, m$. These hyperplanes partition $\mathbf{R}^d$ into convex regions such that $\mathbf{w}_1, \mathbf{w}_2 \in \mathbf{R}^d$ belong to the same region iff $\mathbf{x}_i^T\mathbf{w}_1 - 1$ and $\mathbf{x}_i^T\mathbf{w}_2 - 1$ have the same sign ($+, -$ or 0) for every $i = 1, \ldots, m$. One can easily show that for every target concept C_T the points $\mathbf{w}$ such that $C_T = X \cap F_{\mathbf{w},1}$ form one of these regions having a nonempty interior, and some of the lower dimensional regions on its boundary.

Now select points $\mathbf{w}_1, \ldots, \mathbf{w}_s$, one from each full dimensional region contained in the halfspace $\{\mathbf{y} \mid (\mathbf{x}^*)^T\mathbf{y} \geq 1\}$. We shall use the fact that $s = O(m^d)$ (see Edelsbrunner (1987)). The learning algorithm will identify that one of these points which determines the target concept.

During the course of the algorithm we maintain a set CAND of points which are not eliminated by previous counterexamples. Initially CAND $= \{\mathbf{w}_1, \ldots, \mathbf{w}_s\}$. If $|\text{CAND}| = 1$ then the learning process is completed.

If $|\text{CAND}| > 1$ then the next hypothesis is $X \cap F_{\mathbf{w},1}$ where $\mathbf{w}$ is a centerpoint of CAND. We have $\mathbf{w} \neq 0$ since CAND $\subseteq \{\mathbf{y} \mid (\mathbf{x}^*)^T\mathbf{y} \geq 1\}$ and $\mathbf{w}$ belongs to the convex hull of CAND.

If $\mathbf{x}$ is a positive counterexample to this hypothesis then CAND can be updated to CAND $\cap \{\mathbf{y} \mid \mathbf{x}^T\mathbf{y} \geq 1\} \subseteq$ CAND $\cap \{\mathbf{y} \mid \mathbf{x}^T\mathbf{y} > \mathbf{x}^T\mathbf{w}\}$, using $\mathbf{x}^T\mathbf{w} < 1$.

If $\mathbf{x}$ is a negative counterexample then CAND can be updated to $\text{CAND} \cap \{\mathbf{y} \mid \mathbf{x}^T\mathbf{y} < 1\} \subseteq \text{CAND} \cap \{\mathbf{y} \mid \mathbf{x}^T\mathbf{y} < \mathbf{x}^T\mathbf{w}\}$, using $\mathbf{x}^T\mathbf{w} \geq 1$.

Thus in both cases one can apply Lemma 4.4 to conclude that |CAND| decreases by a factor of at least $\frac{d}{d+1}$. Hence, using that $s = O(m^d)$ as noted above, the number of iterations needed to achieve $|\text{CAND}| \leq 1$ is $O\big(\log_{\frac{d+1}{d}}(m^d)\big) = O(d^2 \log m)$. This gives the claimed upper bound for the learning complexity of the algorithm. □

5. Lower bounds to the complexity of learning algorithms for threshold gates.

In this section we show that the learning algorithms of the previous sections are not too far from being optimal. Furthermore we show that there does not exist a fast learning algorithm for learning the intersection of two halfspaces.

First we consider the problem of learning a threshold gate with d inputs from $\{0, \ldots, n-1\}^d$; i.e. learning a concept from HALFSPACE_n^d. In Section 3 a computationally feasible learning algorithm was given for this problem, which identifies the target concept after at most $O\big(d^2(\log d + \log n)\big)$ counterexamples.

The next theorem shows that *every* halfspace learning algorithm (even if it is not computationally feasible) requires in the worst case $\Omega(d^2 \log n)$ counterexamples.

Theorem 5.1. If $d \geq 2$ then $LC(\text{HALFSPACE}_n^d) \geq \binom{d}{2} \log n$. If $d = 1$ then $LC(\text{HALFSPACE}_n^d) \geq \lfloor \log(n+1) \rfloor$.

Proof. In order to describe the argument it is useful to introduce the concept of a *decision tree*. A decision tree T for HALFSPACE_n^d is a rooted binary tree with the following properties:

- each inner node is labelled by an element $\mathbf{x}$ of $\{0, \ldots, n-1\}^d$ (representing a *query* "$\mathbf{x} \in C_T$?"),
- there are two edges leaving each inner node, labelled "yes", resp. "no" (corresponding to possible answers to the query asked at the node),
- each leaf is labelled by a concept from HALFSPACE_n^d in such a way that each concept occurs as the label of exactly one leaf, and the label of every leaf is consistent with all labels along the path leading from the root to the leaf (the number of inner nodes along this path is the depth of the leaf).

Thus a decision tree can be thought of as describing an algorithm to learn a concept from HALFSPACE_n^d in a learning model different from the one used in this paper. In that model the learner can ask queries (called *membership queries*) to determine the membership of elements in the target concept.

The following lemma is a special case of a result of Littlestone (1988).

Lemma 5.2. (Littlestone (1988)). Assume that there is a decision tree T for HALFSPACE$_n^d$ such that the depth of every leaf of T is at least t. Then $LC(\text{HALFSPACE}_n^d) \geq t$.

Proof. Let A be an arbitrary learning algorithm for HALFSPACE$_n^d$. It has to be shown that for some target concept and for some choice of the counterexamples to the hypotheses of A, A needs at least t hypotheses before it can identify the target concept. It is convenient to imagine that the target concept is not decided in advance, but that there is an *adversary* providing the counterexamples to the hypotheses of A. The adversary uses T to determine his responses. The first counterexample is the element at the root. In general, the adversary moves down the tree, always giving the element $\mathbf{x}$ labelling the current node as a counterexample. If $\mathbf{x}$ was a positive (resp. negative) counterexample then he moves along the edge labelled "yes" (resp. "no"). As all concepts occurring as labels of leaves in the subtree below the current node are still candidates for being the target concept, the learning process has to continue until a leaf is reached. Thus by the assumption on T, the adversary forces A to ask at least t hypotheses. □

This lemma is useful as it reduces the problem of proving a lower bound for every learning algorithm to the problem of constructing a single decision tree with the required property. Such a tree can be constructed using the standard proof for the lower bound to the number of threshold functions (see Muroga (1971), generalized to non-Boolean inputs in Hampson and Volper (1990)).

Lemma 5.3. There is a decision tree T_n^d for HALFSPACE$_n^d$ with all its leaves having depth at least $\sum_{i=1}^{d} \lfloor \log((n-1)n^{i-1}+1) \rfloor$.

Proof. We argue by induction on d. For $d = 1$ the claim follows by considering the decision tree corresponding to binary search over the domain $\{0,\ldots,n-1\}$. For the induction step one relates HALFSPACE$_n^d$ to HALFSPACE$_n^{d-1}$ in a way that is easy to visualize for the case $d = 3$. Each concept in HALFSPACE$_n^2$ corresponds to a line L through the plane $\{0,\ldots,n-1\}^2 \times \{0\}$. Consider a hyperplane $H \subseteq \mathbf{R}^3$ that contains L. By rotating H around the axis L one can realize $\left|\{0,\ldots,n-1\}^2 \times \{1,\ldots,n-1\}\right| + 1 = (n-1)\cdot n^2 + 1$ different halfspaces from HALFSPACE$_n^3$ (provided that the angle of L is chosen in such a way that no hyperplane H with $L \subseteq H$ contains more than one point from $\{0,\ldots,n-1\}^2 \times \{1,\ldots,n-1\}$). For a precise proof of the induction step fix for any $d \geq 2$ a decision tree T_n^{d-1} for HALFSPACE$_n^{d-1}$ that exists by the induction hypothesis. Let T be a variation of T_n^{d-1} where every query $(x_1,\ldots,x_{d-1}) \in \{0,\ldots,n-1\}^{d-1}$ is replaced by the query $\{x_1,\ldots,x_{d-1},0\} \in \{0,\ldots,n-1\}^d$.

Now consider an arbitrary leaf ℓ of T_n^{d-1}. If $C_\ell \in$ HALFSPACE$_n^{d-1}$ denotes the concept arriving at ℓ, and ℓ' denotes the leaf corresponding to ℓ in T, then it holds that

$$C_\ell := \{C \in \text{HALFSPACE}_n^d \mid C \cap \{0,\ldots,n-1\}^{d-1} \times \{0\} = C_\ell\}$$

is the class of concepts arriving at ℓ'.

We fix weights $w_1, \ldots, w_{d-1} \in \mathbf{R}$ and a threshold $t \in \mathbf{Q}$ such that $C_\ell = \left\{ \mathbf{x} \in \mathbf{R}^{d-1} \middle| \sum_{i=1}^{d-1} w_i x_i \geq t \right\}$ and $1, w_1, \ldots, w_{d-1}$ are linearly independent over $\mathbf{Q}$. The linear independence can be achieved since for a suitable fixed threshold $t \in \mathbf{Q}$ one can choose the representation $(w_1, \ldots, w_{d-1})$ of C_ℓ arbitrarily from some small ball in $\mathbf{R}^{d-1}$ (see Lemma 3.2).

Now add a d-th weight $w_d \in \mathbf{R}$, and let $\mathbf{w} := (w_1, \ldots, w_d)$. Then

$$H_{\mathbf{w},t} = \left\{ (x_1, \ldots, x_d) \in \{0, \ldots, n-1\}^d \mid \sum_{i=1}^{d-1} w_i x_i + w_d x_d \geq t \right\} =$$

$$= C_\ell \times \{0\} \cup \left\{ (x_1, \ldots, x_d) \in \{0, \ldots, n-1\}^{d-1} \right.$$

$$\left. \times \{1, \ldots, n-1\} \mid \sum_{i=1}^{d-1} w_i x_i + w_d x_d \geq t \right\}.$$

As w_d increases from $-\infty$ to $+\infty$, $H_{\mathbf{w},t}$ increases from $C_\ell \times \{0\}$ to $C_\ell \times \{0\} \cup (\{0, \ldots, n-1\}^{d-1} \times \{1, \ldots, n-1\})$. An element $(x_1, \ldots, x_d) \in \{0, \ldots, n-1\}^{d-1} \times \{1, \ldots, n-1\}$ enters $H_{\mathbf{w},t}$ for $w_d = \frac{t - \sum_{i=1}^{d-1} w_i x_i}{x_d}$. Thus by the assumption on the w_i's we get $(n-1)n^{d-1} + 1$ different concepts, ordered under inclusion corresponding to the linear ordering $\prec$ of $\{0, \ldots, n-1\}^{d-1} \times \{1, \ldots, n-1\}$ according to the value of $\frac{t - \sum_{i=1}^{d-1} w_i x_i}{x_d}$. Hence one can perform a binary search on $\prec$ to identify one of these concepts. As there may be other extensions of $C_\ell \times \{0\}$ which are not in this sequence of concepts, it may be necessary to add further queries after completing the binary search to identify a concept.

Let T_n^d be the decision tree for HALFSPACE_n^d obtained from T by appending to each leaf of T the subtree that implements the queries described above. As the depth of each subtree is at least $\lfloor \log((n-1)n^{d-1}+1) \rfloor$, Lemma 5.3 follows from the induction hypothesis. □

To complete the proof of Theorem 5.1 note that the bound for $d = 1$ follows directly from considering binary search. If $d \geq 2$, $n \geq 3$ then

$$\sum_{i=1}^{d} \lfloor \log((n-1)n^{i-1} + 1) \rfloor \geq d \log(n-1) + \binom{d}{2} \log n - d \geq \binom{d}{2} \log n,$$

and if $d \geq 2$, $n = 2$ then

$$\sum_{i=1}^{d} \lfloor \log(2^{i-1}+1) \rfloor = \binom{d}{2}. \qquad \square$$

With the same induction argument as in the proof of Lemma 5.3 one can derive the following lower bound for the number of concepts in $\mathrm{HALFSPACE}_n^d$. This lower bound is a slight improvement of the lower bound of $n^{\binom{d}{2}}$ in Hampson and Volper (1990). We will use this lower bound in the proof of Theorem 6.1.

Proposition 5.4. $|\,\mathrm{HALFSPACE}_n^d\,| \geq n^{\binom{d}{2}} \cdot (n-1)^d$. $\square$

We note that the lower bound of Theorem 5.1 holds for a larger class of learning algorithms, where the learner may propose *arbitrary* subsets of the domain $\{0,\ldots,n-1\}^d$ as hypotheses. It is easy to see that Lemma 5.2 remains valid with the same proof. (Littlestone (1988) also proved a converse of the lemma for this class of algorithms, see also Maass and Turán (1990 b).) Hence one obtains the following lower bound result.

Theorem 5.5. Let A be a learning algorithm for $\mathrm{HALFSPACE}_n^d$ which is allowed to use arbitrary subsets of $\{0,\ldots,n-1\}^d$ as hypotheses. Then for some target concept C_T and some choice of the counterexamples, the number of hypotheses used by A to learn C_T is at least $\binom{d}{2}\log n$ if $d \geq 2$, and at least $\lfloor \log(n+1) \rfloor$ if $d = 1$. $\square$

Comparing the lower bound with the $O\big(d^2(\log d + \log n)\big)$ upper bound of Theorem 3.4, one can conclude that allowing arbitrary hypotheses does not increase significantly the speed of on-line learning for halfspaces. There are classes for which is not the case (see Angluin (1988), Maass and Turán (1990 b) for surveys of the power of different formal models of on-line learning).

We note that recent results of Maass (1991) and Littlestone imply that the lower bound (multiplied by $\frac{1}{2}$) remains valid even if the learner can use randomization and the environment is assumed to present the examples in an oblivious manner.

In the remainder of this section we will present lower bounds for on-line learning of threshold circuits and multithreshold gates.

First we consider on-line learning for a simple type of *threshold circuit*, which is the conjunction of two threshold gates having fan-in 2, where the inputs of these threshold gates are from $\{0,\ldots,n-1\}$. The subsets accepted by the circuit with different choices of the weights and thresholds form the concept class

$$2-\mathrm{HALFSPACE}_n^2 := \{C \cap C' \mid C \in \mathrm{HALFSPACE}_n^2\}.$$

As $LC(\mathrm{HALFSPACE}_n^2) = O(\log n)$, it would be interesting to have a similarly efficient learning algorithm for $2-\mathrm{HALFSPACE}_n^2$ as well. It turns out that this is not possible.

Theorem 5.6. $LC(2-\text{HALFSPACE}_n^2) = \Omega(n)$.

Proof. We describe an adversary strategy which forces every learning algorithm to use $\Omega(n)$ hypotheses before it can identify the target concept.

Let $Q := \{(i,j) \in \{0,\ldots,n-1\}^2 \mid i \in \{0,n-1\} \text{ or } j = \{0,n-1\}\}$ be the perimeter of the domain $\{0,\ldots,n-1\}^2$ and $U := \{(\lfloor \frac{n}{2} \rfloor + k, \lfloor \frac{n}{2} \rfloor + \ell) \mid k = 0,1,\ \ell = 0,1\}$ be the four corners of the unit square in the middle of the domain.

We observe that if H is a concept from $2-\text{HALFSPACE}_n^2$ such that $U \subseteq H$ then it holds that $H \cap Q \neq \emptyset$. Indeed, consider a representation of H as $F_{\mathbf{w}_1,t_1} \cap F_{\mathbf{w}_2,t_2} \cap \{0,\ldots,n-1\}^2$, where $F_{\mathbf{w}_1,t_1}$ and $F_{\mathbf{w}_2,t_2}$ are halfplanes in $\mathbf{R}^2$. As the convex hull of U contains a circle of diameter 1, $F_{\mathbf{w}_1,t_1} \cap F_{\mathbf{w}_2,t_2}$ either contains one of the four cornerpoints $\{0,0\},\ldots,\{n-1,n-1\}$ from Q, or it contains a segment of length at least 1 on one of the sides of the square determined by the cornerpoints. The latter implies that it contains a point from Q.

The adversary uses the following strategy. If the hypothesis $H \in 2-\text{HALFSPACE}_n^2$ is such that $U \backslash H \neq \emptyset$ then he responds with an element of $U \backslash H$ as a positive counterexample. Otherwise the observation above implies that $H \cap Q \neq \emptyset$, and the response of the adversary is any element from $H \cap Q$ given as a negative counterexample.

Let us call a concept from $2-\text{HALFSPACE}_n^2$ a *strip* if it is of the form $S \cap \{0,\ldots,n-1\}^2$, where S is bounded by two parallel lines touching the circumscribed circle of U. Clearly there are $\Theta(n)$ strips, and each element of Q is contained in only constantly many strips.

Now a positive counterexample of the adversary does not eliminate any strip as a candidate for being the target concept. A negative counterexample $\mathbf{x}$ from Q rules out only those concepts which contain $\mathbf{x}$, hence it eliminates only contantly many strips. Thus every learning algorithm is forced to use $\Omega(n)$ hypothesis before being able to identify the target concept. □

Finally we show that the same argument implies a negative result for learning multi-threshold gates, as mentioned in Section 4.

We consider learning a target function f_T from $\text{MULTITHRESHOLD}_n^{2,2}$, i.e. a multi-threshold function with inputs from $\{0,\ldots,n-1\}^2$, having 2 thresholds, assuming the weak model M_1. Here the response to each hypothesis f is counterexample $\mathbf{x}$ such $f(\mathbf{x}) \neq f_T(\mathbf{x})$. Thus the learner is not told, whether $f(\mathbf{x}) > f_T(\mathbf{x})$ or $f(\mathbf{x}) < f_T(\mathbf{x})$.

Theorem 5.7. Every algorithm for learning a function from $\text{MULTITHRESHOLD}_n^{2,2}$ in the model M_1 requires $\Omega(n)$ hypotheses for some target function and some choice of the counterexamples.

Proof. A *strip* considered in the proof of Theorem 5.6 corresponds to two multithreshold functions with two thresholds, each assigning value 1 to points between the parallel lines, and 0, resp. 2 to the other two parts of the domain.

The adversary strategy is the following. If for the hypothesis function f there is some $\mathbf{x} \in U$ such that $f(\mathbf{x}) \neq 1$, then $\mathbf{x}$ is given as a counterexample. Otherwise, as outlined above, there must be a $\mathbf{y} \in Q$ such that $f(\mathbf{y}) = 1$, and this element is given as a counterexample.

As there are $\Theta(n)$ multithreshold functions corresponding to strips and each counterexample eliminates only constantly many from being a candidate for the target function, every learning algorithm is forced to present $\Omega(n)$ hypotheses. □

6. Distributed learning algorithms.

In this section we consider algorithms for learning a threshold gate which are *distributed* in the sense that the weights $w_1, \ldots, w_d$ and the threshold t are controlled by separate processors, with some limited amount of communication. Such learning algorithms are of particular interest in the context of computational brain models, where emphasis is on learning without a global control.

There are several important examples of distributed learning algorithms for threshold gates such as the perceptron algorithm (also called Hebb's rule, see Rosenblatt (1962), Minsky and Papert (1988), Rumelhart and McClelland (1986)) and the Winnow algorithms (Winnow 1 and Winnow 2) of Littlestone (1988). All these learning algorithms have in common that w_i remains unchanged if for the current counterexample $\mathbf{x} = (x_1, \ldots, x_d)$ it holds that $x_i = 0$. Otherwise, if $\mathbf{x}$ is a positive counterexample then the perceptron algorithm replaces w_i by $w_i + 1$, and the Winnow algorithms replace w_i by αw_i (for some fixed constant $\alpha > 1$); if $\mathbf{x}$ is a negative counterexample then the perceptron algorithm replaces w_i by $w_i - 1$, Winnow 1 replaces w_i by 0, and Winnow 2 replaces w_i by $\frac{w_i}{\alpha}$. These algorithms have in common that there is a bounded number of different updating operations and that these operations commute. Hence they are *k-bounded* according to the following definition. We will show in Theorem 6.1 that all k-bounded learning algorithms are inherently slow.

Definition. A learning algorithm A for HALFSPACE_n^d is *k-bounded* (for some $k \in \mathbf{N}$) if the following conditions are satisfied.

(a) There are $d + 1$ sets $S_1, \ldots, S_{d+1}$, where each S_i consists of at most k functions $h : \mathbf{R} \to \mathbf{R}$, such that $h \circ h' = h' \circ h$ for all $h, h' \in S_i$ $(i = 1, \ldots, d+1)$.

(b) The hypotheses of A are updated in the following manner: assume that the s-th hypothesis of A is $H_{\mathbf{w}(s),t(s)}$, where $\mathbf{w}(s) = \big(w_1(s), \ldots, w_d(s)\big)$, and $\mathbf{x}$ is a counterex-

ample to $H_{\mathbf{w}(s),t(s)}$. Then the next hypothesis $H_{\mathbf{w}(s+1),t(s+1)}$ is obtained by setting $w_i(s+1) = h_i\big(w_i(s)\big)$ for $i = 1, \ldots, d$, $t(s+1) = h_{d+1}\big(t(s)\big)$, where $h_i \in S_i$ for $i = 1, \ldots, d+1$ (there is no limitation on the way in which the operations $h_i \in S_i$ are selected in each learning step).

We note that the definition of a k-bounded learning algorithm does not attempt to capture the intuitive notion of a distributed learning algorithm. However a distributed learning algorithm where each processor can receive at any step only one of k possible signals from its environment (i.e. from the part of the input to which it has access, from other processors and from the feedback device), is likely to be k-bounded (provided that the weight-change operations of each processor are commutative). In particular, the perceptron algorithm and the Winnow algorithms are k-bounded for $k = 3$. An example for a distributed learning algorithm for threshold gates that is not k-bounded can be found in Duda and Hart (1973). They discuss in Table 5.1 a variation of Hebb's rule where the increment depends also on the time step at which it occurs. Note that this algorithm requires more global control than Hebb's rule or the Winnow rule, since each processor must have access to a global clock. The following result gives a lower bound for the complexity of all k-bounded learning algorithms.

Theorem 6.1. Let A be a k-bounded learning algorithm for $\mathrm{HALFSPACE}_n^d$. Then A requires $n^{\Omega(\frac{d}{k})}$ hypotheses for some target concept and some choice of the counterexamples.

Proof. As the weight-change operations of A are commutative, the hypothesis of A depends only on *how often* each operation has been applied to the weights and the threshold. Thus within t steps A can produce at most $(t+1)^{k(d+1)}$ different hypotheses. Hence if A can learn any target concept from $\mathrm{HALFSPACE}_n^d$ within t steps it must be the case that

$$(t+1)^{k(d+1)} \geq |\,\mathrm{HALFSPACE}_n^d\,|.$$

From Proposition 5.4 one gets then

$$t+1 \geq |\,\mathrm{HALFSPACE}_n^d\,|^{\frac{1}{k(d+1)}} \geq \left(n^{\binom{d}{2}}(n-1)^d\right)^{\frac{1}{k(d+1)}} = n^{\Omega(\frac{d}{k})}$$

if $d \geq 2$. If $d = 1$ then the bound follows as well, as $|\,\mathrm{HALFSPACE}_n^1\,| \geq n$. □

In several cases it is also of interest to study the efficiency of a halfspace learning algorithm for the case where all target concepts belong to a subclass of $\mathrm{HALFSPACE}_n^d$ (e.g. monomials, in the case $n = 2$). Therefore we formulate a generalization of Theorem 6.1 which is proved by the same argument.

Theorem 6.2. Let A be a k-bounded learning algorithm which can learn any concept from some concept class $\mathcal{C} \subseteq \text{HALFSPACE}_n^d$ with at most t hypotheses. A is allowed to produce hypotheses from HALFSPACE_n^d not belonging to $\mathcal{C}$. Then $t \geq |\mathcal{C}|^{\frac{1}{k(d+1)}} - 1$. □

It is easy to see that the perceptron algorithm requires $n^{\Omega(d)}$ hypotheses to learn some target concepts as it increases each weight by at most 1, and there are concepts requiring weights of size $n^{\Omega(d)}$. This follows by a counting argument from Proposition 5.4; one can also construct concrete examples with this property (Hampson and Volper (1990)). Thus Theorem 6.1 does not imply a new lower bound for this algorithm.

The Winnow algorithms modify the weights by multiplication, and thus they can produce exponential size weights in polynomially many steps. Hence the above counting argument cannot be used to prove a lower bound. On the other hand it follows from Theorem 6.2 that the Winnow algorithms need $2^{\Omega(d)}$ steps to learn some monotone Boolean threshold function, using the fact that there are at least $2^{\binom{d}{2}-d}$ such functions. This argument provides the first lower bound for the Winnow algorithms that is superpolynomial in d.

We close this section with a remark showing that in a sense the perceptron algorithm is an optimal k-bounded distributed halfspace learning algorithm.

Theorem 6.2 implies that if any k-bounded distributed learning algorithm can learn a class $\mathcal{C} \subseteq \text{HALFSPACE}_2^d$ with the number of hypotheses bounded by a polynomial in d, then $|\mathcal{C}| \leq 2^{O(d \log d)}$. The following proposition implies that this upper bound on the size of $\mathcal{C}$ is in fact optimal up to a constant factor in the exponent.

Let p be a polynomial and consider the concept class

$$\mathcal{C}_p := \{C \in \text{HALFSPACE}_2^d \mid C = H_{\mathbf{w},t} \text{ for some } \mathbf{w} = (w_1, \ldots, w_d) \in \mathbf{Z}^d,\ t \in \mathbf{Z} \text{ such that } |w_i| \leq p(d),\ i = 1, \ldots, d\}$$

Proposition 6.3. If $p(d) \geq d$ for every d, then $|\mathcal{C}_p| = 2^{\Theta(d \log d)}$ and the perceptron algorithm can learn any target concept from $\mathcal{C}_p$ with at most $O(d^2 p^2(d))$ hypotheses.

Proof. Consider $H_{\mathbf{w},t} \in \text{HALFSPACE}_2^d$ such that $\mathbf{w} = (w_1, \ldots, w_d)$, $(w_1, \ldots, w_{\lfloor \frac{d}{2} \rfloor})$ is a permutation of $(1, 2, \ldots, \lfloor \frac{d}{2} \rfloor)$, $w_{\lfloor \frac{d}{2} \rfloor + 1} = \ldots = w_d := -1$ and $t := 0$. Then it is easy to see that different permutations correspond to different concepts, thus $|\mathcal{C}_p| = 2^{\Omega(d \log d)}$. The upper bound for $|\mathcal{C}_p|$ is obvious. The upper bound for the learning complexity of the perceptron algorithm on concepts from $\mathcal{C}_p$ follows from the familiar upper bound on the complexity of this algorithm (see Minsky and Papert (1988)). □

7. Some open problems.

Let $C_{k,d}$ be the class of all monotone threshold functions for which only k of the d input variables are "relevant", i.e.

$$C_{k,d} := \{C \subseteq \{0,1\}^d \mid \exists \alpha_1, \ldots, \alpha_d, t \in \mathbf{R} \text{ such that } \alpha_1, \ldots, \alpha_d \geq 0,$$
$$\alpha_i > 0 \text{ for at most } k \text{ indices } i \in \{1, \ldots, d\},$$
$$\text{and } \forall x_1, \ldots, x_d \in \{0,1\}\big((x_1, \ldots, x_d) \in C$$
$$\Leftrightarrow \sum_{i=1}^{d} \alpha_i x_i \geq t\big)\}.$$

Littlestone (1988) has shown that with the Winnow learning algorithm one can learn any target concept C_T from $C_{k,d}$ with $O(\frac{k \log d}{\delta^2})$ hypotheses from HALFSPACE_2^d, where δ is a separability parameter of C_T that may be exponentially small in k. It remains an open problem whether $LC^{\text{HALFSPACE}_2^d}(C_{k,d})$ can be bounded from above by a polynomial in k and $\log d$ (the superscript HALFSPACE_2^d indicates that the learning algorithm for $C_{k,d}$ is allowed to use hypotheses from the larger concept class HALFSPACE_2^d).

It is shown in Theorem 5.6 that $LC(2 - \text{HALFSPACE}_n^2) = \Omega(n)$. However it remains an open problem whether $LC(2 - \text{HALFSPACE}_2^d) = O(d^{O(1)})$. Note that the results of Blum and Rivest (1988) only imply that this polynomial error bound cannot be realized by a polynomial time computable algorithm (provided that $P \neq NP$).

Another important open problem is whether one can achieve polynomial upper bounds in the LC-model for these and other concept classes that correspond to feedforward neural nets of small depth, if one allows a larger hypothesis space (e.g. corresponding to neural nets of larger depth and size).

Finally some interesting problems related to *distributed* learning algorithms for threshold gates are left open by the results of Section 6. In particular it remains open whether the lower bound of Theorem 6.1 remains valid without the condition of commutativity for the weight-change operations (see (a) of the definition of k-boundedness at the beginning of Section 6).

Acknowledgement. We would like to thank David Haussler and Carsten Lund for their valuable remarks.

REFERENCES

ANGLUIN, D. (1988). Queries and concept learning. *Machine Learning,* **2**, 319–342.

BLUM, A. AND RIVEST, R.L. (1988). Training a 3-node neural network is *NP*-complete. *Proceedings of the 1988 Workshop on Computational Learning Theory* (pp. 9–18). San Mateo, CA: Morgan Kaufmann.

BLUMER, A., EHRENFEUCHT, A., HAUSSLER, D. AND WARMUTH, M.K. (1989). Learnability and the Vapnik - Chervonenkis dimension. *Journal of the ACM,* **36**, 929–965.

COHEN, P.R. AND FEIGENBAUM, E.T. (1982). *The Handbook of Artificial Intelligence.* Volume III. Los Altos, CA: William Kaufmann.

CRICK, F. AND ASANUMA, C. (1986). Certain aspects of the anatomy and physiology of the cerebral cortex. In J.L. McClelland and D.E. Rumelhart (Eds.), *Parallel distributed processing.* Vol. II. Cambridge, MA: MIT Press.

DUDA, R.O. AND HART, P.E. (1973). *Pattern classification and scene analysis.* New York: Wiley.

EDELSBRUNNER, H. (1987). *Algorithms in combinatorial geometry.* (EATCS monographs on theoretical computer science, v. **10**). Berlin, New York: Springer.

FELDMAN, J.T. AND BALLARD, D.H. (1982). Connectionist models and their properties. *Cognitive Science,* **6**, 205–254.

GOLES, E. AND MARTINEZ, S. (1981). A short proof on the cyclic behaviour of multithreshold symmetric automata. *Information and Control,* **51**, 91–97.

GRÖTSCHEL, M., LOVÁSZ, L. AND SCHRIJVER, A. (1988). *Geometric algorithms and combinatorial optimization. (Algorithms and Combinatorics;* **2**). Berlin, Heidelberg: Springer.

HAMPSON, S.E. AND VOLPER, D.J. (1990). Representing and learning Boolean functions of multivalued features. *IEEE Transactions on Systems, Man, and Cybernetics,* **20**, 67–80.

HEBB, D.O. (1949). *Organization of behavior.* New York: Wiley.

HOPFIELD, J.J. (1982). Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the National Academy of Sciences of the U.S.A.,* **79**, 2554–2558.

KHACHIAN, L.G. (1979). A polynomial algorithm in linear programming. *Doklady Akademii Nauk SSSR,* **244**, 1093–1096. English translation: *Soviet Mathematics Doklady,* **20**, 191–194.

LITTLESTONE, N. (1988). Learning quickly when irrelevant attributes abound: a new linear-threshold algorithm. *Machine Learning,* **2**, 285–318.

MAASS, W. (1991). On-line learning with an oblivious environment and the power of randomization. *Proceedings of the 1991 Workshop on Computational Learning Theory* (pp. 167–175). San Mateo, CA: Morgan Kaufmann.

MAASS, W. AND TURÁN, GY. (1989). On the complexity of learning from counterexamples. *Proceedings of the Thirtieth Annual Symposium on Foundations of Computer Science* (pp. 262–267). Washington, DC: IEEE Computer Society Press.

MAASS, W. AND TURÁN, GY. (1990 a). On the complexity of learning from counterexamples and membership queries. *Proceedings of the Thirty-First Annual Symposium on Foundations of Computer Science* (pp. 203–210). Washington, DC: IEEE Computer Society Press.

MAASS, W. AND TURÁN, GY. (1990 b). Lower bound methods and separation results for on-line learning models. To appear in *Machine Learning*.

MAASS, W. AND TURÁN, GY. (1990 c). Algorithms and lower bounds for on-line learning of geometrical concepts. Unpublished manuscript.

MCCULLOCH, W.S. AND PITTS, W. (1943). A logical calculus of ideas imminent in neural nets. *Bulletin of Mathematical Biophysics,* **5**, 115–137.

MINSKY, M. AND PAPERT, S. (1988). *Perceptrons: an introduction to computational geometry,* Expanded edition. Cambridge, MA: MIT Press.

MITCHELL, J.M. (1977). Version spaces: a candidate elimination approach to rule learning. *Fifth International Joint Conference on Artificial Intelligence* (305–310).

MUROGA, S. (1971). *Threshold logic and its applications.* New York: Wiley.

NILSSON, N.J. (1965). *Learning machines,* New York: McGraw-Hill.

OLAFSSON, S. AND ABU-MOSTAFA, Y.S. (1988). The capacity of a multilevel threshold function. *IEEE Transactions on Pattern Analysis and Machine Intelligence,* **10**, 277–281.

PAPADIMITRIOU, C.H., AND STEIGLITZ, K. (1982). *Combinatorial optimization: algorithms and complexity.* Englewood Cliff, NJ: Prentice-Hall.

ROSENBLATT, F. (1962). *Principles of neurodynamics.* New York: Spartan Books.

RUMELHART, D.E. AND MCCLELLAND, J.L. (1986). *Parallel distributed processing: explorations in the microstructure of cognition.* Cambridge, MA: MIT Press.

SCHRIJVER, A. (1986). *Theory of linear and integer programming.* New York: Wiley.

SCHWARTZ, E.L. (1990). *Computational Neuroscience.* Cambridge, MA: MIT Press.

VAIDYA, P.M. (1989). A new algorithm for minimizing convex functions over convex sets. *Proceedings of the Thirtieth Annual Symposium on Foundations of Computer Science* (pp. 338–343). Washington, DC: IEEE Computer Society.

VAIDYA, P.M. (1990). Personal communication.

VALIANT, L.G. (1984). A theory of the learnable. *Communications of the ACM,* **27**, 1134–1142.

YAGLOM, M. AND BOLTYANSKII, V.G. (1961). *Convex figures.* English translation. New York: Holt, Rinehart and Winston.

Appendix

Lemma A1 is a standard upper bound for the size of a solution of a system of linear inequalities. In Lemma A2 we restate and prove Lemma 3.2.

Lemma A1. Let C be a halfspace over $\{-n,\dots,n\}^d$. Then there are integers $w_1,\dots,w_d,t$ having absolute value at most $2^{3d(\log d+\log n+3)}$, such that $C = F_{\mathbf{w},t}\cap\{-n,\dots,n\}^d$, where $F_{\mathbf{w},t} = \left\{(x_1,\dots,x_d)\in\mathbf{R}^d \,\middle|\, \sum_{i=1}^{d} w_i x_i \geq t\right\}$.

Proof. Let $\widetilde{\mathbf{w}} = (\widetilde{w}_1,\dots,\widetilde{w}_d)\in\mathbf{R}^d$, $\widetilde{t}\in\mathbf{R}$ such that $C = F_{\widetilde{\mathbf{w}},\widetilde{t}}\cap\{-n,\dots,n\}^d$. Thus

$$\sum_{i=1}^{d}\widetilde{w}_i x_i \geq \widetilde{t} \quad \text{for every } (x_1,\dots,x_d)\in C,$$
$$\sum_{i=1}^{d}\widetilde{w}_i x_i < \widetilde{t} \quad \text{for every } (x_1,\dots,x_d)\in\{-n,\dots,n\}^d\backslash C.$$

It may also be assumed w.l.o.g. that for every $(x_1,\dots,x_d)\in\{-n,\dots,n\}^d\backslash C$

$$\sum_{i=1}^{d}\widetilde{w}_i x_i \leq \widetilde{t}-1.$$

Indeed, as

$$\min\left\{\widetilde{t}-\sum_{i=1}^{d}\widetilde{w}_i x_i \;\middle|\; (x_1,\dots,x_d)\in\{-n,\dots,n\}^d\backslash C\right\} > 0$$

this can be achieved by multiplying with a sufficiently large positive constant. Hence $\widetilde{w}_1,\dots,\widetilde{w}_d,\widetilde{t}$ is a solution to the following system of $(2n+1)^d$ linear inequalities in the variables $y_1,\dots,y_{d+1}$:

$$\text{(1)}\qquad \sum_{i=1}^{d} x_i y_i \geq y_{d+1} \quad \text{for every } (x_1,\dots,x_d)\in C,$$
$$\sum_{i=1}^{d} x_i y_i \leq y_{d+1}-1 \quad \text{for every } (x_1,\dots,x_d)\in\{-n,\dots,n\}^d\backslash C.$$

By setting

$$\text{(2)}\qquad y_i = y_i^{+} - y_i^{-} \qquad \text{for } i = 1,\dots,d+1$$

and adding $(2n+1)^d$ slack variables $\delta_{\mathbf{x}}$ we obtain the system

(3)

$$-\sum_{i=1}^{d} x_i(y_i^+ - y_i^-) + (y_{d+1}^+ - y_{d+1}^-) + \delta_{\mathbf{x}} = 0 \quad \text{for every } \mathbf{x} = (x_1, \dots, x_d) \in C,$$

$$\sum_{i=1}^{d} x_i(y_i^+ - y_i^-) - (y_{d+1}^+ - y_{d+1}^-) + \delta_{\mathbf{x}} = -1 \quad \text{for every } \mathbf{x} = (x_1, \dots, x_d) \in \{-n, \dots, n\}^d \backslash C,$$

$$y_i^+ \geq 0,\ y_i^- \geq 0 \quad \text{for every } i = 1, \dots, d+1,$$

$$\delta_{\mathbf{x}} \geq 0 \quad \text{for every } \mathbf{x} \in \{-n, \dots, n\}^d.$$

Clearly (3) has a solution which can be obtained from $\widetilde{w}_1, \dots, \widetilde{w}_d, \widetilde{t}$. The rows of the matrix A describing the first $(2n+1)^d$ inequalities of (3) are linearly independent, as the coefficients of the δ's form an identity matrix. Hence (3) has a basic feasible solution $\mathbf{z}$ (i.e., a solution for which the columns of A corresponding to positive components are linearly independent; this statement is Theorem 2.1 on p. 31 in Papadimitriou-Steiglitz (1982)). The submatrix B formed by these columns has the following form (after rearranging some rows and columns):

C	O
D	I

where the elements of C and D are $0, \pm 1, \dots, \pm n$, and C is a square matrix of size at most $2(d+1)$. Hence $|\det B|$ and the absolute value of all subdeterminants of B is at most $\big(2(d+1)\big)! n^{2(d+1)}$. The right hand sides of (3) are $0, -1$, and in each column of B there are at most $2d+3$ elements for which the determinant of the matrix obtained by deleting the row and the column containing the given element is nonzero. Thus from Cramer's rule each component z_j of the basic feasible solution $\mathbf{z}$ can be written as $z_j = \frac{u_j}{\det B}$, where $u_j \in \mathbf{Z}$ and $|u_j| \leq \big(2(d+1)\big)! n^{2(d+1)}(2d+3)$. Now an integer solution to (1) can be obtained from $\mathbf{z}$ using (2) and finally multiplying everything with the common denominator $|\det B|$. The absolute values of $w_1, \dots, w_d, t$ obtained in this way are at most

$$2\big(2(d+1)\big)! n^{2(d+1)}(2d+3) < 2^{1+2(d+1)\log\left(2(d+1)\right)+2(d+1)\log n+\log(2d+3)} <$$
$$< 2^{1+3d\log 3d+3d\log n+\log 4d} = 2^{3d(\log d+\log n)+\left(1+(3\log 3)d+\log(4d)\right)} <$$
$$< 2^{3d(\log d+\log n+3)},$$

where we use that $\log 3 < \frac{8}{5}$ and $d \geq 2$ (if $d = 1$, the claim is trivial). □

Lemma A2. $2^{4d(\log d+\log n+3)}$ is a quarantee for SOL_{C_T}.

Proof. We have to show that the volume of $\mathrm{SOL}_{C_T} \cap B$ is at least r^{-d}, where $r = 2^{4d(\log d+\log n+3)}$ and B is the ball of radius r around the origin. Let $\widetilde{\mathbf{w}} \in \mathbf{R}^d$, $\widetilde{t} \in \mathbf{R}$ be values such that $C_T = F_{\widetilde{\mathbf{w}},\widetilde{t}} \cap U$, where $U = \bigtimes_{i=1}^{d} \{-x_i^*, \ldots, n-1-x_i^*\}$. Consider $C := F_{\widetilde{\mathbf{w}},\widetilde{t}} \cap \{-n, \ldots, n\}^d$. By Lemma A1 there exist integers $w_1, \ldots, w_d, t$ of absolute value $\leq 2^{3d(\log d+\log n+3)}$ such that $C = F_{\mathbf{w},t} \cap \{-n, \ldots, n\}^d$. Then clearly $C_T = F_{\mathbf{w},t} \cap U$ and as $\mathbf{0} \notin C_T$ it holds that $t > 0$.

Now let $\mathbf{w}^* := (w_1^*, \ldots, w_d^*)$, where $w_i^* = \frac{2w_i}{2t-1}$ for $i = 1, \ldots, d$. It is obvious that $C_T = F_{\mathbf{w}^*,1} \cap U$.

We will show that

$$\bigtimes_{i=1}^{d} \left[w_i^* - \frac{1}{2tn\,d},\ w_i^* + \frac{1}{2tn\,d} \right] \subseteq \mathrm{SOL}_{C_T} \cap B.$$

Consider any $\mathbf{u} = (u_1, \ldots, u_d) \in \mathbf{R}^d$ with $u_i = w_i^* + \epsilon_i$ for some arbitrary $\epsilon_i \in \left[-\frac{1}{2tn\,d}, \frac{1}{2tn\,d}\right]$. We show that $C_T = F_{\mathbf{u},1} \cap U$. Consider any $(x_1, \ldots, x_d) \in U$.

(a) Assume $\mathbf{x} = (x_1, \ldots, x_d) \in C_T$. Then as $\mathbf{w}^T\mathbf{x} \geq t$, we get

$$\sum_{i=1}^{d} u_i x_i = \sum_{i=1}^{d} w_i^* x_i + \sum_{i=1}^{d} \epsilon_i x_i = \sum_{i=1}^{d} \frac{2w_i}{2t-1} x_i + \sum_{i=1}^{d} \epsilon_i x_i \geq$$
$$\geq \frac{2t}{2t-1} - \frac{1}{2tn\,d} n\,d = 1 + \frac{1}{2t-1} - \frac{1}{2t} > 1.$$

Hence $(x_1, \ldots, x_d) \in F_{\mathbf{u},1}$.

(b) Assume $\mathbf{x} = (x_1, \ldots, x_d) \in U \backslash C_T$. Then as $\mathbf{w}^T\mathbf{x} \leq t-1$, we get

$$\sum_{i=1}^{d} u_i x_i = \sum_{i=1}^{d} \frac{2w_i}{2t-1} x_i + \sum_{i=1}^{d} \epsilon_i x_i \leq \frac{2(t-1)}{2t-1} + \frac{1}{2tn\,d} n\,d = 1 - \frac{1}{2t-1} + \frac{1}{2t} < 1.$$

Hence $(x_1, \ldots, x_d) \notin F_{\mathbf{u},1}$.

Clearly $\mathbf{u} \in B$, thus $\bigtimes_{i=1}^{d} \left[w_i^* - \frac{1}{2tn\,d}, w_i^* + \frac{1}{2tn\,d}\right] \subseteq \mathrm{SOL}_{C_T} \cap B$. The volume of this box is $(tn\,d)^{-d} \geq \left(2^{3d(\log d+\log n+3)} n\,d\right)^{-d} \geq r^{-d}$, proving the lemma. □

Finally we give some details of the ellipsoid method and Vaidya's algorithm.

Algorithm 1. The ellipsoid method.

The bounds and the formulas are from Grötschel, Lovász and Schrijver (1988). An ellipsoid $E(A, \mathbf{a})$ is given by its matrix A and its center $\mathbf{a}$ such that $E(A, \mathbf{a}) := \{\mathbf{x} \in \mathbf{R}^d \mid (\mathbf{x} - \mathbf{a})^T A^{-1} (\mathbf{x} - \mathbf{a}) \leq 1\}$.

The algorithm computes a sequence of ellipsoids $E(A_k, \mathbf{a}_k)$. The next query presented to the oracle is the center $\mathbf{a}_k$.

Initially $A_0 = r^2 I$, $\mathbf{a}_0 = 0$ (where r is the guarantee for the unknown convex body).

If the oracle returns the separating halfspace $F = \{\mathbf{x} \mid \mathbf{c}^T \mathbf{x} \geq b\}$ for the query $\mathbf{a}_k$, then the next ellipsoid $E(A_k, \mathbf{a}_k)$ is given by

$$\mathbf{a}_{k+1} :\approx \mathbf{a}_k + \frac{1}{n+1} \frac{A_k \mathbf{c}}{\sqrt{\mathbf{c}^T A_k \mathbf{c}}},$$

$$A_{k+1} :\approx \frac{2n^2 + 3}{2n^2} \left(A_k - \frac{2}{n+1} \frac{A_k \mathbf{c} \mathbf{c}^T A_k}{\mathbf{c}^T A_k \mathbf{c}} \right).$$

Here "$\approx$" means that all computations have to be done with a precision of $80d^2 \log 2r$ bits. The upper bound for the number of iterations necessary is $10d^2 \log 2r$.

Algorithm 2. Vaidya's algorithm.

The bounds and the formulas are from Vaidya (1989).

The algorithm maintains a pair $(Q, \mathbf{z})$, where $Q = \{\mathbf{x} \in \mathbf{R}^d \mid \mathbf{a}_i^T \mathbf{x} \geq b_i, i = 1, \ldots, m\}$ is a full-dimensional polytope and $\mathbf{z} \in Q$. The point $\mathbf{z}$ is an approximation to the *volumetric center* of Q, which is defined as follows.

The *logarithmic barrier* for Q is the function $-\sum_{i=1}^{m} \ln(\mathbf{a}_i^T \mathbf{x} - b_i)$. The Hessian of this function is given by

$$H(\mathbf{x}) = \sum_{i=1}^{m} \frac{\mathbf{a}_i \mathbf{a}_i^T}{(\mathbf{a}_i^T \mathbf{x} - b_i)^2}.$$

The volumetric center ω of Q is the minimum of

$$F(\mathbf{x}) := \frac{1}{2} \ln(\det(H(\mathbf{x}))).$$

The heuristic for considering ω given in Vaidya (1989) is that this is the point where the ellipsoid $\{\mathbf{y} \in \mathbf{R}^d \mid (\mathbf{y} - \mathbf{x})^T H(\mathbf{x})(\mathbf{y} - \mathbf{x}) \leq 1\} \subseteq Q$ (providing a local quadratic approximation to Q) has the largest volume. Hence using this point as the next query can be expected to eliminate a large portion of Q from consideration.

Let ϵ and δ be fixed constants such that $\delta \leq 10^{-4}$ and $\epsilon \leq 10^{-3}\delta$.

The sequence of pairs generated by the algorithm is $(P_k, \mathbf{z}_k)$. Initially $P_0 := \left\{ \mathbf{x} \in \mathbf{R}^d \mid x_i \geq -r \text{ for } i = 1, \ldots, d, \text{ and } \sum_{i=1}^{d} x_i \leq dr \right\}$ is a simplex and $\mathbf{z}_o$ is the explicitly computable volumetric center of P_0.

For a given pair $(P_k, \mathbf{z}_k)$ the algorithm performs the following computation. (In order to avoid additional indices, assume that $P_k = \{\mathbf{x} \in \mathbf{R}^d \mid \mathbf{a}_i^T\mathbf{x} \geq b_i, i = 1, \ldots, m\}$.)

Compute $H(\mathbf{x})$, the quantities

$$\sigma_i(\mathbf{x}) := \frac{\mathbf{a}_i^T H(\mathbf{x})^{-1}\mathbf{a}_i}{(\mathbf{a}_i^T\mathbf{x} - b_i)^2} \qquad \text{for } i = 1, \ldots, m,$$

$$\nabla F(\mathbf{x}) := -\sum_{i=1}^{m} \sigma_i(\mathbf{x}) \frac{\mathbf{a}_i}{\mathbf{a}_i^T\mathbf{x} - b_i}$$

and

$$Q(\mathbf{x}) := \sum_{i=1}^{m} \sigma_i(\mathbf{x}) \frac{\mathbf{a}_i\mathbf{a}_i^T}{(\mathbf{a}_i^T\mathbf{x} - b_i)^2}.$$

There are two cases.

Case 1. (Querying the oracle)

If $\sigma_i(\mathbf{z}_k) \geq \epsilon$ for $i = 1, \ldots, m$ then present the query $\mathbf{z}_k$ to the oracle. Assume that the oracle provides a separating halfspace $F = \{\mathbf{x} \in \mathbf{R}^d \mid \mathbf{c}^T\mathbf{x} \geq b\}$. Choose a β such that $\mathbf{c}^T\mathbf{z}_k \geq \beta$ and

$$\frac{\mathbf{c}^T H(\mathbf{z}_k)^{-1}\mathbf{c}}{(\mathbf{c}^T\mathbf{z}_k - \beta)^2} = \frac{\sqrt{\delta\epsilon}}{2}.$$

Let P_{k+1} be obtained from P_k by adding the inequality $\mathbf{c}^T\mathbf{x} \geq \beta$. The point z_{k+1} is obtained by executing the following iteration $\lceil 30 \ln(2\epsilon^{-4.5}) \rceil$ times:

$$\mathbf{z} \leftarrow \mathbf{z} - 0.18 Q(\mathbf{z})^{-1} \nabla F(\mathbf{z}).$$

Case 2. (Simplifying the polytope)

If the condition of case 1 does not hold then let i be a value for which $\sigma_i(\mathbf{z}_k)$ is as small as possible $(1 \leq i \leq m)$. Let P_{k+1} be obtained from P_k by deleting the defining inequality $\mathbf{a}_i^T\mathbf{x} \geq b_i$. The point $\mathbf{z}_{k+1}$ is determined by executing the iteration

$$\mathbf{z} \leftarrow \mathbf{z} - 0.18 Q(\mathbf{z})^{-1} \nabla F(\mathbf{z})$$

$\lceil 30 \ln(4\epsilon^{-3}) \rceil$ times.

Now Vaidya (1989) showed that the volume of P_k is at most

$$d\left(\log r + \ln\left(\frac{2d}{\epsilon}\right) + 1\right) + \ln(d+1) - \frac{(k+1)\epsilon}{2}.$$

Hence the guarantee implies that the number of iterations needed is $O(d \log r)$.