Geometrical Concept Learning and Convex Polytopes

Tibor Hegedűs*
Department of Computer Science
Comenius University, 84215 Bratislava, Slovakia
`hegedus@fmph.uniba.sk`

Abstract

We consider exact identification of geometrical objects over the domain $\{0, 1, \ldots, n-1\}^d$, $d \geq 1$ fixed. We give efficient implementations of the general incremental scheme "identify the target concept by constructing its convex hull" for learning convex concepts. This approach is of interest for intersections of halfspaces over the considered domain, as the convex hull of a concept of this type is known to have "few" vertices. In this case we obtain positive results on learning intersections of halfspaces with superset/disjointness queries, and on learning single halfspaces with membership queries. We believe that the presented paradigm may become important for neural networks with a fixed number of discrete inputs.

1 Introduction

We consider exact learning of geometrical objects over the discrete domain $X_n^d = \{0, 1, \ldots, n-1\}^d$. Prior work in this setting has addressed the problem of learning halfspaces, intersections of halfspaces, balls, axis-parallel rectangles, and rectangles in general position. Some of these classes have been shown to be efficiently learnable in standard on-line learning models [8, 9, 18, 19, 20, 21]. However, a weak point of this line of research is the lack of general paradigms for the design of efficient learning algorithms. From this point of view, the following result could be potentially quite important: for any fixed $d \geq 1$, if a concept $c \subseteq X_n^d$ is an intersection of k halfspaces over X_n^d, then the number of vertices of the convex hull of c is polynomial in k and $\log n$ (this was originally proved by Shevchenko [27, 29], and the best known upper bound is implied by the work of Cook *et al.* [10]). Using this result, Shevchenko [30] showed how to "decipher a threshold function of many-valued logic in quasi-polynomial time", *i.e.*, how to learn the class HALFSPACE$_n^d$ of single halfspaces over X_n^d ($d \geq 1$ fixed) with membership queries in *poly*($\log n$) time. In this paper we have a look at the applicability of the result from a more general perspective.

A common feature of many natural geometrical concept classes over X_n^d – such as halfspaces, balls, rectangles, and their intersections – is that they consist of "convex" concepts, *i.e.*, concepts c such that the convex hull of c contains no element of $X_n^d \setminus c$. Clearly, a convex concept is uniquely determined by the vertices of its convex hull. In this paper we investigate the power of the learning paradigm "identify the target concept by finding the vertices of its convex hull" in learning convex concepts over X_n^d. Our approach is incremental: we try to identify the vertices of the convex hull of the target concept one by one. While such a straightforward approach is unlikely to be efficiently implementable if both d and n are unbounded, the situation is better if the dimension d is fixed. We give a high-level algorithm that – having access to an optimization oracle returning "extremal" points of the target concept in given directions – identifies any convex concept over X_n^d ($d \geq 1$ fixed) in time polynomial in $\log n$ and the number of vertices of the convex hull of the target concept. The number of optimization queries is $O\left(m^{\lfloor d/2 \rfloor}\right)$ when learning a concept whose convex hull has m vertices. We view this general algorithm – using quite powerful queries – as a first step in the development of more conventional learning algorithms. We show how to simulate optimization queries by superset or disjointness queries, implying that for any fixed $d \geq 1$ the class POLYTOPE$_n^d$ of all convex concepts over X_n^d is learnable with superset/disjointness queries.

The relevance of these general results is in their applicability for the mentioned natural geometrical classes. We obtain that each of those classes is exactly identifiable in some models by an algorithm that is efficient

*Supported by a Universidad Complutense de Madrid fellowship awarded by the A. Dubcek Center, Comenius University, Bratislava, and by the ALTEC (Algorithms for Future Technologies) project of the EC Cooperative Action IC1000.

with respect to the number of vertices of the convex hull of the target concept. Using the cited bounds on the number of vertices of the convex hull of a concept that is an intersection of halfspaces, we obtain that for any fixed $d \geq 1$, $\alpha \geq 0$, and any function $k(n)$ such that $k(n) = O((\log n)^{\alpha})$, the class $k(n)$-HALFSPACE$_n^d$ of intersections of $k(n)$ halfspaces over X_n^d is learnable with superset/disjointness queries in *poly*($\log n$) time. As a special case, we obtain that the class GP-BOX$_n^d$ of d-dimensional boxes in general position over X_n^d ($d \geq 1$ fixed) is learnable with superset/disjointness queries in *poly*($\log n$) time. The algorithms use "proper" queries (that is, the queries are chosen from the target class), but the final hypotheses are represented as a list of vertices of the convex hull of the target concept. Arguments from [30] show that for the class HALFSPACE$_n^d$ of single halfspaces the above technique can be further simplified in the sense that the superset queries can be efficiently simulated by membership queries. Using this simulation we obtain a specific implementation of the Shevchenko approach from [30], establishing that for any fixed $d \geq 1$ the class HALFSPACE$_n^d$ is learnable in *poly*($\log n$) time making $O\left((\log n)^{(d-1)\lfloor d/2 \rfloor + d}\right)$ membership queries (here the final hypotheses are represented as a list of coefficients of a defining halfspace). We observe that a certain speed-up of this last algorithm can be achieved using parallelism. Then we give applications of our "convex hull approach" in connection with teaching. Among others, we show that for any fixed $d \geq 1$ the teaching dimension of the class HALFSPACE$_n^d$ is $O\left((\log n)^{d-1}\right)$. Finally, we conclude with some related results on learning monotone halfspaces.

Our results usually give the first efficient algorithms for the considered classes in the considered models. The only exception is learning HALFSPACE$_n^2$ – the class of single halfspaces in the plane – with membership queries, where the Bultman-Maass algorithm [8] was proved to use $O(\log n)$ membership queries, as opposed to our $O\left((\log n)^3\right)$ upper bound on the query complexity of the incremental approach given here.

2 Preliminaries

Let X be a finite set called *instance space*. A *concept* is a subset of X, and a *concept class* is a non-empty collection of concepts over X. Our setting is a standard model for *exact learning* [1, 22], where the goal of the learner is to identify an unknown target concept c_t chosen from a known concept class C, making *queries* about c_t. Here we consider *equivalence queries* (questions "Is $h = c_t$?" for some $h \subseteq X$), *membership queries* ("Is $y \in c_t$?" for some $y \in X$), *superset queries* ("Is $c_t \subseteq h$?" for some $h \subseteq X$), and *disjointness queries* ("Is $c_t \cap h = \emptyset$?" for some $h \subseteq X$). The learner receives a *yes/no* answer to a membership query, and a *yes/no* answer plus (if appropriate) a *counterexample* – an example witnessing the correctness of a *no* answer – to the other types of queries. For equivalence, superset and disjointness queries we distinguish the case when the queried hypotheses h are chosen from the target class C. Such queries are called *proper*. A learning algorithm is called *proper* if it uses only proper queries.

A particular *learning model* is specified by determining which types of queries are allowed to use by an algorithm in that model. The learning complexity of a learning algorithm for C is the maximum number of queries it makes, over all possible target concepts $c_t \in C$ and all possible choices of counterexamples. The learning complexity of a class C in a certain model is the minimum learning complexity over all learning algorithms for C in that model (see [22] for formal definitions). We will be interested in the following learning complexities (the notation is from [18, 19, 22]): LC(C) (only proper equivalence queries are allowed), LC-ARB(C) (arbitrary equivalence queries), MEMB(C) (membership queries), LC-MEMB(C) (membership queries plus proper equivalence queries), LC-ARB-MEMB(C) (membership queries plus arbitrary equivalence queries), DISJ(C) (proper disjointness queries), and SUPR(C) (proper superset queries).

We say that a set $T \subseteq X$ is a *teaching set* for $c \in C$ (with respect to class C) if no other concept from C agrees with c on the whole T. Denote TD(c, C) the smallest k such that there is a k-element teaching set for c. The *teaching dimension* TD(C) [11] of a class C is then defined as $\mathrm{TD}(C) = \max_{c \in C} \mathrm{TD}(c, C)$.

In the sequel, we will consider the discrete domain $X_n^d = \{0, 1, \ldots, n-1\}^d$ ($d, n \geq 1$), and denote C_n^d a concept class over X_n^d. For $S \subseteq X_n^d$, denote CH(S) the convex hull of S. We call a concept $c \subseteq X_n^d$ *convex* if $\mathrm{CH}(c) \cap X_n^d = c$. A set $h \subseteq \Re^d$ is a *halfspace over* $\Re^d$ if there exist real weights $w_1, \ldots, w_d$ and a real threshold t such that $h = \left\{ x \mid x \in \Re^d \text{ and } \sum_{i=1}^d w_i x_i \geq t \right\}$. We will be interested in the following geometrical concept classes:

$$\begin{aligned}
\text{POLYTOPE}_n^d &= \{\, \mathrm{CH}(S) \cap X_n^d \mid S \subseteq X_n^d \,\} \\
\text{HALFSPACE}_n^d &= \{\, h \cap X_n^d \mid h \text{ is a halfspace over } \Re^d \,\} \\
k\text{-HALFSPACE}_n^d &= \{ c_1 \cap c_2 \cap \cdots \cap c_k \mid \\
&\qquad\qquad c_1, \ldots, c_k \in \text{HALFSPACE}_n^d \,\} \\
\text{GP-BOX}_n^d &= \{\, c \mid \text{there is a } d\text{-dimensional rectangle} \\
&\qquad\qquad R \subseteq \Re^d \text{ with } R \cap X_n^d = c \,\}
\end{aligned}$$

For shortness, we will call the concepts belonging to the class POLYTOPE$_n^d$ (convex) polytopes, and the elements of k-HALFSPACE$_n^d$ intersections of k halfspaces.

As a learning algorithm actually deals with representations of concepts, we always assume that a concept class is associated with a fixed representation language. The *size* of a concept is then defined as the length of its shortest encoding in the representation language. A learning algorithm for C_n^d is considered to be efficient if it runs in time *poly*(size(c_t), d, $\log n$), where the running time is measured in a logarithmic cost RAM model augmented to allow for queries in some natural way. For the above geometrical concept classes we will consider their natural representations implied by the definitions. A concept $c \in \mathrm{POLYTOPE}_n^d$ is assumed to be represented as a list of vertices of $\mathrm{CH}(c)$, and a concept $c \in k\text{-HALFSPACE}_n^d$ by listing the coefficients of k halfspaces whose intersection defines c. As it is known [29, 21] that a halfspace over X_n^d can be always represented using integer coefficients with $poly(d, \log n)$ long binary encodings, an efficient learning algorithm for $k(n)\text{-HALFSPACE}_n^d$ is required to run in $poly(k(n), d, \log n)$ time, while for $\mathrm{POLYTOPE}_n^d$ in $poly(m, d, \log n)$ time, where m is the number of vertices of the target polytope.

3 Learning convex polytopes

In this section we will design efficient incremental learning algorithms for the class $\mathrm{POLYTOPE}_n^d$ in some special cases. For an incremental algorithm that tries to identify the vertices of the target polytope one by one, a major problem is to use computational resources polynomial in the number of vertices of the target polytope. Even if the algorithm is not always guaranteed to receive vertices of the target polytope as counterexamples, it should be able to find new vertices reasonably fast. This problem is trivially overcome in the Boolean case ($n = 2$ fixed), as it is known [7] that every set $c \subseteq X_2^d$ is convex, and all elements of c are vertices of the convex hull of c (both results are easily proved using the fact that any element of X_2^d can be separated from the rest of X_2^d by a single hyperplane). This implies that $c \in \mathrm{POLYTOPE}_2^d$ is learnable in a straightforward way "from below" (observe that this result is just a reformulation of the fact that Boolean functions represented by lists of positive examples are learnable).

Lemma 3.1. *The class* $\mathrm{POLYTOPE}_2^d$ *is learnable in polynomial time from equivalence (or superset) queries.*

If n is unbounded, then the above algorithm approximating the target concept form below does not work, as the counterexamples received are not necessarily vertices of the target polytope any more. In this case it seems to be necessary to be able to find "extremal" points of the target concept efficiently. We first take a high-level approach and assume that this can be done directly.

Let $C_n^d \subseteq 2^{X_n^d}$ be a concept class, and let $c \in C_n^d$. An *optimization oracle* for c works in the following way: for an input vector $(a_1, \ldots, a_d) \in \mathrm{Q}^d$ it outputs a point $y = (y_1, \ldots, y_d) \in c$ such that

$$\sum_{i=1}^{d} a_i y_i = \min \left\{ \sum_{i=1}^{d} a_i x_i \mid x \in c \right\}$$

if $c \neq \emptyset$, or answers that $c = \emptyset$ otherwise.

Having access to an optimization oracle, one can learn $\mathrm{POLYTOPE}_n^d$ by identifying the vertices of the target polytope one by one. The idea is the following. Assume that we have a list of *some* vertices of $\mathrm{CH}(c_t)$, where c_t is the target concept. The convex hull of these vertices determines a polytope c'. Compute a representation of c' as an intersection of halfspaces. Then try to find a further vertex of $\mathrm{CH}(c_t)$ – not belonging to c' – making optimization queries in the directions determined by the facets of c' (any such vertex must be an "extremal" point of c_t in one of these directions). If no such extremal point is found, then all vertices of $\mathrm{CH}(c_t)$ have been identified and we are done.

However, we have to face two problems here. First, the time complexity of even a single stage of the above algorithm is exponential in d. Second, as a convex polytope can in general have exponentially more facets than vertices, it cannot be even guaranteed that the number of optimization queries is polynomial in the number of vertices of the target polytope. Both these problems can be understood as "the curse of dimensionality", as they are caused by the fact that the dimension d is unbounded.

In the sequel, we will consider the situation when the dimension d is fixed. In this case the above problems disappear, as a convex polytope with m vertices has only $O\left(m^{\lfloor d/2 \rfloor}\right)$ facets (this follows from the McMullen Upper Bound Theorem [23]), and – given the vertices of the polytope – the facets can be efficiently computed. These observations lead to an efficient implementation of the above general incremental approach.

Theorem 3.2. *For any fixed* $d \geq 1$, *there exists an algorithm that learns the class* $\mathrm{POLYTOPE}_n^d$ *in polynomial time using optimization queries. The learning algorithm makes* $O\left(m^{\lfloor d/2 \rfloor}\right)$ *optimization queries for a target polytope with* m *vertices.*

Proof: Consider the *LearnPolytope* algorithm described in Figure 1.
The *Initialize* part of the algorithm – line (5) – is assumed to do the following. If the target polytope $c_t \in \mathrm{POLYTOPE}_n^d$ is of dimension k, then *Initialize* finds $k+1$ vertices of $\mathrm{CH}(c_t)$ that are not contained in any single $k-1$-dimensional affine subspace of $\Re^d$ (this can be achieved using optimization queries similarly as in the second part of the algorithm). For simplicity, the rest of the algorithm assumes that the target polytope c_t is full-dimensional, the *Initialize* part computes $d+1$ vertices of $\mathrm{CH}(c_t)$ that do not lie on a single hyperplane, and stores them in set V (in general, after some transformation of the domain the "k-dimensional" version of *LearnPolytope* would have to be run).
Without loss of generality we will assume that the coefficients of the halfspaces computed in line (8) are integers, and none of them is zero (this can be achieved using standard procedures).
The *Transform* function – line (13) – works as follows. Given an input vector $(a_1, \ldots, a_d)$ – where $a_1, \ldots, a_d$ are non-zero integers –, *Transform* computes the vector $((\alpha_1 + \beta)a_1, \ldots, (\alpha_i + \beta)a_i, \ldots, (\alpha_d + \beta)a_d)$, where $\alpha_d = 1$, $\alpha_i = \left\lceil \left((n-1) \sum_{j=i+1}^{d} \alpha_j |a_j| \right) / |a_i| \right\rceil + 1$

```
The LearnPolytope Algorithm
(1)  begin
(2)    $V \leftarrow \emptyset$; { $V$ contains the vertices of $c_t$ identified so far }
(3)    $H \leftarrow \emptyset$; { $H$ contains the facets of $c_t$ identified so far }
(4)    finished ← false;
(5)    Initialize;
(6)    repeat (*)
(7)      let $p$ be the convex polytope such that $p = \mathrm{CH}(V)$;
(8)      compute a representation of $p$ as an intersection of halfspaces corresponding to its facets;
(9)      let $H_{new}$ be the list of those of the computed halfspaces that are not in $H$;
(10)     NewVertexFound ← false;
(11)     repeat (**)
(12)       choose a halfspace $\sum_{i=1}^d a_i x_i \geq b$ from $H_{new}$ (and remove it from $H_{new}$);
(13)       $(q_1, \ldots, q_d) \leftarrow Transform(a_1, \ldots, a_d)$;
(14)       make an optimization query with the vector $(q_1, \ldots, q_d)$;
(15)       let $\mathbf{y} = (y_1, \ldots, y_d) \in c_t$ be the example returned by the optimization oracle;
(16)       if $\sum_{i=1}^d a_i y_i < b$ then begin
(17)         add $\mathbf{y}$ to $V$;
(18)         NewVertexFound ← true
(19)       end else add the halfspace $\sum_{i=1}^d a_i x_i \geq b$ to $H$
(20)     until (**) NewVertexFound or ($H_{new} = \emptyset$);
(21)     if not NewVertexFound then finished ← true
(22)   until (*) finished;
(23)   output $\mathrm{CH}(V) \cap X_n^d$ as the final hypothesis
(24) end.
```

Figure 1: *An algorithm that learns the class* POLYTOPE$_n^d$ *using optimization queries*

for $i = d-1, \ldots, 1$, and $\beta = 1 + (n-1)\sum_{i=1}^d \alpha_i |a_i|$.
We first prove the correctness of *LearnPolytope*. Let $c_t \in$ POLYTOPE$_n^d$ be the target polytope. We claim that the set V always contains only vertices of $\mathrm{CH}(c_t)$. The claim holds true when V consists of the $d+1$ points computed by the *Initialize* part. Assume now that the claim holds true for some intermediate value of V. If a new point $\boldsymbol{y}$ is added to V, then it must be the case that $\sum_{i=1}^d a_i y_i < b$, where $\mathrm{CH}(V) \subseteq \left\{ \boldsymbol{x} \mid \boldsymbol{x} \in X_n^d \text{ and } \sum_{i=1}^d a_i y_i \geq b \right\}$, *i.e.*, $\boldsymbol{y} \in c_t \setminus \mathrm{CH}(V)$, and $\boldsymbol{y}$ was returned by the optimization oracle to the query with $Transform(a_1, \ldots, a_d)$. The choice of the optimization query ensures that $\boldsymbol{y}$ is the minimum of the set $\{ \boldsymbol{x} \mid \boldsymbol{x} \in c_t$ and $\boldsymbol{x}$ minimizes $\sum_{i=1}^d a_i x_i$ over $c_t \}$ with respect to the following modified lexicographic order (associated with $a_1, \ldots, a_d$): let $\boldsymbol{u}, \boldsymbol{v}$ be two vectors such that $u_{i_0} < v_{i_0}$ and $u_i = v_i$ for $i < i_0$; then $\boldsymbol{u}$ is "smaller" than $\boldsymbol{v}$ if $a_{i_0} > 0$, and $\boldsymbol{v}$ is "smaller" than $\boldsymbol{u}$ if $a_{i_0} < 0$. Now – arguing similarly as in [28] – one can check that this minimality property of $\boldsymbol{y}$ implies that $\boldsymbol{y}$ is necessarily a vertex of $\mathrm{CH}(c_t)$ (notice that if just the optimization query $(a_1, a_2, \ldots, a_d)$ were made, then the oracle would return an *arbitrary* element of c_t minimizing $\sum_{i=1}^d a_i x_i$ over c_t, and not all such elements of c_t are vertices of $\mathrm{CH}(c_t)$).
That is, we have proved that V always contains only vertices of $\mathrm{CH}(c_t)$. Moreover, as it can be easily seen that *LearnPolytope* stops if and only if there are no more elements of c_t outside $\mathrm{CH}(V)$, *i.e.*, if and only if V contains all vertices of $\mathrm{CH}(c_t)$, it follows that *LearnPolytope* exactly identifies the target polytope c_t after finitely many steps.
Next, we prove that if $\mathrm{CH}(c_t)$ has m vertices, then *LearnPolytope* runs in *poly*$(m, \log n)$ time and makes $O\left(m^{\lfloor d/2 \rfloor}\right)$ optimization queries.
The *Initialize* part of the algorithm can be implemented to run in *poly*$(\log n)$ time and make $O(1)$ optimization queries.
As each execution of the (*) repeat cycle results in finding a further vertex of $\mathrm{CH}(c_t)$ (except for the final one), and $\mathrm{CH}(c_t)$ has m vertices, the (*) repeat cycle is carried out $O(m)$ times.
One execution of the (*) cycle has the following time complexity. As V contains at most m elements, computing the polyhedron representation of $\mathrm{CH}(V)$ (line (8)) takes *poly*$(m, \log n)$ time: it suffices to solve a $d \times (d+1)$ system of linear equations with integer coefficients not greater (in absolute value) than $n-1$ for each d-element subset of V, *i.e.*, at most m^d sets. The coefficients of the computed halfspaces can be ensured to be non-zero integers of magnitude *poly*(n) (multiply with suitable values).
The (**) repeat cycle is then carried out *poly*(m) times, and altogether takes time polynomial in m and $\log n$. Observe that the coefficients of the optimiza-

tion queries are also integers of magnitude *poly*(n) – we will use this fact later.
This means that the overall time complexity of *Learn-Polytope* is *poly*($m, \log n$).
Finally, observe that each call to the optimization oracle identifies either a new vertex or a new facet of the target polytope CH(c_t). As – according to the McMullen Upper Bound Theorem [23] – CH(c_t) has $O\left(m^{\lfloor d/2 \rfloor}\right)$ facets, we obtain that the number of calls to the optimization oracle is $O\left(m^{\lfloor d/2 \rfloor}\right)$. ∎

Now we show that the apparently quite powerful optimization queries used in our high-level approach can be replaced by more standard queries. The idea is to simulate an optimization query by superset or disjointness queries using binary search.

Theorem 3.3. *For any fixed $d \geq 1$, there is an algorithm that learns* POLYTOPE$_n^d$ *in polynomial time from superset (or from disjointness) queries. The algorithm makes $O\left(m^{\lfloor d/2 \rfloor} \log n\right)$ superset (disjointness) queries when learning a polytope with m vertices.*

Proof: It suffices to show that every optimization query of the *LearnPolytope* algorithm can be simulated in *poly*($\log n$) time using $O(\log n)$ superset/disjointness queries.
We have seen in the proof of Theorem 3.2 that every optimization query of *LearnPolytope* is of the form $(a_1, \ldots, a_d)$, where all a_i's are integers of magnitude *poly*(n). Denote $m_l = \min\left\{\sum_{i=1}^d a_i x_i \mid \boldsymbol{x} \in X_n^d\right\}$, $m_u = \max\left\{\sum_{i=1}^d a_i x_i \mid \boldsymbol{x} \in X_n^d\right\}$. Then $m_u - m_l =$ *poly*(n), and it is easy to see how to minimize the function $\sum_{i=1}^d a_i x_i$ over c_t using binary search. Let $m \leftarrow \left\lfloor \frac{m_u - m_l}{2} \right\rfloor$. First, make the superset query with $\left\{\boldsymbol{x} \mid \boldsymbol{x} \in X_n^d \text{ and } \sum_{i=1}^d a_i x_i \geq m\right\}$. If we are given a counterexample, then let $m_u \leftarrow m$, otherwise let $m_l \leftarrow m$, and continue the binary search in the interval $\{m_l, \ldots, m_u\}$ analogously. In the disjointness query version of the algorithm the disjointness queries are always the complements of the above superset queries.
In both cases, as the length of the original interval $\{m_l, \ldots, m_u\}$ is *poly*(n), and the value $\sum_{i=1}^d a_i x_i$ is integer for all $\boldsymbol{x} \in X_n^d$, after $O(\log n)$ superset/disjointness queries – and in *poly*($\log n$) time – we find a point $\boldsymbol{y} \in c_t$ that minimizes $\sum_{i=1}^d a_i x_i$ over c_t. This $\boldsymbol{y}$ is then given as the answer to the original optimization query $(a_1, \ldots, a_d)$. ∎

There is a technical problem in the above simulation: the superset/disjointness queries of the algorithm are in the form of halfspaces, instead of polytopes (given by lists of vertices). However, as the queries are about single halfspaces, the corresponding polytope representations can be efficiently constructed using the following result [28].

Lemma 3.4. [28] *For any fixed $d \geq 1$, the vertices of the convex hull of a set $\left\{\boldsymbol{x} \mid \boldsymbol{x} \in X_n^d \text{ and } \sum_{i=1}^d a_i x_i \geq b\right\}$, where the a_i's and b are integers of magnitude poly(n), can be constructed in poly($\log n$) time.*

Proof: We give a self-contained proof based on the *LearnPolytope* algorithm. Given the integer coefficients $a_1, \ldots, a_d, b$ of magnitude *poly*(n), run *Learn-Polytope* for $c_t = \left\{\boldsymbol{x} \mid \boldsymbol{x} \in X_n^d \text{ and } \sum_{i=1}^d a_i x_i \geq b\right\} \in$ POLYTOPE$_n^d$ as the target concept. The key observation is that – according to Corollary 4.3 given in the next section – CH(c_t) has only *poly*($\log n$) vertices. Theorem 3.2 then gives that *LearnPolytope* computes the vertices of CH(c_t) in *poly*($\log n$) time, if we are able to answer the optimization queries in *poly*($\log n$) time. However, in this case an optimization query of *LearnPolytope* can be answered by solving an integer linear programming (ILP) problem (see Schrijver [26]) with a fixed number of variables and inequalities, and integer coefficients of magnitude *poly*(n). As such ILP problems can be solved in *poly*($\log n$) time using Lenstra's algorithm [16, 26], we obtain that the whole convex hull of c_t can be constructed in *poly*($\log n$) time. ∎

4 Learning intersections of halfspaces

We have given general algorithms for learning the class POLYTOPE$_n^d$ for fixed $d \geq 1$. Clearly, these algorithms can be used to learn any class of convex concepts over X_n^d. However, the following complication occurs in this case. Let C_n^d be a class of convex concepts over X_n^d represented *not* as polytopes, but as (say) intersections of halfspaces, balls, *etc.*; the size measure for C_n^d is defined with respect to this representation scheme. We know that a concept $c_t \in C_n^d$ is exactly identifiable in time polynomial in the number of vertices of CH(c_t) and $\log n$, but this becomes interesting only if for every $c \in C_n^d$ the number of vertices of CH(c) is polynomial in size(c) and $\log n$. Surprisingly, this is the case for a basic geometrical concept class: intersections of halfspaces (the "simplicity" of this class is quite in contrast with the fact that previous results on learning intersections of halfspaces were mainly negative in almost all contexts [4, 15, 19, 20, 21]).

Call a set $P \subseteq \Re^d$ a *rational polyhedron* if it is given by finitely many inequalities of the form $\sum_{i=1}^d a_i x_i \geq b$, where $a_1, \ldots, a_d, b$ are rational numbers. The size of an inequality is the number of bits necessary to encode it as a binary string. Given a polyhedron $P \subseteq \Re^d$, write P_I for the convex hull of the grid points belonging to P, *i.e.*, $P_I = \text{CH}(\{\boldsymbol{y} \mid \boldsymbol{y} \in P \text{ and } y_1, \ldots, y_d \text{ are integers}\})$. P_I is called the *integer hull* of P (see Figure 2). Strengthening and generalizing some earlier results of Shevchenko [27], Hayes and Larman [14] and Morgan [24], Cook *et al.* [10] proved the following.

Theorem 4.1. [10] *For any fixed $d \geq 1$, if P is a rational polyhedron in $\Re^d$ which is the solution of a system of*

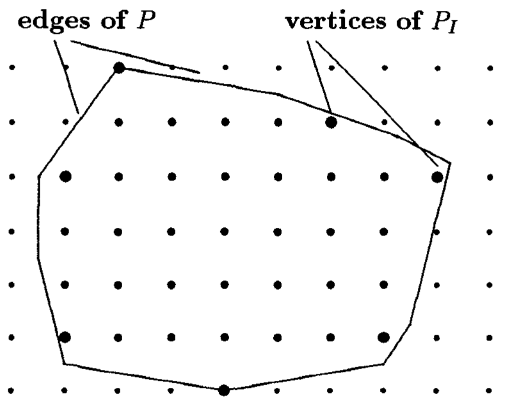

Figure 2: *A polygon P and its integer hull P_I*

at most k linear inequalities, and the size of each defining inequality is at most ρ, then the number of vertices of P_I is $O\left(k^d \rho^{d-1}\right)$.

Proof: (Main idea) Let $P \subseteq \Re^d$ be a polyhedron and P_I its integer hull. Given $\boldsymbol{x}, \boldsymbol{y} \in \Re^d$, call the point $2\boldsymbol{y} - \boldsymbol{x}$ the *reflection of $\boldsymbol{x}$ about $\boldsymbol{y}$*, and $2\boldsymbol{x} - \boldsymbol{y}$ the *reflection of $\boldsymbol{y}$ about $\boldsymbol{x}$* (see Figure 3). If $\boldsymbol{x}, \boldsymbol{y}$ are lattice points, then also $2\boldsymbol{x} - \boldsymbol{y}$ and $2\boldsymbol{y} - \boldsymbol{x}$ are lattice points. Now, partition the lattice points in P into "reflecting sets" $A_1, \ldots, A_l$ such that if two lattice points $\boldsymbol{x}, \boldsymbol{y}$ belong to the same reflecting set A_i (for any i), then at least one of the reflection points $2\boldsymbol{x} - \boldsymbol{y}$ and $2\boldsymbol{y} - \boldsymbol{x}$ belongs to P. Then, clearly, no reflecting set can contain more than one vertex of P_I, thus the number of reflecting sets gives an upper bound on the number of vertices of P_I. It remains to give a decomposition into a small number of reflecting sets (see [10] for details). ∎

Together with some standard bounds [21, 30] on the size of weights needed to represent halfspaces over X_n^d, the above result implies a bound on the number of vertices of the convex hull of a concept $c \subseteq X_n^d$ that is an intersection of halfspaces over X_n^d. For simplicity we focus on intersections of *poly*$(\log n)$ many halfspaces.

Lemma 4.2. [21, 30] *For any fixed $d \geq 1$, if c is a halfspace over X_n^d, then there exist integers $w_1, \ldots, w_d, t$ having absolute value poly(n) such that*

$$c = \left\{ \boldsymbol{x} \mid \boldsymbol{x} \in X_n^d \text{ and } \textstyle\sum_{i=1}^d w_i x_i \geq t \right\}.$$

Corollary 4.3. *For any fixed $d \geq 1$, $\alpha \geq 0$, and any function $k(n)$ such that $k(n) = O((\log n)^\alpha)$, the convex hull of the elements of a concept $c \in k(n)$-HALFSPACE$_n^d$ has $O\left((\log n)^{d(\alpha+1)-1}\right)$ vertices.*

The result given in Theorem 3.3 now implies that the class of intersections of *poly*$(\log n)$ many halfspaces over X_n^d is learnable with superset/disjointness queries in *poly*$(\log n)$ time.

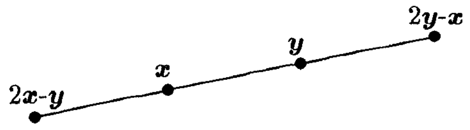

Figure 3: *Reflection points for $\boldsymbol{x}$ and $\boldsymbol{y}$*

Theorem 4.4. *For any fixed $d \geq 1$, $\alpha \geq 0$, and any function $k(n)$ such that $k(n) = O((\log n)^\alpha)$,*

$$\text{SUPR}\left(k(n)\text{-HALFSPACE}_n^d\right) = O\left((\log n)^{(d\alpha+d-1)\lfloor d/2\rfloor+1}\right)$$

and

$$\text{DISJ}\left(k(n)\text{-HALFSPACE}_n^d\right) = O\left((\log n)^{(d\alpha+d-1)\lfloor d/2\rfloor+1}\right).$$

Furthermore, there is a learning algorithm for the class $k(n)$-HALFSPACE$_n^d$ that after poly$(\log n)$ computation steps and $O\left((\log n)^{(d\alpha+d-1)\lfloor d/2\rfloor+1}\right)$ proper superset/disjointness queries outputs the list of vertices of the convex hull of the target concept $c_t \in k(n)$-HALFSPACE $_n^d$.

For instance, we obtain that the class k-HALFSPACE$_n^2$ of intersections of k halfspaces in the plane, $k \geq 1$ fixed, is learnable in *poly*$(\log n)$ time with $O\left((\log n)^2\right)$ proper superset/disjointness queries. Observe that the algorithms in Theorem 4.4 are "logically" proper, but not "representationally" proper, *i.e.*, even though all their queries are proper (they are about single halfspaces), the final hypotheses are not represented as intersections of halfspaces. Of course, the results on the query complexity are not affected by this and are valid for proper learning.

As GP-BOX$_n^d \subseteq 2d$-HALFSPACE$_n^d$, as a special case we obtain results on learning d-dimensional boxes in general position (as the queries are about single halfspaces, the query complexity results again hold true for proper learning).

Corollary 4.5. *For any fixed $d \geq 1$,*

$$\text{SUPR}\left(\text{GP-BOX}_n^d\right) = O\left((\log n)^{(d-1)\lfloor d/2\rfloor+1}\right)$$

and

$$\text{DISJ}\left(\text{GP-BOX}_n^d\right) = O\left((\log n)^{(d-1)\lfloor d/2\rfloor+1}\right).$$

Furthermore, there is an algorithm for GP-BOX$_n^d$ that makes $O\left((\log n)^{(d-1)\lfloor d/2\rfloor+1}\right)$ proper superset/disjointness queries, and after poly$(\log n)$ computation steps outputs the list of vertices of the convex hull of the target concept $c_t \in$ GP-BOX$_n^d$.

An analogous positive learnability result holds for intersections of *poly*$(\log n)$ many boxes in general position as well.

Consider now the class HALFSPACE$_n^d$ of single halfspaces. For this class the above approach can be further simplified in the sense that the superset/disjointness queries can be simulated by membership queries. The key observation here – due to Shevchenko [30] – is the following.

Lemma 4.6. [30] *Let c be a halfspace over X_n^d. Then, for any $a_1, \ldots, a_d, b \in \Re$, the set $S = \left\{ \boldsymbol{x} \mid \boldsymbol{x} \in X_n^d \text{ and } \sum_{i=1}^d a_i x_i < b \right\}$ contains a positive example of c if and only if there exists a vertex of the convex hull of S that is a positive example of c.*

The above lemma indicates how to simulate efficiently the queries of our superset query algorithm by membership queries. The details of the simulation are given below. The resulting algorithm is a specific implementation of the Shevchenko approach from [30].

Theorem 4.7. *For any fixed $d \geq 1$,*

$$\text{MEMB}\left(\text{HALFSPACE}_n^d\right) = O\left((\log n)^{(d-1)\lfloor d/2 \rfloor + d}\right).$$

Furthermore, there is an algorithm for HALFSPACE_n^d that makes $O\left((\log n)^{(d-1)\lfloor d/2 \rfloor + d}\right)$ membership queries, and after $poly(\log n)$ computation steps outputs some integer coefficients $w_1, \ldots, w_d, t$ such that

$$c_t = \left\{ \boldsymbol{x} \mid \boldsymbol{x} \in X_n^d \text{ and } \sum_{i=1}^d w_i x_i \geq t \right\}.$$

Proof: It suffices to show that if the target concept is a single halfspace, then each superset query of the algorithm in Theorem 4.4 can be simulated in $poly(\log n)$ time making $O\left((\log n)^{d-1}\right)$ membership queries.

Let $\left\{ \boldsymbol{x} \mid \boldsymbol{x} \in X_n^d \text{ and } \sum_{i=1}^d a_i x_i \geq b \right\}$ be a superset query of the algorithm, where $a_1, \ldots, a_d, b$ are integers of magnitude $poly(n)$. Denote S the set $\left\{ \boldsymbol{x} \mid \boldsymbol{x} \in X_n^d \text{ and } \sum_{i=1}^d a_i x_i < b \right\}$. Lemma 4.6 says that the correct answer to the query is "no", *i.e.*, there exists a positive example $\boldsymbol{y}$ of the target halfspace c_t satisfying $\sum_{i=1}^d a_i y_i < b$, if and only if one of the vertices of the convex hull of S is a positive example of c_t. Corollary 4.3 gives that the convex hull of S has $O\left((\log n)^{d-1}\right)$ vertices, and using the algorithm given in Lemma 3.4 the list of these vertices can be computed in $poly(\log n)$ time. Now, the original superset query can be answered in the following way. Construct the convex hull of S, and make a membership query about each vertex of this convex hull. If a vertex $\boldsymbol{y}$ is found that is a positive example of c_t, then give $\boldsymbol{y}$ as a counterexample to the superset query. If no such vertex is found, then answer "yes" to the superset query. All this computation takes $poly(\log n)$ time and $O\left((\log n)^{d-1}\right)$ membership queries.

To see that in this case the final hypothesis can be output in the form of coefficients of a defining hyperplane, notice that using the membership oracle for c_t we can also learn the halfspace $X_n^d \setminus c_t$, and then it remains to solve a system of linear inequalities induced by the vertices of $\text{CH}(c_t)$ and $\text{CH}(X_n^d \setminus c_t)$, what can be done in $poly(\log n)$ time. ∎

An interesting feature of the above simulation is that the membership queries used to simulate a superset query are "non-adaptive", *i.e.*, they are independent of each other. This indicates that a speed-up can be achieved using parallelism. Bshouty and Cleve [6] introduced a model for exact learning in parallel, called UPRAM (a variant of PRAM extended to allow for queries). A UPRAM with p processors can ask p queries in one step (see [6] for details). Considering a modification of the above simulation in which the $O((\log n)^{d-1})$ membership queries are performed in a single parallel learning step we obtain the following.

Theorem 4.8. *For any fixed $d \geq 1$, HALFSPACE_n^d is learnable on a UPRAM with $O((\log n)^{d-1})$ processors in $O\left((\log n)^{(d-1)\lfloor d/2 \rfloor + 1}\right)$ parallel – membership query – learning steps, and $poly(\log n)$ time.*

The "convex hull approach" of this paper is of interest from the point of view of teaching as well. First, consider the original – and most demanding – teaching model [11], in which a set of labelled examples uniquely identifying c has to be computed for a given concept c.

Theorem 4.9. *For any fixed $d \geq 1$,*

$$\text{TD}\left(\text{HALFSPACE}_n^d\right) = O\left((\log n)^{d-1}\right).$$

Moreover, given a description of length l of a halfspace $c \in \text{HALFSPACE}_n^d$ (as a list of coefficients of a defining hyperplane), a teaching set of size $O\left((\log n)^{d-1}\right)$ can be computed in $poly(l, \log n)$ time for the concept c.

Proof: As for a halfspace c over X_n^d the concept $X_n^d \setminus c$ (the complement of c) is also a halfspace, the vertices of the convex hull of c plus the vertices of the convex hull of $X_n^d \setminus c$ can be chosen as a teaching set for c. According to Corollary 4.3, this teaching set contains $O\left((\log n)^{d-1}\right)$ examples, and – given a description of c – it can be efficiently computed using the membership query algorithm for HALFSPACE_n^d from Theorem 4.7. ∎

Notice that as $\text{TD}(C) \leq \text{MEMB}(C)$ for any concept class C [11], Theorem 4.7 directly implies a bound on the teaching dimension of the class of halfspaces, but the above result is stronger.

One can check that $\text{TD}\left(2\text{-HALFSPACE}_n^2\right) = \Omega(n)$, hence intersections of halfspaces are not efficiently teachable in the sense of [11]. However, the situation is different in the more realistic Goldman-Mathias *T/L teaching model* [12]. In this model teacher-learner (T/L) pairs are considered. A T/L pair for class C works in the following way. Given a concept $c \in C$, the teacher produces a set $S(c)$ of examples labelled according to c. The learner is then required to extract c from any set of the form $S(c) \cup A$, where A contains additional examples labelled according to c (see [12] for details).

Theorem 4.10. *For any fixed $d \geq 1$, $\alpha \geq 0$, and any function $k(n)$ such that $k(n) = O((\log n)^{\alpha})$, the class $k(n)\text{-HALFSPACE}_n^d$ is polynomially T/L-teachable using $O\left((\log n)^{d(\alpha+1)-1}\right)$ examples.*

Proof: Given a concept $c \in k(n)$-HALFSPACE$_n^d$ the teacher finds the vertices of CH(c) using the *Learn-Polytope* algorithm and an integer linear programming approach similar to that in Lemma 3.4. The teacher then outputs the vertices of CH(c) labelled as positive examples. The learner, given a set S of labelled examples, computes the convex hull of all positive examples in the sample, and outputs this convex hull as its final hypothesis. Both the teacher and the learner can be efficiently implemented. ∎

5 Related results

In this section we give some results on the complexity of learning monotone halfspaces. A halfspace $c \in$ HALFSPACE$_n^d$ is *monotone* iff there exist non-negative weights $w_1, \ldots, w_d \in \Re$ and a non-negative threshold $t \in \Re$ realizing c. Denote MON-HALFSPACE$_n^d$ the class of monotone halfspaces over X_n^d. Our motivation for studying monotone halfspaces is the fact that monotonicity is a structural property that often makes learning easier (monotone DNF vs. general DNF, monotone read-once formulas vs. general read-once formulas, *etc.*). We are interested in the question whether this holds true for halfspaces as well.

Theorem 5.1.

(i) $\mathrm{LC}\left(\text{MON-HALFSPACE}_n^d\right) = O\left(d^2(\log d + \log n)\right)$, *and the upper bound is achieved by a learning algorithm with poly$(d, \log n)$ running time.*

(ii) $\text{LC-ARB}\left(\text{MON-HALFSPACE}_n^d\right) = \Omega\left(d^2 \log n\right)$

(iii) $\text{LC-ARB-MEMB}\left(\text{MON-HALFSPACE}_n^d\right) = \Omega\left(\frac{d^2 \log n}{\log d + \log\log n}\right)$

(iv) $\mathrm{TD}\left(\text{MON-HALFSPACE}_2^d\right) = \mathrm{MEMB}\left(\text{MON-HALFSPACE}_2^d\right) = \binom{d+1}{\lfloor \frac{d+1}{2} \rfloor}$

Proof: (i) The Maass-Turán algorithm [18, 21] for general halfspaces can be modified to use only monotone halfspaces as hypotheses if the target concept is a monotone halfspace.
(ii) The general lower bound proof in [18, 21] can be modified to cover the monotone case.
(iii) This follows from (ii) and a general relationship between the two learning models proved in [19, 22].
(iv) The lower bound on the teaching dimension is implicitly contained in [2] and partially in [30], and the upper bound on the number of membership queries follows from an even stronger result of Hansel [13]. ∎

Observe that the bounds (i)-(iii) are the same as those in [18, 19, 20, 21, 22] for general halfspaces, while the teaching dimension of the class HALFSPACE$_2^d$ is known to be 2^d [2], as opposed to the bounds in (iv).

6 Discussion and open problems

We have designed efficient algorithms for exact identification of geometrical objects over a fixed-dimensional grid, based on the general paradigm "identify the target concept by constructing its convex hull". This paradigm may become important in connection with neural networks with a constant number of discrete inputs.

All our positive learnability results are based on the general *LearnPolytope* algorithm that constructs the convex hull of the target concept by searching for its vertices one by one. At least in special cases, there may exist more efficient constructions of the target convex hull, using a more sophisticated approach and/or other types of queries than those considered here. In fact, some previous positive results – the Maass-Turán algorithm [18, 21] learning HALFSPACE$_n^d$ for fixed $d \geq 1$ in *poly*$(\log n)$ time making $O(\log n)$ proper equivalence queries, the Bultman-Maass algorithm [8] learning HALFSPACE$_n^2$ in *poly*$(\log n)$ time making $O(\log n)$ membership queries, and the Bultman-Maass algorithm [8] learning GP-BOX$_n^2$ (restricted to rectangles with minimal edge length at least 1) in *poly*$(\log n)$ time making $O(\log n)$ membership and proper equivalence queries – can be understood as such efficient constructions, as all these algorithms can be modified to construct the convex hull of the target concept without affecting the known bounds on their performance.

The "simplicity" of the convex hull of a concept formed by intersections of halfspaces relativizes previous negative results on learning the classes 2-HALFSPACE$_n^2$ and GP-BOX$_n^2$ [19, 20, 21] giving an $\Omega(n)$ lower bound on the number of *proper* equivalence queries needed to learn any of them. Our incremental approach could be perhaps useful in showing that the same classes are efficiently learnable using extended, *i.e.*, non-proper, equivalence queries.

As the query complexity of our algorithms relies on the known bounds on the number of vertices of certain polytopes, it is of interest how tight these bounds are. It is known that the $O\left((\log n)^{d-1}\right)$ bound is matched for k-HALFSPACE$_n^d$ for sufficiently large constant k. More precisely, the general construction of Bárány *et al.* [3] implies that for any fixed $d \geq 1$ there is a concept $c \in 2d^2$-HALFSPACE$_n^d$ such that CH(c) has $\Omega\left((\log n)^{d-1}\right)$ vertices. For $k < 2d^2$ the situation is not clear. The construction of Rubin [25] gives a concept $c \in$ HALFSPACE$_n^2$ such that CH(c) has $\Omega(\log n)$ vertices (together with Corollary 4.3 we have a matching $\Theta(\log n)$ bound for HALFSPACE$_n^2$), and the construction of Morgan [24] provides $c \in 2$-HALFSPACE$_n^3$ such that CH(c) has $\Omega\left((\log n)^2\right)$ vertices (again this gives a matching $\Theta\left((\log n)^2\right)$ bound for 2-HALFSPACE$_n^3$). It is not even known whether the $O\left((\log n)^{d-1}\right)$ bound is tight for the class of single halfspaces for $d \geq 3$. A further thing is that we actually need bounds on the number of *facets* of the target polytope. Can one prove stronger bounds on the number of facets than those implied by the Upper Bound Theorem? It would be also interesting to give lower bounds on the complexity of learning the considered classes in the considered models. The best known lower bounds are usually $\Omega(\log n)$, implied by the "chain" argument from [19, 22]. Can the mentioned lower bounds on the number of vertices of some polytopes be used in this context?

Finally, it should be noted that in our fixed-dimensional geometrical setting PAC learning seems to be much easier than exact learning. Say, the class of convex concepts over X_n^2 represented as intersections of halfspaces is PAC learnable in polynomial time [5], and for any fixed $d, k \geq 1$ the class k-HALFSPACE$_n^d$ is properly PAC learnable in polynomial time [17].

References

[1] D. Angluin, "Queries and Concept Learning", *Machine Learning* **2** (1988) 319–342.

[2] M. Anthony, G. Brightwell, D. Cohen and J. Shawe-Taylor, "On Exact Specification by Examples", in: *Proceedings of the 5th Annual ACM Workshop on Computational Learning Theory (COLT'92)*, ACM Press, New York, NY, 1992, pp. 311–318.

[3] I. Bárány, R. Howe and L. Lovász, "On Integer Points in Polyhedra: a Lower Bound", *Combinatorica* **12**(2) (1992) 135–142.

[4] A. L. Blum and R. L. Rivest, "Training a 3-Node Neural Network is NP-Complete", *Neural Networks* **5** (1992) 117–127.

[5] A. Blumer, A. Ehrenfeucht, D. Haussler and M. Warmuth, "Learnability and the Vapnik-Chervonenkis Dimension", *Journal of the ACM* **36**(4) (1989) 929–965.

[6] N. H. Bshouty and R. Cleve, "On the Exact Learning of Formulas in Parallel", in: *Proceedings of the 33rd Annual Symposium on Foundations of Computer Science (FOCS'92)*, IEEE Computer Society Press, Los Alamitos, CA, 1992, pp. 513–522.

[7] M. Budinich and E. Milotti, "Feed-Forward Neural Networks: a Geometrical Perspective", *Journal of Physics A* **24** (1991) 881–888.

[8] W. J. Bultman and W. Maass, "Fast Identification of Geometric Objects with Membership Queries", in: *Proceedings of the 4th Annual Workshop on Computational Learning Theory (COLT'91)*, Morgan Kaufmann, San Mateo, CA, 1991, pp. 337–353.

[9] Z. Chen and W. Maass, "On-Line Learning of Rectangles", in: *Proceedings of the 5th Annual ACM Workshop on Computational Learning Theory (COLT'92)*, ACM Press, New York, NY, 1992, pp. 16–28.

[10] W. Cook, M. Hartmann, R. Kannan and C. McDiarmid, "On Integer Points in Polyhedra", *Combinatorica* **12**(1) (1992) 27–37.

[11] S. A. Goldman and M. J. Kearns, "On the Complexity of Teaching", in: *Proceedings of the 4th Annual Workshop on Computational Learning Theory (COLT'91)*, Morgan Kaufmann, San Mateo, CA, 1991, pp. 303–314.

[12] S. A. Goldman and H. D. Mathias, "Teaching a Smarter Learner", in: *Proceedings of the 6th Annual ACM Conference on Computational Learning Theory (COLT'93)*, ACM Press, New York, NY, 1993, pp. 67–76.

[13] G. Hansel, "Sur le Nombre des Fonctions Booléennes Monotones de n Variables", *Comptes Rendus de l'Académie des Sciences Paris* **262**(20) (1966) 1088-1090.

[14] A. C. Hayes and D. G. Larman, "The Vertices of the Knapsack Polytope", *Discrete Applied Mathematics* **6**(1983) 135–138.

[15] T. Hegedűs, "On Training Simple Neural Networks and Small-Weight Neurons", in: *Proceedings of the 1st European Conference on Computational Learning Theory (EuroCOLT'93)*, Royal Holloway, University of London, December 1993, Oxford University Press, to appear.

[16] H. W. Lenstra, Jr., "Integer Programming with a Fixed Number of Variables", *Mathematics of Operations Research* **8** (1983) 538–548.

[17] W. Maass, "On the Complexity of Learning on Feed-Forward Neural Nets", Lecture notes for a short course at the *Advanced School on Computational Learning and Cryptography* of the EATCS, Vietri sul Mare, Italy, September 1993.

[18] W. Maass and Gy. Turán, "On the Complexity of Learning from Counterexamples", in: *Proceedings of the 30th Annual Symposium on Foundations of Computer Science (FOCS'89)*, IEEE Computer Society Press, Los Angeles, CA, 1989, pp. 262–267.

[19] W. Maass and Gy. Turán, "On the Complexity of Learning from Counterexamples and Membership Queries", in: *Proceedings of the 31st Annual Symposium on Foundations of Computer Science (FOCS'90)*, IEEE Computer Society Press, Washington, DC, 1990, pp. 203–210.

[20] W. Maass and Gy. Turán, "Algorithms and Lower Bounds for On-Line Learning of Geometrical Concepts", Report 316, IIG-Report Series, Graz University of Technology, 1991.

[21] W. Maass and Gy. Turán, "How Fast can a Threshold Gate Learn?", Report 321, IIG-Report Series, Graz University of Technology, 1991.

[22] W. Maass and Gy. Turán, "Lower Bound Methods and Separation Results for On-Line Learning Models", *Machine Learning* **9** (1992) 107–145.

[23] P. McMullen and G. C. Shephard, *Convex Polytopes and the Upper Bound Conjecture*, Cambridge University Press, Cambridge, 1971.

[24] D. A. Morgan, "Upper and Lower Bound Results on the Convex Hull of Integer Points in Polyhedra", *Mathematika* **38** (1991) 321–328.

[25] D. S. Rubin, "On the Unlimited Number of Faces in Integer Hulls of Linear Programs with a Single Constraint", *Operations Research* **18** (1970) 940–946.

[26] A. Schrijver, *Theory of Linear and Integer Programming*, Wiley, Chichester, 1986.

[27] V. N. Shevchenko, "On the Number of Extremal Points in Integer Programming", *Kibernetika* (2) (1981) 133–134 (in Russian).

[28] V. N. Shevchenko, "An Algebraic Approach to Integer Programming", *Kibernetika* (4) (1984) 36–41 (in Russian), English translation: *Cybernetics* **20**(4) (1984) 508–515.

[29] V. N. Shevchenko, "On Some Functions of Many-Valued Logic Connected with Integer Programming", *Metody Diskretnogo Analiza* **42** (1985) 99–108 (in Russian).

[30] V. N. Shevchenko, "On Deciphering a Threshold Function of Many-Valued Logic", in: *Combinatorial-Algebraic Methods and their Applications*, Gorkii State University, 1987, pp. 155–163 (in Russian).