Analog versus Discrete Neural Networks[1]

Bhaskar DasGupta	Georg Schnitger
Department of Computer Science	Fachbereich 20, Informatik
University of Waterloo	Universität Frankfurt
Waterloo, Ontario N2L 3G1	60054 Frankfurt
Canada	Germany
Email: *bdasgupt@daisy.uwaterloo.ca*	Email: *georg@thi.informatik.uni-frankfurt.de*

Abstract

We show that neural networks with three-times continuously differentiable activation functions are capable of computing a certain family of n-bit Boolean functions with two gates, whereas networks composed of binary threshold functions require at least $\Omega(\log n)$ gates.

Thus, for a large class of activation functions, analog neural networks can be more powerful than discrete neural networks, even when computing Boolean functions.

1 Introduction.

Artificial neural networks have become a popular model for machine learning and many results have been obtained regarding their application to practical problems. Typically, the network is trained to encode complex associations between inputs and outputs during supervised training cycles, where the associations are encoded by the weights of the network. Once trained, the network will compute an input/output mapping which (hopefully) is a good approximation of the original mapping.

[1] Partially supported by NSF Grant CCR-9114545

In this paper we are mostly interested in feedforward neural networks, i.e., neural networks whose underlying graph is acyclic. We concentrate on computing Boolean functions to allow a comparison of the computing power of analog and discrete neural networks. We start by formally introducing feedforward neural networks (with binary inputs and a single output neuron).

Definition 1.1 *Let $\gamma : \mathcal{R} \to \mathcal{R}$ be given.*

(a) *The architecture of a γ-net C is given by a directed graph G with a single sink (i.e. a single vertex with no outgoing edge). C is obtained, if we additionally specify a labeling of the edges and vertices of G by real numbers. The real number assigned to an edge (resp. vertex) is called its weight (resp. its threshold).*

(b) *C computes a function $f_C : \{0,1\}^n \to \mathcal{R}$ as follows. The components of the input vector $x = (x_1, \ldots, x_n)$ are assigned to the sources of G (i.e., to the vertices of G with no incoming edge).*

Let $v_1, \ldots, v_r$ be the immediate predecessors of vertex v. The input for v is then

$$s_v(x) = \sum_{i=1}^{r} w_i y_i - t_v,$$

where w_i is the weight of the edge (v_i, v), t_v is the threshold of v and y_i is the value assigned to v_i. If v is not the sink, then we assign the value $\gamma(s_v(x))$ to v. Otherwise, we assign $s_v(x)$ to v.

(b) *The size of C is the number of vertices of its architecture G (excluding the sources). The the depth of C is the number of edges on a longest directed path from the sources to the sink.*

Since the output of a γ-net is a real value, an appropriate convention has to be adopted when computing a Boolean function. We employ the same convention as in (Maass *et al.* 1991).

Definition 1.2 *Let ε be a positive real number and let C be a γ-net. Then C computes the Boolean function $F : \{0,1\}^n \to \{0,1\}$ with separation ϵ provided there exists a real number t_C, such that for all $(x_1, \ldots, x_n) \in \{0,1\}^n$*

$$F(x_1, \ldots, x_n) = 0 \quad \Leftrightarrow \quad f_C(x_1, \ldots x_n) \leq t_C - \epsilon$$
$$F(x_1, \ldots, x_n) = 1 \quad \Leftrightarrow \quad f_C(x_1, \ldots x_n) \geq t_C + \epsilon$$

γ-nets (for various activation functions γ and real-valued domain) have been investigated for their approximation power and other related complexity-theoretic properties in (DasGupta and Schnitger 1993; Höffgen 1993; Macintyre and Sontag 1993; Maass 1993; Zhang 1992).

In this paper, however, we consider the computation of Boolean functions only. Our goal is a comparison of the computational power of a large class of γ-nets and of *binary threshold networks*, i.e., $\mathcal{H}$-nets with the binary threshold function $\mathcal{H}$ defined by

$$\mathcal{H}(x) = \begin{cases} 0 & \text{if } x < 0 \\ 1 & \text{otherwise.} \end{cases}$$

(Since we will consider $\mathcal{H}$-nets for the computation of Boolean functions, the binary threshold function will also be used for the output gates.)

Threshold networks have been extensively studied in the literature. In (Reif 1987) it is shown, that binary threshold networks of bounded depth (and size polynomial in n) can compute the sum and the product of n n-bit numbers. In (Hajnal *et. al.* 1987) and (Goldmann and Hastad 1987) lower bounds on the size of depth-two (resp. depth-3) binary threshold networks are given. Thus binary threshold networks are known to be powerful and lower bounds (when computing Boolean functions) are correspondingly rather weak.

Moreover, the binary threshold function also plays an important role when considering the approximation power of neural networks with real inputs and outputs. In (DasGupta and Schnitger 1993), activations functions are considered which are capable of tightly approximating polynomials and the binary threshold function $\mathcal{H}$ with neural nets of bounded depth and small size. These activation functions have therefore at least the approximation power of spline-networks and thus have considerable approximation power. This fact is used in (DasGupta and Schnitger 1993) to show the "equivalence" of various activation functions including the standard sigmoid $\sigma(x) = \frac{1}{1+e^{-x}}$, rational functions and roots (which are not polynomials).

The following question, originally posed in (Sontag 1990), is the main topic of this paper:

- Does there exist a family of Boolean functions $f_n : \{0,1\}^n \to \{0,1\}$ which can be computed

by σ-nets with a constant number of gates, but which requires binary threshold networks with more than a constant number of gates?

Thus, our goal is a comparison of the computational power of analog and discrete neural networks with binary inputs.

If the number of inputs is counted when determining network size, then binary threshold networks are equivalent to σ-nets for the case of bounded depth and small weights even if we don't allow depth to increase (but allow a polynomial increase in size; see Maass *et. al.* 1991). The simulation results in (DasGupta and Schnitger 1993) imply that binary threshold networks and σ-nets are equivalent when computing Boolean functions even for unbounded depth and even with large weights *provided* depth of the simulating binary threshold network is allowed to increase by a constant factor and size is allowed to increase polynomially.

But, the above equivalence does not hold anymore if we do not count the number of inputs when determining network size and analog computation may indeed turn out to be more powerful. This was first demonstrated in (Maass *et al.* 1991), who construct a family f_n of Boolean functions which can be computed by a γ-nets (for a large class of functions γ) of constant size in depth two. It is then shown, that binary threshold networks of depth two require non-constant size. On the other hand, each function f_n can be computed by binary threshold networks in depth three and constant size.

This separation result is therefore depth-dependent. In this paper, we give a separation which holds for arbitrary depth. In particular, we consider the problem of "unary squaring", i.e., the family of languages SQ_n with

$$SQ_n = \{(x, y) : \ x \in \{0,1\}^n, y \in \{0,1\}^{n^2} \ \text{ and } \ [x]^2 \geq [y]\}.$$

($[z]$ denotes the bit sum of the binary string z; i.e. $[z] = \sum_i z_i$). We obtain the following result,

Theorem 1.1 *A binary threshold network accepting SQ_n must have size at least $\Omega(\log n)$. But SQ_n can be computed by a σ-net with two gates.*

In fact, we give a generalized upper bound in Theorem 2.1, where γ-nets with two gates are constructed for a large class of functions γ.

The lower bound of Theorem 1.1 is "almost" tight, since it is possible to design a binary threshold net of size $O(\log n \cdot \log\log n \cdot \log\log\log n)$ which accepts SQ_n.

The proof of 1.1 uses techniques of circuit theory. We refer the reader to (Wegener 1987; Boppana and Sipser 1990) for a detailed account of circuit theory and restrict ourselves to a few comments. A circuit corresponds (using the notation of this paper) to a Γ-net, where Γ is a class of Boolean functions and where functions in Γ are assigned to the vertices of the net-architecture. { AND, OR, NOT}-circuits are perhaps the most prominent circuit class.

One of the main tasks of circuit theory is to derive lower bounds for the size and/or depth of circuits computing specific Boolean functions. Little progress has been made in deriving lower bounds for { AND, OR, NOT}-circuits of bounded fan-in. (The fan-in of a circuit is the maximum, over all vertices, of the number of immediate predecessors of a vertex.) For instance, no specific function is known which requires super-linear size. The situation improves considerably if {AND, OR, NOT}-circuits of *unbounded* fan-in (and small depth) are considered: Razborov (Razborov 1987) gives exponential lower bounds for the size of circuits computing the parity of n bits in bounded depth. However, as already mentioned above, threshold-circuits (or threshold networks in our notation) of bounded depth have an impressive computing power and, perhaps not surprisingly, not even superlinear lower bounds on the size are known.

In (Wegener 1991) sublinear lower bounds on the size of threshold circuits are given. There the notion of *sensitivity* is introduced: a Boolean function f of n variables is called k-sensitive, if no setting of $n-k$ variables to arbitrary (zero or one) values transforms f into a constant function of the remaining k free variables. (For example, the parity function of n variables is k-sensitive for any k with $1 \leq k \leq n$).

We face the problem that SQ_n is *not* k-sensitive even for large values of k; for instance, if we set all x-bits to 1 and one y-bit to 0, then SQ_n becomes a constant function of the remaining free variables. Also, intermediate forms of sensitivity (i.e. k-sensitivity in which at least a constant

fraction of both the x-bits and the y-bits are set) have to be ruled out: setting half of the x-bits to 0 and setting half of the y-bits to 1 again reduces SQ_n to a constant function of the remaining free variables. Therefore, Wegener's lower bound for sensitive functions (Wegener 1991) does not apply. Our lower bound proof does proceed by trying to examine the given circuit gate by gate. But we were not successful in trivializing each gate (i.e., by setting input bits to appropriate zero/one values, we were unable to guarantee that a considered gate gate computes a constant function of the remaining free input bits). Instead we construct a *subdomain* of the input space which allows to trivialize threshold gates while not trivializing SQ_n.

The rest of the paper is organized as follows. In section 2 we show that SQ_n can be computed by γ-nets with two gates, where γ is any real-valued activation function at least three times continuously differentiable in some small neighborhood. In section 3 we prove that any binary threshold network accepting SQ_n must have size at least $\Omega(\log n)$.

A preliminary version of this result appeared in (DasGupta and Schnitger 1993).

2 Computing SQ_n by γ-nets

We say, that a function $\gamma : \mathcal{R} \to \mathcal{R}$ has the Q-property, if and only if there exist real numbers a and $\delta > 0$ such that

(a) $\gamma(x)$ is at least 3 times continuously differentiable in the interval $[a - \delta, a + \delta]$ and

(b) $\gamma''(a) \neq 0$.

Notice that the standard sigmoid $\sigma(x) = \frac{1}{1+e^{-x}}$ has the Q-property. Next we show that SQ_n can be computed with relatively large separation by small γ-nets with small weights, provided γ has the Q-property.

Theorem 2.1 *Assume that γ has the Q-property. Then there is a γ-net with two gates which accepts SQ_n with separation $\Omega(1)$.*

Moreover, all weights are bounded in absolute value by a polynomial in n.

The proof of Theorem 2.1 utilizes the Q-property to extract square polynomials from γ. In particular, we approximate the quadratic polynomial $[x]^2 - [y]$ with *small* error. Finally, the function SQ_n is computed with $\Omega(1)$-separation by comparing the approximated polynomial with a suitable threshold value.

Proof of Theorem 2.1. Since γ is at least 3-times continuously differentiable in $I = [a-\delta, a+\delta]$, we obtain

$$\gamma(a+z) = \gamma(a) + \gamma'(a) \cdot z + \frac{\gamma''(a)}{2} \cdot z^2 + r(z),$$

where $r(z) = \frac{\gamma'''(\theta_z)}{6} \cdot z^3$ (for $z \in [-\delta, \delta]$ and some $\theta_z \in I$). Moreover, by continuity, there is a constant Max with $\mid \gamma'''(u) \mid \leq Max$ for all $u \in I$. We set

$$L = \max\{\left\lceil \frac{4 \cdot Max \cdot n^3}{3 \cdot \gamma''(a)} \right\rceil, \left\lceil \frac{n}{\delta} \right\rceil\}.$$

Since $0 \leq [x] \leq n$, we obtain $\left|\frac{[x]}{L}\right| \leq \left|\frac{n}{L}\right| \leq \delta$ and thus

$$L^2 \cdot \gamma(a + \frac{[x]}{L}) = L^2 \cdot \gamma(a) + L \cdot \gamma'(a) \cdot [x] + \frac{\gamma''(a)}{2} \cdot [x]^2 + \frac{\gamma'''(\theta_{[x]})}{6L} \cdot [x]^3$$

or, equivalently,

$$\left(\frac{2L^2}{\gamma''(a)} \cdot \gamma(a + \frac{[x]}{L}) - \frac{2L^2 \cdot \gamma(a)}{\gamma''(a)} - \frac{2L \cdot \gamma'(a)}{\gamma''(a)} \cdot [x] \right) - [x]^2 = \frac{2 \cdot \gamma'''(\theta_{[x]})}{6L \cdot \gamma''(a)} \cdot [x]^3. \tag{1}$$

Also, since $|\gamma'''(\theta_{[x]})| \leq Max$, we obtain the bound

$$\left| \frac{2\gamma'''(\theta_{[x]})}{6L \cdot \gamma''(a)} \cdot [x]^3 \right| \leq \frac{1}{4}.$$

The γ-net accepting SQ_n consists of a first neuron computing $u(x) = \gamma(a + \frac{[x]}{L})$. The second neuron, the output neuron, computes the weighted sum

$$v(x,y) = \frac{2L^2}{\gamma''(a)} \cdot u - \frac{2L^2 \cdot \gamma(a)}{\gamma''(a)} - 2L \cdot \frac{\gamma'(a)}{\gamma''(a)} \cdot [x] - [y].$$

As a consequence of equation (1), the output neuron approximates $[x^2] - [y]$ with error at most $\frac{1}{4}$. Thus,

$$\left([x]^2 \geq [y] \Leftrightarrow v(x,y) \geq -\frac{1}{4} \right) \quad and \quad \left([x]^2 < [y] \Leftrightarrow v(x,y) \leq -\frac{3}{4} \right).$$

Thus, setting $t_C = -\frac{1}{2}$ in Definition 1.2, it follows that our γ-net C accepts SQ_n with separation at least $\frac{1}{4}$. The weight bound follows, since $L = O(n^3)$. ♠

3 A Lower Bound for "Unary Squaring"

We have to show the following result.

Theorem 3.1 *Any binary threshold network accepting SQ_n must have size at least $\Omega(\log n)$.*

Let $SQ_n^=$ denote the language

$$SQ_n^= = \{(x,y) : x \in \{0,1\}^n, y \in \{0,1\}^{n^2} \text{ and } [x]^2 = [y]\}.$$

Proposition 3.1 *Assume that there exists a binary threshold network of size t_n accepting SQ_n. Then there exists a binary threshold network of size $t_n + t_{n+1} + 1$ accepting $SQ_n^=$.*

Proof. Since there exists a binary threshold network of size t_n accepting SQ_n, there also exists a binary threshold network of size t_n accepting $\overline{SQ_n}$, the complement of SQ_n.

We show how to compute the language

$$SQ_n^\leq = \{(x,y) : x \in \{0,1\}^n, y \in \{0,1\}^{n^2} \text{ and } [x]^2 \leq [y]\}.$$

Consider the binary threshold network for $\overline{SQ_{n+1}}$, with binary inputs $x_1, \ldots, x_{n+1}$ and $y_1, \ldots, y_{(n+1)^2}$. We set $x_{n+1} = 0$, $y_{n^2+1} = 1$ and $y_{n^2+2} = \cdots = y_{(n+1)^2} = 0$. With those bits fixed, the threshold network for $\overline{SQ_{n+1}}$ accepts the input $(x_1, x_2, \ldots, x_n, y_1, y_2, \ldots, y_{n^2})$ if and only if $(\sum_{i=1}^{n^2} x_i)^2 < \sum_{j=1}^{n^2} y_j + 1$. But this is equivalent to $(\sum_{i=1}^{n^2} x_i)^2 \leq \sum_{j=1}^{n^2} y_j$. Hence, size t_{n+1} threshold circuits can compute $SQ_n^\leq$.

But note that $SQ_n^= \equiv SQ_n \wedge SQ_n^\leq$. Hence, $SQ_n^=$ can be computed with $t_n + t_{n+1} + 1$ threshold gates. ♠

Thus it suffices to show that any binary threshold network accepting $SQ_n^=$ must have size at least $\Omega(\log n)$. Let us assume that C_s is a binary threshold network with s gates accepting $SQ_n^=$. Our approach will be to successively trivialize (i.e. *partially* fix the outcomes of) the gates of C_s by fixing appropriate bits of the input (x,y). The process of trivialization starts with source

gates, continues with gates all of whose immediate predecessors have been trivialized and finally terminates with the sink gate of C_s.

Let us assume that the process of trivilization has reached gate g. Moreover assume that the bits $x_{k+1}, \ldots, x_n$ and $y_{l+1}, \ldots, y_{n^2}$ have been fixed with $(x_{k+1}, \ldots, x_n) = \xi$ and $(y_{l+1}, \ldots, y_{n^2}) = \eta$. Determine α with $l = 2\alpha k + k^2$ and set

$$\text{domain}(k, l, \xi, \eta) = \{(x, \xi; y, \eta) : \ x \in \{0,1\}^k, y \in \{0,1\}^l \text{ and } 2\alpha[x] \leq [y] \leq 2\alpha[x] + [x]^2\}$$

as well as

$$SQ_n^=(k, l, \xi, \eta) = \{(x, \xi; y, \eta) \in \text{domain}(k, l, \xi, \eta) : \ (\alpha + [x])^2 = \alpha^2 + [y]\}.$$

(α will coincide with the number of x-bits that are set to one. Therefore, $\alpha \geq 0$ and α will only increase during the trivialization process.) We demand that the following *invariant* holds for $\text{domain}(k, l, \xi, \eta)$:

(a) $l = 2\alpha k + k^2$, where α is a *non-negative* integer.

(b) Each already processed gate is constant over $\text{domain}(k, l, \xi, \eta)$.

(c) For every $u \in \text{domain}(k, l, \xi, \eta)$: C_s accepts u if and only if $u \in SQ_n^=(k, l, \xi, \eta)$.

In other words, all previously processed gates have been trivialized over $\text{domain}(k, l, \xi, \eta)$, whereas the network C_s still accepts the non-trivial language $SQ_n^=(k, l, \xi, \eta)$. Thus, if $g(x, y)$ denotes the function computed by gate g for $(x, \xi; y, \eta) \in \text{domain}(k, l, \xi, \eta)$, then we obtain the representation

$$g(x, y) \equiv (\sum_{i=1}^{k} a_i x_i + \sum_{i=1}^{l} b_i y_i \geq t).$$

This follows, since g only depends on the free inputs of C_s and a constant threshold-value t (t is completely determined by the old threshold-value of g in the given circuit and the constant outputs of the already processed and therefore trivialized gates). Next we make a few basic observations concerning the trivialization process:

Proposition 3.2 **(a)** *Before the process of trivialization starts, the invariant holds with* $k = n, l = n^2$ *and (consequently)* $\alpha = 0$.

(b) *Assume that the sink of* C_s *has been processed. Then, the processed network cannot accept* $SQ_n^=(k, l, \xi, \eta)$, *unless* $k = 0$.

(c) *Assume that* k *decreases by at most a factor of* $1/96$ *for each processed gate. Then,* C_s *must have at least* $\Omega(\log n)$ *gates in order to correctly accept* $SQ_n^=$.

Proof. **(a)** is immediate and **(c)** is a direct consequence of **(b)**. It remains to verify part **(b)**. Assume that the sink of C_s has been processed and domain(k, l, ξ, η) is obtained. Then the sink (and thus the network) is constant for all elements of domain(k, l, ξ, η), but accepts $SQ_n^=(k, l, \xi, \eta)$. This language, however, is not constant for $k > 0$: we have $(0^k, \xi, 0^l, \eta) \in SQ_n^=(k, l, \xi, \eta)$, whereas $(0^k, \xi, 10^{l-1}, \eta) \notin SQ_n^=(k, l, \xi, \eta)$ (observe that $l = 2\alpha k + k^2 > 0$). ♠

We will assume from now on that n is a power 96. If this is not the case, then it suffices to set an appropriate number of x- and y-bits to zero such the number of free x-bits equals the next lower power of 96 (and the number of free y-bits is a square of the number of free x-bits).

Assume that the invariant holds for domain(k, l, ξ, η) (with $l = 2\alpha k + k^2$) and that we fix additional bits leaving K x-bits and L y-bits free. Assume that, among those additionally fixed bits, r x-bits are set to one and s y-bits are set to one. Let ξ' (resp. η') be the set of fixed x-bits (resp. y-bits) including the additionally fixed bits. When does the invariant hold for domain(K, L, ξ', η')?

Proposition 3.3 *The invariant holds for domain*(K, L, ξ', η'), *provided*

- $s = 2\alpha r + r^2$ *and*
- $L = 2(\alpha + r)K + K^2$ *(and hence* α *is replaced by* $\alpha + r$*).*

Proof. We first show that domain$(K, L, \xi', \eta') \subseteq$ domain(k, l, ξ, η). Let $u = (x_1, x_2, \xi; y_1, y_2, \eta)$ be an arbitrary element of domain(K, L, ξ', η'), where the bits in x_2 (resp. y_2) have been freshly fixed. Consequently,

$$2(\alpha + r)[x_1] \leq [y_1] \leq 2(\alpha + r)[x_1] + [x_1]^2$$

and therefore

$$\begin{aligned}
2\alpha[(x_1,x_2)] &= 2\alpha[x_1] + 2\alpha r \leq 2(\alpha+r)[x_1] + 2\alpha r \\
&\leq [y_1] + 2\alpha r \\
&\leq [y_1] + 2\alpha r + r^2 = [(y_1,y_2)] \\
&\leq 2(\alpha+r)[x_1] + [x_1]^2 + 2\alpha r + r^2 \\
&\leq (2\alpha[(x_1,x_2)] - 2\alpha r + 2r[x_1]) + [x_1]^2 + 2\alpha r + r^2 \\
&= 2\alpha[(x_1,x_2)] + 2r[x_1] + [x_1]^2 + r^2 \\
&= 2\alpha[(x_1,x_2)] + [(x_1,x_2)]^2
\end{aligned}$$

Condition (a) of the invariant is satisfied because of the assumed relationships between K and L. Since constant gates remain constant if the domain is further restricted, condition (b) of the invariant is satisfied as well.

Now we consider condition (c).

$$\begin{aligned}
u \in SQ_n^{=}(K,L,\xi',\eta') &\Leftrightarrow (\alpha + r + [x_1])^2 = (\alpha+r)^2 + [y_1] \\
&\Leftrightarrow (\alpha + [(x_1,x_2)])^2 = \alpha^2 + 2\alpha r + r^2 + [y_1] \\
&\Leftrightarrow (\alpha + [(x_1,x_2)])^2 = \alpha^2 + [(y_1,y_2)] \\
&\Leftrightarrow u \in SQ_n^{=}(k,l,\xi,\eta) \\
&\Leftrightarrow C_s \text{ accepts } u. \qquad \spadesuit
\end{aligned}$$

Hence, to complete the proof of Theorem 3.1, we have to perform the trivialization process such that

- k, the number of free bits, decreases by at most a factor of $1/96$ for each processed gate and
- the invariant is maintained whenever a gate is processed (by setting $2\alpha r + r^2$ new y-bits to one, if r new x-bits are set to one.)

We have to describe the trivialization of gate g. Observe that $l = 2\alpha k + k^2$ holds. Moreover, remember that g computes the function $g(x,y) \equiv \left(\sum_{i=1}^{k} a_i x_i + \sum_{i=1}^{l} b_i y_i \geq t\right)$.

Throughout the trivialization process we will assume that $k = 96^i$ for some positive integer i. First we will reduce k by a factor of *at most* $1/96$ after one step of the trivilization process. If we are left with $k' > 96^{i-1}$ free bits, then an additional $k' - 96^{i-1}$ bits can be set to zero without violating the invariant.

Our first goal is to enforce that all a_i's have the same sign and that all b_i's have the same sign. To achieve identical sign for the a_i's, we set an appropriate collection of $\frac{k}{2}$ x-bits to zero. We repeat this procedure for the y-bits by setting $\frac{l}{2}$ y-bits to zero, but we also set an additional number of $\frac{k^2}{4}$ y-bits to zero. Thus, with

$$k_1 = \frac{k}{2} \quad \text{and} \quad l_1 = \frac{l}{2} - \frac{k^2}{4},$$

we have $l_1 = (\alpha k + \frac{k^2}{2}) - \frac{k^2}{4}$ and therefore $l_1 = 2\alpha k_1 + k_1^2$. Proposition 3.3 guarantees, that the invariant holds for domain(k_1, l_1, ξ, η)) (where ξ (resp. η) consists of all fixed x-bits (resp. y-bits). We now face, after appropriate renumbering positions if necessary, the following situation.

(a) Bits $x_1, \ldots, x_{k_1}$ and $y_1, \ldots, y_{l_1}$ are free and

(b) $|a_1| \leq \ldots \leq |a_{k_1}|$ as well as $|b_1| \leq \ldots \leq |b_{l_1}|$.

Case 1. The x- and y-weights are both nonnegative.

Set $r = \frac{k_1}{2}$ and $s = 2\alpha r + r^2$. Let $\beta_1 = \sum_{i=k_1-r+1}^{k_1} a_i$ and $\beta_2 = \sum_{i=l_1-s+1}^{l_1} b_i$.

Case 1.1. $\beta_1 + \beta_2 \geq t$.

We set the last r bits of x and the last s bits of y to one (i.e., $x_{k_1-r+1} = \cdots = x_{k_1} = 1$ and $y_{l_1-s+1} = \cdots = y_{l_1} = 1$). Since the x- and y-weights are nonnegative, gate g has been trivialized: its output will always be one. Thus $k_1 - r = r$ free x-bits and $l_1 - s$ free y-bits remain. Observe that

$$l_1 - s = l_1 - (2\alpha r + r^2) = 2\alpha k_1 + k_1^2 - (2\alpha r + r^2) = 2\alpha r + 3r^2 = 2(\alpha + r)r + r^2$$

and the invariant is satisfied with Proposition 3.3. We are left with $r = k/4 > k/96$ free x-bits.

Case 1.2. $\beta_1 + \beta_2 < t$.

This time we set the last r bits of x as well as the last $s + 2r^2$ bits of y to zero. Observe first that r bits of x and $l_1 - s - 2r^2 = 2\alpha r + r^2$ bits of y remain free.

Next observe that, $\sum_{i=1}^{r} a_i \leq \beta_1$ and $\sum_{i=1}^{l_1-s-2r^2} b_i \leq \beta_2$ (since $l_1 - s - 2r^2 = 2\alpha r + r^2 = s \leq s + 2r^2$. Hence, gate g will always output zero and thus has been trivialized.

The invariant is guaranteed with Proposition 3.3, since α is unchanged. Again, $r = k/4 > k/96$ free x-bits remain.

Case 2. The x- and y-weights are both nonpositive.

The construction is analogous to Case 1.

Case 3. The x-weights are nonnegative and the y-weights are nonpositive.

Let $\kappa = k_1/3$. Observe that $l_1 = 2\alpha(3\kappa) + (3\kappa)^2 = 6\alpha\kappa + 9\kappa^2$. First we partition the indices for the free x-bits into the three classes $S_x = \{1, \ldots, \kappa\}$, $M_x = \{\kappa + 1, \ldots, 2\kappa\}$ and $L_x = \{2\kappa + 1, \ldots, 3\kappa\}$.

Analogously, we three-partition the indices for the free y-bits into the sets S_y (of the first $2\alpha\kappa + 3\kappa^2$ y-positions), M_y (of the second $2\alpha\kappa + 3\kappa^2$ y-positions) and L_y (of the last $2\alpha\kappa + 3\kappa^2$ y-positions).

Let r be an integer. We say, that r is *legal*, provided $\frac{\kappa}{16} \leq r \leq \kappa$ (and thus $r \geq \frac{k}{96}$). We will make two attempts at trivializing gate g. In both attempts r bits of x (and $2\alpha r + r^2$ bits of y) will be fixed to one. Also, $r \geq \frac{k}{96}$ bits of x (and $2(\alpha + r)r + r^2 = 2\alpha r + 3r^2$ bits of y) will remain free. Thus, by Proposition 3.3, the invariant still holds and we have to only ensure, that gate g will be additionally trivialized.

In the first (second) attempt, we try to force g to be constantly zero (one). Then we show, that one of the two attempts has to be successful. In both attempts we only set x-bits with positions in M_x and y-bits with positions in M_y to one. Let $\xi = \frac{1}{|M_x|}\sum_{i \in M_x} a_i$, $\eta = \frac{1}{|M_y|}\sum_{i \in M_y} b_i$ and $\omega = \xi + 2\alpha\eta$. Observe that η is non-positive.

Attempt 1: Trying to set gate g to be constantly zero.

To keep the weighted sum of gate g as small as possible, we leave only the r bits of x corresponding to the first r positions of S_x free. Also, only the $2\alpha r + 3r^2$ bits of y, corresponding to the first $2\alpha r + 3r^2$ positions of L_y, are left free as well.

We then set x_i (for the **first** r positions $i \in M_x$) as well as y_i (for the **last** $2\alpha r + r^2$ positions $i \in M_y$) to one. The remaining bits to be fixed are all set to zero.

Assume for the moment that all free bits are set to 0. Then the weighted sum of gate g will be *upper-bounded* by

$$r \cdot \xi + (2\alpha r + r^2)\eta = r \cdot (\xi + 2\alpha\eta) + r^2 \cdot \eta = r \cdot \omega + r^2 \cdot \eta.$$

This follows, since the x-weights are in increasing order and hence the average of the first r weights of M_x is not bigger than the overall average. Moreover, the y-weights are in decreasing order and hence the average of the last $2\alpha r + r^2$ weights of M_y is not bigger than the overall average.

How large can the contribution of the free bits be, when (say) r' free bits of x are set to one? Since the condition $2(\alpha + r)[x] \leq [y] \leq (2\alpha + r)[x] + [x]^2$ has to be satisfied and since we are trying to maximize the weighted sum of gate g, as few y-bits as possible (namely $2(\alpha + r)r'$ bits) will be set to one. Thus the contribution of the free bits is *at most*

$$r'\xi + 2(\alpha + r)r'\eta = r' \cdot (\xi + 2\alpha\eta + 2r \cdot \eta) = r' \cdot (\omega + 2r \cdot \eta).$$

This follows, since the x-weights are in increasing order and hence the average of any r' weights of S_x is not bigger than the average of the weights in M_x. Moreover, the y-weights are in decreasing order and hence the average of any $2(\alpha + r)r'$ weights of L_y is not bigger than the average of the weights in M_y.

Summarizing, the value of g is either upper-bounded by

- $r \cdot \omega + r^2\eta$, if $\omega + 2r\eta \leq 0$, or by
- $2r \cdot \omega + 3r^2\eta$ otherwise.

Consequently we are done, if we can find a legal value for r such that the respective upper bound is less than t, the threshold value of gate g. But let us assume, that our first attempt fails for all

legal values of r.

Attempt 2: Trying to set gate g to be constantly one.

To make the weighted sum of gate g as large as possible, we leave only the r bits of x corresponding to the first r positions of L_x free. Also, the $2\alpha r + 3r^2$ bits of y, corresponding to the first $2\alpha r + 3r^2$ positions of S_y, are left free as well.

We set x_i (for the **last** r positions $i \in M_x$) as well as y_i (for the **first** $2\alpha r + r^2$ positions $i \in M_y$) to one. The remaining bits to be fixed are all set to zero.

Assume for the moment that all free bits are set to 0. Then the weighted sum of gate g will be *lower-bounded* by

$$r \cdot \xi + (2\alpha r + r^2)\eta = r \cdot \omega + r^2 \cdot \eta$$

How small can the contribution of the free bits be, when (say) r' free bits of x are set to one? Since the condition $2(\alpha + r)[x] \leq [y] \leq 2(\alpha + r)[x] + [x]^2$ has to be satisfied and since the weighted sum of gate g should be minimized, as many y-bits as possible (namely $2(\alpha + r)r' + (r')^2$ bits) will be set to one. And we obtain a contribution of *at least*

$$r' \cdot \xi + (2(\alpha + r)r' + (r')^2) \cdot \eta = r' \cdot (\xi + 2(\alpha + r)\eta + r' \cdot \eta) \geq r' \cdot (\xi + 2\alpha\eta + 3r\eta) = r' \cdot (\omega + 3r\eta).$$

Summarizing, the value of g is either lower-bounded by

- $2r \cdot \omega + 4r^2\eta$, if $\omega + 3r\eta \leq 0$, or by
- $r \cdot \omega + r^2\eta$ otherwise.

Consequently we are done, if we can find a legal value for r such that the respective lower bound is greater than or equal to t, the threshold value of gate g. But let us assume, that also this second attempt fails.

Failure of both attempts: Let

$$I_1 = \{r \ : \ \omega + 3r\eta > 0\}, \ \ I_2 = \{r \ : \ \omega + 3r\eta \leq 0, \ \omega + 2r\eta > 0\} \quad \text{and} \quad I_3 = \{r \ : \ \omega + 2r\eta \leq 0\}.$$

Case 3.1. $\omega + 3\frac{\kappa}{4}\eta \leq 0$.

Set $r_0 = \kappa/2$ and observe that r_0 and $2r_0$ are legal values for r. As a consequence of the case assumption, we obtain $\omega + 2r_0\eta \leq \omega + 3\frac{\kappa}{4}\eta \leq 0$ and hence $[r_0, \infty] \subseteq I_3$. Since both attempts fail,

$$t \leq (2r_0)\omega + (2r_0)^2\eta \text{ and } 2r_0\omega + 4r_0^2\eta < t.$$

But this is impossible and one of the two attempts has to succeed for a legal value of r.

Case 3.2. $\omega + 3\frac{\kappa}{4}\eta > 0$.

Let $r_1 = \frac{\kappa}{4}$ and observe that r_1 and $\frac{r_1}{4}$ are legal values of r. Moreover $[-\infty, r_1] \subseteq I_1$. Since both attempts fail,

$$r_1\omega + r_1^2\eta < t \leq 2r_1\omega + 3r_1^2\eta \text{ and } r_1\omega + r_1^2\eta < t \leq 2(\frac{r_1}{4})\omega + 3(\frac{r_1}{4})^2\eta.$$

As a consequence

$$-2r_1\eta < \omega \text{ and } \omega < -\frac{13}{8}r_1\eta.$$

We again arrive at a contradiction and one of the two attempts has to succeed for a legal value of r.

Case 4. The x-weights are nonpositive and the y-weights are nonnegative.

Analogous to Case 3 and the proof of Theorem 3.1 is complete.♠

Remark 3.1 (a) *The lower bound of Theorem 3.1 is "almost" tight. It is quite easy to construct a binary threshold network of* $O(\log n)$ *gates that computes the binary representation of* $[x]$. *Now we apply the Schönhage-Strassen multiplication algorithm to obtain a binary threshold network of size* $O(\log n \cdot \log\log n \cdot \log\log\log n)$ *which computes all the bits of* $[x]^2$. *Hence* SQ_n *can be computed with* $O(\log n \cdot \log\log n \cdot \log\log\log n)$ *threshold gates (with a final gate comparing* $[x]^2$ *and* $[y]$*).*

(b) *A better separation of binary threshold networks and* γ*-nets might be possible by considering the language* L *of binary squaring, i.e.*

$$L = \{(x, y) : x \in \{0,1\}^n, y \in \{0,1\}^{2n} \text{ such that } (\sum_{i=1}^{n} 2^{i-1}x_i)^2 \geq \sum_{i=1}^{2n} 2^{i-1}y_i\}.$$

The problem of deriving a superlogarithmic lower bound for networks with weights of unbounded size seems to be difficult however.

4 References

BOPPANA R., and SIPSER M. 1990. The complexity of finite functions, *in* Handbook of theoretical computer science: algorithms and complexity, J. van Leeuwen (ed), MIT Press.

DASGUPTA B., and SCHNITGER G. 1993. The Power of Approximating: A Comparison of Activation Functions, *in* "Advances in Neural Information Processing Systems 5" (Giles, C.L., Hanson, S.J., and Cowan, J.D., eds), Morgan Kaufmann, San Mateo, CA, pp. 615-622.

HAJNAL A., MAASS W., PUDLAK P., SZEGEDY M. and TURAN G. 1987. Threshold circuits of bounded depth, *in* "Proc. of the 28th IEEE Symp. on Foundations of Computer Science", pp. 99-110.

GOLDMANN M. and HASTAD J. 1991. On the power of small-depth threshold circuits, *Computational Complexity*, 1(2), pp. 113-129.

HÖFFGEN K-U. 1993. Computational limitations on training sigmoidal neural networks, *Information Processing Letters*, 46, pp. 269-274.

MAASS W. 1993. Bounds for the computational power and learning complexity of analog neural nets, *in* "Proc. of the 25th ACM Symp. Theory of Computing", pp. 335-344 .

MACINTYRE A., and SONTAG E. D. 1993. Finiteness results for sigmoidal 'neural' networks, *in* "Proc. 25th Annual Symp. on Theory Computing, pp. 325-334.

MAASS W., SCHNITGER G., and SONTAG E. D. 1991. On the computational power of sigmoid versus boolean threshold circuits,*in* "Proc. of the 32nd Annual Symp. on Foundations of Computer Science", pp. 767-776.

RAZBOROV A. A. 1987. Lower bounds on the size of bounded depth networks over a complete basis with logical addition, *Math. Notes of the Academy of Science of the USSR* 41(4), pp. 333-338.

REIF J. H. 1987. On threshold circuits and polynomial computation, *in* "Proceedings of the 2nd

Annual Structure in Complexity theory", pp. 118-123.

SONTAG E. D. 1990. Comparing Sigmoids and Heavisides, *in* "Proc. Conf. Info. Sci. and Systems", pp. 654-659.

WEGENER I. 1987. The complexity of Boolean functions, *Wiley-Teubner Series in Computer Science*.

WEGENER I. 1991. The complexity of the parity function in unbounded fan-in, unbounded depth circuits, *Theo. Comp. Sci*, 85, pp. 155-170.

ZHANG X-D. 1992. Complexity of neural network learning in the real number model, preprint, Comp. Sci. Dept., U. Mass.