Neural Nets with Superlinear VC-Dimension

Wolfgang Maass

Institute for Theoretical Computer Science
Technische Universitaet Graz
Klosterwiesgasse 32/2
A-8010 Graz, Austria
e-mail: maass@igi.tu-graz.ac.at

Abstract

It has been known for quite a while that the Vapnik-Chervonenkis dimension (VC-dimension) of a feedforward neural net with linear threshold gates is at most $O(w \cdot \log w)$, where w is the total number of weights in the neural net. We show in this paper that this bound is in fact asymptotically optimal. More precisely, we exhibit for any depth $d \geq 3$ a large class of feedforward neural nets of depth d with w weights that have VC-dimension $\Omega(w \cdot \log w)$. This lower bound holds even if the inputs are restricted to boolean values. The proof of this result relies on a new method that allows us to encode more "program-bits" in the weights of a neural net than previously thought possible.

The Vapnik-Chervonenkis dimension VC-dimension($\mathcal{N}$) of a neural net $\mathcal{N}$ with n input nodes is defined as the size of the largest set $S \subseteq \mathbf{R}^n$ which is "shattered" by $\mathcal{N}$ in the sense that every function $F : S \rightarrow \{0,1\}$ can be computed by $\mathcal{N}$ with some assignment of real numbers to its weights.

The VC-dimension of a neural net $\mathcal{N}$ is an important measure for the expressiveness for $\mathcal{N}$, i.e. for the variety of functions that can be computed by $\mathcal{N}$ with different choices for its weights. In particular it has been shown in [BEHW] and [EHKV] that the VC-dimension of $\mathcal{N}$ essentially determines the number of training examples that are needed to train $\mathcal{N}$ in Valiant's model [V] for probably approximately correct learning ("PAC-learning").

It has been known for quite a while that the VC-dimension of a neural net with linear threshold gates and w edges (respectively w weights) is at most $O(w \cdot \log w)$. This result, which holds for arbitrary real valued input patterns, was first shown by Cover in [C 64] (see also [C 68]), and later by Baum and Haussler [BH]. It has frequently been conjectured that the "true" upper bound is $O(w)$. This conjecture is quite plausible, since a single linear threshold gate with w edges has VC-dimension $w + 1$. Furthermore it is hard to imagine that the VC-dimension of a network of linear threshold gates can be larger than the sum of the VC-dimensions of the individual linear threshold gates in the network.

We disprove this popular conjecture by showing that for any depth $d \geq 3$ quite a number of neural nets $\mathcal{N}$ of depth d have a VC-dimension that is superlinear in the number w of edges in $\mathcal{N}$. In particular, we exhibit for arbitrarily large $w \in \mathbf{N}$ neural nets $\mathcal{N}$ of depth 3 (i.e. with 2 hidden layers) with w weights that have VC-dimension $\Omega(w \cdot \log w)$. This shows that the quoted upper bound of $O(w \log w)$ is in fact asymptotically optimal.

It is of some interest to note that the upper bound $O(w \cdot \log w)$ for the VC-dimension of a neural net with w weights holds even for the case of *real valued* inputs, whereas our matching lower bound $\Omega(w \cdot \log w)$ for the VC-dimension of certain neural nets $\mathcal{N}_w$ with w weights holds already for the restriction of $\mathcal{N}_w$ to *boolean* inputs. Our lower bound also shows that the well-known upper bound $2w \log(eN)$ for the VC-dimension of a neural net with w weights and N computation nodes (due to Baum and Haussler [BH]) is asymptotically optimal.

The result of this paper may also be viewed as mathematical evidence for a certain type of "connectionism thesis": that a network of neuron-like elements is more than just the sum of its elements. We show that in a large neural net a single edge may add more than a constant to the VC-dimension of the neural net: its contribution may increase with the logarithm of the total size of the neural net.

Although we consider in this paper only neural nets with linear threshold gates, it is obvious that the same lower bound can also be derived for neural nets with other activation functions such as $\sigma(y) = \frac{1}{1+e^{-y}}$ (see [RM]) or piecewise linear (resp. polynomial) functions of a similar type (see [S], [MSS], [M 93]).

This paper improves our earlier results from [M 92] (see [M 93] for an extended abstract), where we had exhibited neural nets of depth 4 with superlinear VC-dimension. Both our preceding results and the proof of our new result employ classical circuit construction methods due to Neciporuk [N] and Lupanov [L]. Bartlett [B] has independently derived lower bounds for the VC-dimensions of various neural nets of depth 2 and 3 that are *linear* in the number w of weights.

The *neural nets* which are considered in this paper are feedforward neural nets with linear threshold gates (or simpler: threshold gates), i.e. gates which apply the heaviside activation function to the weighted sum $\sum_{i=1}^{m} \alpha_i y_i + \alpha_0$ of their inputs $y_1, \ldots, y_m$. The parameters $\alpha_1, \ldots, \alpha_m$ and α_0 are the *weights* of such gate. We will consider in this paper only neural nets with boolean inputs and one boolean output.

The *depth* of a neural net is the length of the longest path from an input node to the output node (= output gate). The depth of a gate in a neural net is the length of the longest path from an input node to that gate. We refer to all gates of depth d as "level d" of a neural net.

We will focus our attention on neural nets that are *layered* in the sense that for each gate all paths from an input node to that gate have the same length. This means that only gates on successive levels are connected by an edge, and input nodes are connected by an edge only with gates on level 1. This is not a serious restriction, since for $i+1 < j$ one may replace an edge between nodes on levels i and j by a path of length $j-i$ (by introducing $j-i-1$ "dummy" gates on the intermediate levels). It is obvious that a layered neural net of depth d has exactly $d-1$ *hidden layers*.

One calls a layered neural net *fully connected* if any two nodes on successive levels (including input nodes and gates on level 1) are connected by an edge.

We use the standard notation $f = \Theta(g)$ for arbitrary functions $f, g : \mathbf{N} \to \mathbf{N}$ to indicate that both $f = O(g)$ and $f = \Omega(g)$.

Theorem 1: *Assume that $(\mathcal{N}_n)_{n \in \boldsymbol{N}}$ is any sequence of fully connected layered neural nets of depth $d \geq 3$. Furthermore assume that $\mathcal{N}_n$ has n input nodes and $\Theta(n)$ gates, of which $\Omega(n)$ gates are on the first hidden layer, and at least $4 \log n$ gates are on the second hidden layer of $\mathcal{N}_n$.*

Then $\mathcal{N}_n$ has $\Theta(n^2)$ edges and VC-dimension$(\mathcal{N}_n) = \Theta(n^2 \log n)$.

The *proof* of Theorem 1 proceeds by "embedding" into the given neural nets $\mathcal{N}_n$ of Theorem 1 the special neural nets $\mathcal{M}_{\tilde{n}}$ that are constructed in the following Theorem 2. The precise derivation of Theorem 1 from the construction of Theorem 2 is given after the proof of Theorem 2.

Theorem 2: *Assume that n is some arbitrary power of 2. Then one can con-*

struct a neural net $\mathcal{M}_n$ of depth 3 with n input nodes and at most $17n^2$ edges such that VC-dimension($\mathcal{M}_n$) $\geq n^2 \cdot \log n$.

Proof of Theorem 2: One may view the weights of a neural net $\mathcal{M}_n$ collectively as the "program" of $\mathcal{M}_n$. It is obviously no problem to employ large weights in a neural net. But the question is how many bits of the weights are actual "program-bits" in the sense that they contribute to the VC-dimension of the neural net. We will exhibit in this proof a method for "efficient programming" of a neural net that allows us to encode an unusually large number of "program-bits" in the weights. More precisely we show that on average each of the $\Theta(n^2)$ weights can be used to store $\Omega(\log n)$ "program-bits". Furthermore our construction requires only integer weights whose absolute value is polynomial in n. Hence all weights have bit-length $O(\log n)$, and our construction shows that a constant fraction of all weight-bits can be used to store the "program" of $\mathcal{M}_n$.

Before we describe the precise construction of the desired neural net $\mathcal{M}_n$, we first illustrate this method for "efficient programming" of a neural net in a simpler example. This "example" will turn out to be an essential building block for the construction of $\mathcal{M}_n$. We construct a neural net $\mathcal{M}$ with n^2 integer weights of size polynomial in n that can be programmed to compute any n-tuple of permutations of $\{\underline{e}_1, \ldots, \underline{e}_n\}$ (where $\underline{e}_i \in \{0,1\}^n$ is the i-th unit vector). Since $(n!)^n = 2^{\Theta(n^2 \log n)}$ different n-tuples of permutations of $\{\underline{e}_1, \ldots, \underline{e}_n\}$ exist, this construction of $\mathcal{M}$ may be viewed as an example for "efficient programming" of a neural net (in the sense that on average $\Omega(\log n)$ bits of each weight are relevant for determining the function that is computed by $\mathcal{M}$).

The neural net $\mathcal{M}$ will have $2 \cdot n$ inputs and n outputs. For the construction of $\mathcal{M}$ we assume that h is an arbitrary given function from $\{\underline{e}_1, \ldots, \underline{e}_n\}^2$ into $\{\underline{e}_1, \ldots, \underline{e}_n\}$ such that for any $j \in \{1, \ldots, n\}$ the function $h\left(\cdot, \underline{e}_j\right)$ is a permutation of $\{\underline{e}_1, \ldots, \underline{e}_n\}$. Obviously any n-tuple of permutations of $\{\underline{e}_1, \ldots, \underline{e}_n\}$ can be represented by such a function h. We define for $i, j \in \{1, \ldots, n\}$ weights $w_{i,j}$ by

$$w_{i,j} = r \Leftrightarrow h\left(\underline{e}_r, \underline{e}_j\right) = \underline{e}_i.$$

This implies that for any $p, q \in \{1, \ldots, n\}$ one has $h\left(\underline{e}_p, \underline{e}_q\right) = \underline{e}_i \Leftrightarrow w_{i,q} = p \Leftrightarrow \sum_{j=1}^{n} w_{i,j} \cdot \left(\underline{e}_q\right)_j = p$ (we write $(\underline{x})_j$ for the j-th coordinate of a vector $\underline{x}$). Hence one may take as the i-th output gate of $\mathcal{M}$ a gate that checks (for example via the AND of two threshold gates) whether $\sum_{j=1}^{n} w_{i,j} \cdot \left(\underline{e}_q\right)_j = \sum_{r=1}^{n} r \cdot \left(\underline{e}_p\right)_r$ (i.e. $w_{i,q} = p$). It is easy to verify that this neural net $\mathcal{M}$ computes the given function h.

The above method for "efficient programming" of a neural net $\mathcal{M}$ cannot be used directly in order to show that a neural net has a large VC-dimension, because for that we have to be able to compute many $0-1$ valued rather than vector-valued functions F on a neural net. However the subsequent construction shows that any

$0-1$ valued function F (with a suitable domain S) can be encoded by 4 functions $g_1, \ldots, g_4$ which are each n-tuples of permutations of $\{\underline{e}_1, \ldots, \underline{e}_n\}$. Hence we can employ in the construction of $\mathcal{M}_n$ the preceding method for "efficient programming" of n-tuples of permutations in a neural net. The only difference is that in $\mathcal{M}_n$ the output of each function g_k will be considered in *binary* representation (i.e. g_k outputs $\mathrm{bin}(i) \in \{0,1\}^{\log n}$ instead of $\underline{e}_i \in \{0,1\}^n$, where $\mathrm{bin}(i)$ is the binary string that represents the natural number $i-1$). This will be necessary since each bit of $\mathrm{bin}(i)$ will be used to encode the value of F for a different input.

We will now describe the precise construction of a neural net $\mathcal{M}_n$ with $O(n^2)$ edges and VC-dimension$(\mathcal{M}_n) \geq n^2 \cdot \log n$.

We assume that n is of the form $2^{n'}$ for some non-zero $n' \in \mathbf{N}$. This implies that n is even and that $\log n \in \mathbf{N}$. We construct a neural net $\mathcal{M}_n$ with $2n + \log n$ binary inputs and $O(n^2)$ weights that shatters the following set $S \subseteq \{0,1\}^{2n+\log n}$ of size $n^2 \cdot \log n$:

$$S := \left\{\underline{e}_p \underline{e}_q \underline{\tilde{e}}_m : p, q \in \{1, \ldots, n\} \text{ and } m \in \{1, \ldots, \log n\}\right\},$$

where $\underline{e}_r \in \{0,1\}^n$ denotes the r-th unit vector of length n $(r = 1, \ldots, n)$, and $\underline{\tilde{e}}_m \in \{0,1\}^{\log n}$ denotes the m-th unit vector of length $\log n$ $(m = 1, \ldots, \log n)$. Thus each $\underline{u} \in S$ contains exactly three 1's.

Fix any map $F : S \to \{0,1\}$. One can encode F by a function $g : \{\underline{e}_1, \ldots, \underline{e}_n\}^2 \to \{0,1\}^{\log n}$ where the m-th output bit of $g\left(\underline{e}_p, \underline{e}_q\right)$ equals 1 if and only if $F\left(\underline{e}_p \underline{e}_q \underline{\tilde{e}}_m\right) = 1$. It is straightforward to show (as follows) that for any function $g : \{\underline{e}_1, \ldots, \underline{e}_n\}^2 \to \{0,1\}^{\log n}$ there exist for $k = 1, \ldots, 4$ functions $g_k : \{\underline{e}_1, \ldots, \underline{e}_n\}^2 \to \{0,1\}^{\log n}$ such that $g_k\left(\cdot, \underline{e}_q\right)$ is $1-1$ for every fixed $q \in \{1, \ldots, n\}$, and such that for all $p, q \in \{1, \ldots, n\}$:

$$g\left(\underline{e}_p, \underline{e}_q\right) = \begin{cases} g_1\left(\underline{e}_p, \underline{e}_q\right) \oplus g_2\left(\underline{e}_p, \underline{e}_q\right), & \text{if } p \leq \frac{n}{2} \\ g_3\left(\underline{e}_p, \underline{e}_q\right) \oplus g_4\left(\underline{e}_p, \underline{e}_q\right), & \text{if } p > \frac{n}{2}. \end{cases}$$

The symbol $\oplus$ denotes here the bit-wise exclusive OR (i.e. parity) on bit-strings of length $\log n$.

To justify this claim, one chooses for each fixed $\underline{e}_q$ simultaneously for $k = 1, 2$ values for $g_k\left(\underline{e}_1, \underline{e}_q\right)$, $g_k\left(\underline{e}_2, \underline{e}_q\right)$, ... in such a way that earlier assigned values for $g_k\left(\cdot, \underline{e}_q\right)$ are avoided. After $g_1(\underline{e}_p, \underline{e}_q)$ and $g_2(\underline{e}_p, \underline{e}_q)$ have been defined for $p = 1, \ldots, l < \frac{n}{2}$, one can choose for $g_1(\underline{e}_{l+1}, \underline{e}_q)$ any string in $\{0,1\}^{\log n}$ that is not in the set

$$\{g_1(\underline{e}_p, \underline{e}_q) : p = 1, \ldots, l\} \cup \{g(\underline{e}_{l+1}, \underline{e}_q) \oplus g_2(\underline{e}_p, \underline{e}_q) : p = 1, \ldots, l\}.$$

Then one sets $g_2(\underline{e}_{l+1}, \underline{e}_q) := g(\underline{e}_{l+1}, \underline{e}_q) \oplus g_1(\underline{e}_{l+1}, \underline{e}_q)$. One continues the definition of $g_1(\underline{e}_p, \underline{e}_q)$ and $g_2(\underline{e}_p, \underline{e}_q)$ for $p > \frac{n}{2}$ in an arbitrary manner so that $g_1(\cdot, \underline{e}_q)$ and $g_2(\cdot, \underline{e}_q)$ become $1-1$. The definition of $g_3(\cdot, \underline{e}_q)$ and $g_4(\cdot, \underline{e}_q)$ is analogous.

The neural net $\mathcal{M}_n$ computes F in the following way. The output gate on level 3 is an OR of $4\log n$ threshold gates. These threshold gates consist of $\log n$ blocks of 4 threshold gates, such that for any $b \in \{1, \ldots, \log n\}$ some threshold gate in the b-th block outputs 1 for network input $\underline{e}_p \underline{e}_q \tilde{\underline{e}}_m$ if and only if $m = b$ and $\left(g\left(\underline{e}_p, \underline{e}_q\right)\right)_b = 1$ (i.e. $F\left(\underline{e}_p \underline{e}_q \tilde{\underline{e}}_m\right) = 1$). More precisely, the a-th threshold gate in block b outputs 1 if and only if the a-th one of the following 4 conditions is satisfied:

$$
\begin{array}{llllllllll}
(1) & m = b & \wedge & p \leq \frac{n}{2} & \wedge & \left(g_1\left(\underline{e}_p, \underline{e}_q\right)\right)_b = 1 & \wedge & \left(g_2\left(\underline{e}_p, \underline{e}_q\right)\right)_b = 0 \\
(2) & m = b & \wedge & p \leq \frac{n}{2} & \wedge & \left(g_1\left(\underline{e}_p, \underline{e}_q\right)\right)_b = 0 & \wedge & \left(g_2\left(\underline{e}_p, \underline{e}_q\right)\right)_b = 1 \\
(3) & m = b & \wedge & p > \frac{n}{2} & \wedge & \left(g_3\left(\underline{e}_p, \underline{e}_q\right)\right)_b = 1 & \wedge & \left(g_4\left(\underline{e}_p, \underline{e}_q\right)\right)_b = 0 \\
(4) & m = b & \wedge & p > \frac{n}{2} & \wedge & \left(g_3\left(\underline{e}_p, \underline{e}_q\right)\right)_b = 0 & \wedge & \left(g_4\left(\underline{e}_p, \underline{e}_q\right)\right)_b = 1.
\end{array}
$$

The subcondition "$m = b$" is satisfied if and only if $\left(\underline{e}_p \underline{e}_q \tilde{\underline{e}}_m\right)_{2n+b} = 1$. The subcondition "$p \leq \frac{n}{2}$" is satisfied if and only if $\sum_{r=1}^{n} r \cdot \left(\underline{e}_p\right)_r \leq \frac{n}{2}$, hence it can be tested by a threshold gate on level 1 (analogously for "$p > \frac{n}{2}$"). The remaining subconditions are tested with the help of $8n$ threshold gates on level 1 that involve weights $w_{k,i,j} \in \{1, \ldots, n\}$, which are defined by the condition

$$w_{k,i,j} = r \Leftrightarrow g_k\left(\underline{e}_r, \underline{e}_j\right) = \mathrm{bin}(i).$$

Among these there are $4n$ threshold gates $G^+_{k,i}\left(\underline{e}_p, \underline{e}_q\right)$ on level 1 that output 1 if and only if $\sum_{r=1}^{n} r \cdot \left(\underline{e}_p\right)_r \geq \sum_{j=1}^{n} w_{k,i,j} \cdot \left(\underline{e}_q\right)_j$ (i.e. $p \geq w_{k,i,q}$), and $4n$ threshold gates $G^-_{k,i}\left(\underline{e}_p, \underline{e}_q\right)$ on level 1 that output 1 if and only if $\sum_{r=1}^{n} r \cdot \left(\underline{e}_p\right)_r \leq \sum_{j=1}^{n} w_{k,i,j} \cdot \left(\underline{e}_q\right)_j$ (i.e. $p \leq w_{k,i,q}$); for $k = 1, \ldots, 4$ and $i = 1, \ldots, n$. These are the only weights in the neural net $\mathcal{M}_n$ which depend on the function $F : S \to \{0, 1\}$. By definition one has that for each k, i at least one of the two gates $G^+_{k,i}\left(\underline{e}_p, \underline{e}_q\right)$, $G^-_{k,i}\left(\underline{e}_p, \underline{e}_q\right)$ outputs 1. Furthermore for any $k \in \{1, \ldots, 4\}$ and any $\underline{e}_p \underline{e}_q \tilde{\underline{e}}_m \in S$ there is exactly one $i \in \{1, \ldots, n\}$ such that both of these gates output 1. This index i is characterized by the equality $g_k\left(\underline{e}_p, \underline{e}_q\right) = \mathrm{bin}(i)$. Hence one can check whether $\left(g_k\left(\underline{e}_p, \underline{e}_q\right)\right)_b = 1$ by testing whether

$$\sum_{\substack{i=1,\ldots,n \\ \text{with } (\mathrm{bin}(i))_b = 1}} G^+_{k,i}\left(\underline{e}_p, \underline{e}_q\right) + G^-_{k,i}\left(\underline{e}_p, \underline{e}_q\right) \geq \frac{n}{2} + 1,$$

and one can check whether $\left(g_k\left(\underline{e}_p, \underline{e}_q\right)\right)_b = 0$ by testing whether

$$\sum_{\substack{i=1,\ldots,n \\ \text{with } (\mathrm{bin}(i))_b = 0}} G^+_{k,i}\left(\underline{e}_p, \underline{e}_q\right) + G^-_{k,i}\left(\underline{e}_p, \underline{e}_q\right) \geq \frac{n}{2} + 1.$$

Furthermore the sums on the left hand side of both inequalities can only assume the values $\frac{n}{2}$ or $\frac{n}{2}+1$. Therefore one can test the AND of two subconditions of this type and of the subconditions " $m = b$ " and " $p \leq \frac{n}{2}$ " (" $p > \frac{n}{2}$ ") by a *single* threshold gate on level 2 of $\mathcal{M}_n$. Hence one can test each of the conditions $(1), \ldots, (4)$ by a separate threshold gate in the b-th block on level 2.

Altogether the network $\mathcal{M}_n$ has $2n + \log n$ boolean inputs and $8n+2$ threshold gates on level 1 (the first hidden layer), which are connected by $16n^2 + 2n$ edges to level 0 (the input level). Among these are the $8n^2$ edges with the weights $w_{k,i,j}$ that depend on the function $F : S \to \{0,1\}$ (all other weights and thresholds in $\mathcal{M}_n$ are independent of F).

On level 2 (the second hidden layer) the network $\mathcal{M}_n$ has $4 \log n$ threshold gates that test the conditions $(1), \ldots, (4)$ as described before. Each of these gates is connected by $1 + 2n$ edges to gates on level 1, and by one edge to some input node. On level 3 the network $\mathcal{M}_n$ has an OR that is connected by $4 \log n$ edges to the gates on level 2.

In order to change $\mathcal{M}_n$ into a *layered* neural net we add $4 \log n$ threshold gates on level 1 and replace the former $4 \log n$ edges between input nodes and gates on level 2 by pathes of length 2 (with the new $4 \log n$ threshold gates on level 1 as intermediate nodes). Thus as a *layered neural net* $\mathcal{M}_n$ has $8n + 4\log n + 2$ gates on level 1 and $4 \log n$ gates on level 2, $16n^2 + 2n + 4\log n$ edges between input nodes and gates on level 1, and $(2n+2) \cdot 4 \log n$ edges between gates on levels 1 and 2.

Altogether the constructed layered neural net $\mathcal{M}_n$ consists of $2n + \log n$ input nodes, $8n + 8\log n + 3$ computation nodes, and $16n^2 + (8 \log n + 2)n + 16 \log n$ edges. Obviously the nodes and edges of $\mathcal{M}_n$ are independent of the given function $F : S \to \{0,1\}$, whereas the weights on $8n^2$ of the edges depend on F (these weights range over $\{1, \ldots, n\}$).

Since the function $F : S \to \{0,1\}$ that is computed by $\mathcal{M}_n$ was choosen arbitrarily, the construction implies that the set S is shattered by $\mathcal{M}_n$. Hence VC-dimension$(\mathcal{M}_n) \geq |S| = n^2 \cdot \log n$. ∎

Proof of Theorem 1: Since $\mathcal{N}_n$ has $O(n)$ gates, $\Omega(n)$ gates on level 1, and n input nodes, it is obvious that $\mathcal{N}_n$ has $\Theta(n^2)$ edges.

Let $K \geq 1$ be some natural number such that each of the given neural nets $\mathcal{N}_n$ from Theorem 1 has at least $\frac{n}{K}$ gates on level 1. For $n \geq 28K$ we choose $\tilde{n} \in \mathbf{N}$ maximal of the form $2^{n'}$ for some $n' \in \mathbf{N}$ such that

$$(*)\ \ 8\tilde{n} + 4 \log \tilde{n} + 2 \leq \frac{n}{K}\ .$$

It is obvious that the rational number $\frac{n}{14K}$ in place of $\tilde{n}$ satisfies $(*)$. Hence for $n \geq 28K$ we can find within the interval $\left[\frac{n}{28K}, \frac{n}{14K}\right]$ some power of 2 that satisfies

(*). This implies that the previously defined natural number $\tilde{n}$ satisfies $\tilde{n} \geq \frac{n}{28K}$. Furthermore we have $4 \log \tilde{n} \leq 4 \log n$, since $\tilde{n} \leq n$. Thus the given neural net $\mathcal{N}_n$ has at least $8\tilde{n} + 4 \log \tilde{n} + 2$ gates on level 1, at least $4 \log \tilde{n}$ gates on level 2, and at least 1 gate on each level l with $3 \leq l \leq d$. Furthermore $\mathcal{N}_n$ has $n \geq 2\tilde{n} + \log \tilde{n}$ input nodes.

Since $\mathcal{N}_n$ is by assumption fully connected, the preceding considerations imply that the graph of $\mathcal{M}_{\tilde{n}}$ is contained as a subgraph in the graph of $\mathcal{N}_n$ (where each node of $\mathcal{M}_{\tilde{n}}$ occurs in $\mathcal{N}_n$ on the same layer as in $\mathcal{M}_{\tilde{n}}$). Hence we can simulate $\mathcal{M}_{\tilde{n}}$ on $\mathcal{N}_n$ by assigning the value 0 to all superfluous input nodes of $\mathcal{N}_n$, and by assigning the weight 0 to all superfluous edges of $\mathcal{N}_n$. Furthermore we select in the case $d > 3$ one node on each of the levels $4, \ldots, d$ of $\mathcal{N}_n$ and set its weights so that it computes the identity function on the output from the selected gate on the preceding level.

Since $\mathcal{N}_n$ can simulate any computation of $\mathcal{M}_{\tilde{n}}$, it follows that VC-dimension($\mathcal{N}_n$) $\geq$ VC-dimension($\mathcal{M}_{\tilde{n}}$) $\geq$

$$\tilde{n}^2 \cdot \log \tilde{n} \geq (\frac{n}{28K})^2 \cdot \log \frac{n}{28K} = \Omega(n^2 \cdot \log n).$$

∎

Acknowledgement

We would like to thank György Turán for inspiring discussions and Eric Baum for helpful comments.

References

[B] P. L. Bartlett, "Lower bounds on the Vapnik-Chervonenkis dimension of multi-layer threshold networks", *Proc. of the 6th Annual ACM Conference on Computational Learning Theory*, 1993, 144 - 150

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

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

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

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

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

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

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

[M 92] W. Maass, "Bounds for the computational power and learning complexity of analog neural nets", *IIG-Report 349 of the Technische Universität Graz*, (October 1992).

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

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

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

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

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