Electronic Colloquium on Computational Complexity, Report No. 71 (2001)

Neural Circuits for Pattern Recognition with Small Total Wire Length*

Robert A. Legenstein & Wolfgang Maass
Institute for Theoretical Computer Science
Technische Universität Graz, Austria
{legi, maass}@igi.tu-graz.ac.at

October 24, 2001

Abstract

One of the most basic pattern recognition problems is whether a certain local feature occurs in some linear array to the left of some other local feature. We construct in this article circuits that solve this problem with an asymptotically optimal number of threshold gates. Furthermore it is shown that much fewer threshold gates are needed if one employs in addition a small number of winner-take-all gates. In either case the circuits that are constructed have linear or almost linear total wire length, and are therefore not unrealistic from the point of view of physical implementations.

1 Introduction

Biological neural circuits can solve a number of complex pattern recognition tasks very fast, in 100 - 150 milliseconds, see (Thorpe et al., 1996). Since the computational units of neural circuits are relatively slow compared with a transistor, observation gives rise to some optimism regarding the possibility to build artificial circuits, for example analog VLSI chips, that solve complex real-world pattern

*Research for this article was partially supported by the the Fonds zur Förderung der wissenschaftlichen Forschung (FWF), Austria, project P12153, and the NeuroCOLT project of the EC.

recognition tasks in real-time. Classical circuit complexity theory is of little help in the search for such super-efficient circuit designs. Apparently there are two reasons for this. The complexity of circuits is usually analyzed in terms of their number of gates, and much of the existing work focuses on the derivation of polynomial upper bounds for the number of gates. But most circuits that appear to be feasible from this point of view cannot be practically implemented, especially if the number n of input variables is very large (like for example in vision tasks where often $n \approx 10^6$). Furthermore even those circuit designs where one has been able to derive linear or almost linear upper bounds for the number of gates can usually not be implemented in VLSI because the required number of wires (= edges), or the required length of wires grows too fast with the number n of input variables. Therefore we focus in this article directly on the total wire length (the definition is given below) as the most salient complexity measure, the usually most restricted and hence arguably most relevant complexity measure for the practical implementation of an abstract circuit design.

Another obstacle for the application of classical circuit complexity theory to the design of efficient circuits for pattern recognition arises from the fact that most complexity studies focus on arithmetic and graph-theoretic problems, rather than on those computational tasks that typically arise in the context of pattern recognition. Both, in common machine vision approaches and in biological neural circuits for vision, the raw pixel image is first preprocessed by an array of local feature detectors (*e.g.*for the detection of edge segments, line segments, Gabor filters). Hence pattern recognition problems in vision typically require to find particular spatial arrangements of those local features, that are reported by local feature detectors. The local feature detectors are typically arranged in a one- or two- dimensional array that reflects the geometrical relationship between their receptive fields in the sensory space. In order to initiate a computational complexity analysis of algorithmic problems of this type we investigate in this article the arguably most simple problem of this type. We assure that there are two types of local feature detectors with binary output that are linearly arranged at n positions: detectors $a_0, \ldots, a_{n-1}$ for feature a and detectors $b_0, \ldots, b_{n-1}$ for feature b. The pattern recognition task is to decide whether feature a is reported at a location i to the left of some location j where feature b is reported. In other words, we analyze the circuit complexity of the Boolean function P^n_{LR} from $\{0,1\}^{2n}$ into $\{0,1\}$ with

$$P^n_{LR}(a_1, \ldots, a_n, b_1, \ldots, b_n) = \begin{cases} 1, & \text{if } \exists i,j (i < j \text{ and } a_i = b_j = 1) \\ 0, & \text{else .} \end{cases}$$

We investigate in this article circuits that compute P_{LR} with two types of gates that are both frequently discussed in models for neural computation: threshold gates and winner-take-all (WTA) gates. Both of these gates can be implemented very efficiently in analog VLSI, with an area that grows just linearly with the number k of inputs to the gate, see (Mead, 1989), and (Lazzaro et al., 1989). A threshold gate computes a Boolean function $T : \{0,1\}^k \to \{0,1\}$ of the form $T(x_1,\ldots,x_k) = 1 \Leftrightarrow \sum_{i=1}^{k} w_i x_i \geq w_0$. A winner-take-all gate with weights $w_1,\ldots,w_k$ computes a Boolean function $W : \{0,1\}^k \to \{0,1\}^k$ where for input $x_1,\ldots,x_k$ the ith output is 1 if and only if $w_i x_i > w_j x_j$ for all $j \neq i$.

We propose the following abstract model for estimating the total wire length required for the neural implementation of an abstract circuit design (which is formally defined as a directed graph with nodes labeled by specific types of gates, or by input- or output variables):

Gates, input- and output-ports of a circuit are placed on different nodes of a 2-dimensional grid (with unit distance 1 between adjacent grid nodes). Connections between them are represented by (unidirectional) wires that run through the grid-plane in any way that the designer wants, in particular wires may cross and need not run rectilinearly (wires are thought of as running in the 3 dimensional space above the plane, without charge for vertical wire segments)[1]*. We define the minimal value of the sum of all wire lengths that can be achieved by any such arrangement as the* total wire length *of the circuit.*

We would like to make this model also applicable to cases where for $k > 2$ threshold-, or winner-take-all functions of k inputs are computed in analog VLSI by efficient subcircuits that employ a number of transistors, total wire length and area that are all linear in k, with a setting time that is independent of k [2]*. We model such computational modules as "threshold gates" or "winner-take-all gates" of k inputs, that take one unit of time for their computation like all the other gates, but which occupy each a set of k intersection points of the grid that are all connected by an undirected wire (whose length contributes to the total wire length) in some arbitrary fashion*[3].

The attractiveness of this model lies in its mathematical simplicity, and in its generality (see (Legenstein et al., 2001) and (Legenstein et al. ,2001b) for a more

[1] We will allow that a wire from a gate or input port may branch and provide input to several other gates. For reasonable bounds on the maximal fan-out (10^4 in the case of neural circuits) this is realistic both for neural circuits and for VLSI.

[2] see (Lazzaro et al., 1989)

[3] Any one of these k nodes may be used to provide one of the k inputs or to extract one of the outputs of the function.

detailed analysis of the complexity measure total wire length, and results on the total wire length of circuits that solve two other pattern recognition tasks). It provides a rough estimate for the cost of connectivity both in artificial (basically 2-dimensional) circuits and in neural circuits, where 2-dimensional wire crossing problems are apparently avoided (at least on a small scale) since dendritic and axonal branches are routed through 3-dimensional cortical tissue. We also give bounds on the complexity of our circuit designs in the common abstract model for VLSI.

We refer to Section 12.2 in (Savage, 1998) for the precise definition of the abstract model for VLSI-area to which the theorems in this article refer. One assumes there that gates, input- and output-ports and wires cover rectilinear areas with a width and separation of at least λ. Areas occupied by different gates, input- and output-ports are not allowed to intersect with one another. Areas occupied by wires may intersect with areas occupied by gates, input- and output-ports and also with other wires, but there is a constant bound μ on the number of wire areas to which a point of the plane may belong. The complexity measure induced by this model is the *area* of the smallest rectangle that encloses the circuit. We follow (Savage, 1998) in assuming that in the VLSI-model one unit of time is needed to transmit a bit along a wire (of any length), and also for each gate switching. However in contrast to (Savage, 1998) we always assume that all inputs are presented in parallel.

We will show in Theorem 2.1 that P^n_{LR} can be computed by a circuit consisting of $O(\log n)$ threshold gates in depth 2, with a total wire length of $O(n \log n)$. Theorem 2.2 implies that no feedforward circuit can compute P^n_{LR} with fewer threshold gates. Finally it is shown in Theorem 2.3 that P^n_{LR} can be computed by a circuit of depth 2 consisting of two winner-take-all gates and one threshold gate, with total wire length $O(n)$. This result demonstrates that winner-take-all gates can in some contexts be computationally much more powerful than threshold gates, although they do not require much more area in analog VLSI (see (Maass, 2000) for some more general results in this direction).

2 Global Pattern Detection in 1-Dimensional Maps

We start the analysis of this pattern recognition task by showing that P^n_{LR} can be computed very fast by a circuit consisting of $O(\log n)$ threshold gates. We also give bounds on the total wire length of this circuit and the area that it occupies in a VLSI layout.

Theorem 2.1. *P^n_{LR} can be computed by a feedforward circuit of depth 2, consisting of $2\log n + 1$ threshold gates with total wire length $O(n \log n)$ and area $O(n \log n)$ in a VLSI layout.*

Proof: Denote with $\underline{a} = (a_0, \ldots, a_{n-1})$ and $\underline{b} = (b_0, \ldots, b_{n-1})$ the two vectors of input features. It will be convenient to denote the position l of the leftmost occurring feature a with $min(\underline{a})$ and the position r of the rightmost occurring feature b with $max(\underline{b})$. Note that these functions are not defined if there is no feature a respectively b present. The following precise definition eliminates this ambiguity. We define $min(\underline{a}) = min\{i|a_i = 1\}$ if $a \neq (0, \ldots, 0)$ and $min(\underline{a}) = n - 1$ otherwise. Furthermore we define $max(\underline{b}) = max\{i|b_i = 1\}$ if $b \neq (0, \ldots, 0)$ and $max(\underline{b}) = 0$ otherwise. Note that with this simple definition, $P^n_{LR}(\underline{a}, \underline{b}) = 1 \Leftrightarrow min(\underline{a}) < max(\underline{b})$. We construct a threshold circuit which computes the binary encoding of $min(\underline{a})$ and $max(\underline{b})$ in its first layer. Let us call the function that maps $\underline{a}$ onto the binary representation of $min(\underline{a})$ *MinMux* and the function that maps $\underline{b}$ onto the binary representation of $max(\underline{b})$*MaxMux* respectively. The comparison of their outputs yields the desired output of P^n_{LR}.

For convenience, let $n = 2^k$ for some natural number k. The precise definitions of the functions *MinMux* and *MaxMux* are as follows.

$$MinMux^n : \{0,1\}^n \to \{0,1\}^k \text{ is defined by}$$
$$MinMux^n(\underline{a}) = \begin{cases} \text{binary encoding of } min\{i|a_i = 1\}, & \text{if } \exists i(a_i = 1) \\ \text{binary encoding of } n-1, & \text{otherwise .} \end{cases}$$
$$MaxMux^n : \{0,1\}^n \to \{0,1\}^k \text{ is defined by}$$
$$MaxMux^n(\underline{b}) = \begin{cases} \text{binary encoding of } max\{i|b_i = 1\}, & \text{if } \exists i(b_i = 1) \\ \text{binary encoding of } 0, & \text{otherwise .} \end{cases}$$

This comparison of the two $\log n$-bit binary numbers represented by $MinMux$ and $MaxMux$ can be carried out by an additional threshold gate with weights linear in n.

In the following, we construct a circuit consisting of $\log n$ threshold gates that computes *MinMux*. Note that, for any input assignment, setting $a_{n-1} = 1$ does not change the value of the function. We will use this trick to make sure that the output of the circuit is the binary encoding of $n-1$ if there is no feature a present.

Let m_j denote the j-th output bit of $MinMux^n$ ($0 \leq j \leq k-1$), such that $min(\underline{a}) = \sum_{j=0}^{k-1} 2^j m_j$. The j-th bit of the binary encoding of some natural number x is 1 if $\lfloor \frac{x}{2^j} \rfloor \equiv 1 \mod 2$ and 0 otherwise.

This leads to the following threshold function for m_j:

$$m_j(a_0,\ldots,a_{n-1}) = \begin{cases} 1, & \text{if } \sum_{i=0}^{n-1} a_i 2^{n-i}(-1)^{1+(\lfloor \frac{i}{2^j} \rfloor \mod 2)} \geq 1 \\ 0, & \text{otherwise .} \end{cases}$$

Let $l = min(\underline{a})$ and suppose that $\lfloor \frac{x}{2^j} \rfloor \equiv 0 \mod 2$. It follows that

$$\sum_{i=0}^{n-1} a_i 2^{n-i}(-1)^{1+(\lfloor \frac{i}{2^j} \rfloor \mod 2)} \leq$$
$$2^{n-l}(-1) + \sum_{i=l+1}^{n-1} 2^{n-i} \leq$$
$$-2^{n-l} + \sum_{i=1}^{n-l-1} 2^i \leq -2 < 1$$

and the output of the threshold gate is 0. Suppose that $\lfloor \frac{x}{2^j} \rfloor \equiv 1 \mod 2$. It follows that

$$\sum_{i=0}^{n-1} a_i 2^{n-i}(-1)^{1+(\lfloor \frac{i}{2^j} \rfloor \mod 2)} \geq$$
$$2^{n-l}(+1) - \sum_{i=l+1}^{n-1} 2^{n-i} \geq$$
$$2^{n-l} - \sum_{i=1}^{n-l-1} 2^i \geq 1$$

and the output of the threshold gate is 1. Hence, m_j is the j-th bit of the binary representation of $min(\underline{a})$.

MaxMux can be constructed in a similar manner. Hence, each m_j can be computed by one threshold gate and the depth and size of the circuit given in Theorem 2.2 follow.

The VLSI-layout of the circuit for P^4_{LR} is shown in Figure 1a. We place the gates for $MinMux$ on rows beneath $a_0,\ldots,a_{n-1}$ and the gates for $MaxMux$ on rows beneath $b_0,\ldots,b_{n-1}$. Since the circuit consists of $\log n$ gates for $MinMux^n$ and $\log n$ gates for $MaxMux^n$ this occupies $O(\log n)$ rows. The comparison gate can be placed in the column between those gates. Hence, the layout of the circuit occupies $O(n \log n)$ area. A layout to estimate the total wire length is similar. The

layout of the circuit for P^4_{LR} in is shown in Figure 1b. Simply replace a threshold gate of k inputs by k nodes that are connected by a common wire to sum up the inputs. This results in a wire length of $O(n)$ within each gate. The wire from an input port to its successor gates may spread and hence is $O(\log n)$ in length. The comparison gate has a total wire length of $O(\log n)$. Summing up those lengths, results in a total wire length of $O(n \log n)$.

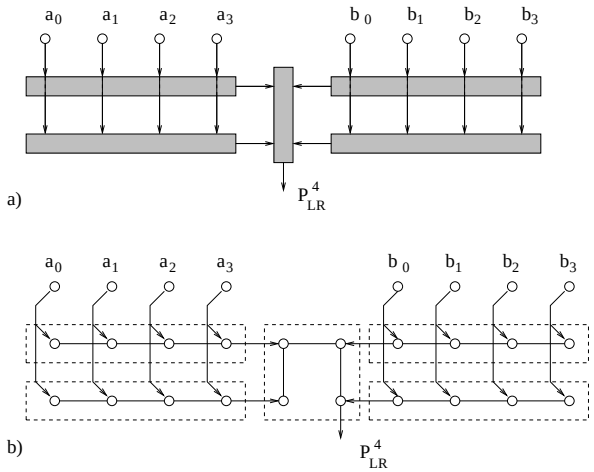

Figure 1: a) The VLSI-circuit layout for P^4_{LR}. The gates for $MinMux$ and $MaxMux$ are placed on rows beneath the inputs. The area used by this layout is $O(n \log n)$. b) A layout to estimate the total wire length of the circuit. A threshold gate of k inputs is represented by k nodes that are connected by a wire (wires without arrows). Such gates are indicated by a dashed rectangle. The total wire length is $O(n \log n)$.

■

The following lower bound result shows that the number of threshold gates used by the circuit of Theorem 2.1 is asymptotically optimal:

Theorem 2.2. *Any feedforward circuit consisting of threshold gates needs to have at least $\Omega(\log n)$ gates for computing P^n_{LR}.*

We use the gate-elimination method to prove Theorem 2.2. The gate-elimination method was used widely in classic circuit complexity theory. It was

used in the context of threshold circuits in a paper by Georg Schnitger and Bhaskar DasGupta (see (DasGupta et al., 1996)). In our case we have to exhibit some properties of P_{LR} that allow us to assign constants to inputs of a circuit S_n that computes P^n_{LR}, such that the circuit computes P_{LR} on the remaining non-constant variables. Furthermore, we use these properties to show that at least one threshold gate computes a constant after the assignment of constants to at most $\frac{63n}{64}$ of its input variables. We use this restriction to construct a circuit that computes $P^{n/64}_{LR}$ and has at least one gate less that S_n. Hence, the size of S_n is at least $S_{n/64} + 1$, which we use as an induction step. The induction hypothesis is that a circuit S_n that computes P^n_{LR} consists of at least $\lfloor \log_{64} n \rfloor$ threshold gates [4].

Proof: We will at first exhibit the three properties of P_{LR} that will be the basis for the proof. Then we will show, how to eliminate one threshold gate in a circuit computing P_{LR} by assigning constants to a fixed fraction of its inputs. Finally, we will use this gate-elimination to give an inductive prove of the lower bound.

The properties of P_{LR} given below are easy to verify.

property 1:

$$P^n_{LR}(a_0, \ldots, a_{i-1}, 0, a_{i+1}, \ldots, a_{n-1}, b_0, \ldots, b_{i-1}, 0, b_{i+1}, \ldots, b_{n-1}) = P^{n-1}_{LR}(a_0, \ldots a_{i-1}, a_{i+1}, \ldots, a_{n-1}, b_0, \ldots, b_{i-1}, b_{i+1}, \ldots, b_{n-1})$$
for all $i \in \{0, \ldots, n-1\}$.

property 2:

$$P^n_{LR}(0, \ldots, 0, a_{k+1}, \ldots, a_{n-1}, 1, \ldots, 1, b_{k+1}, \ldots, b_{n-1}) = P^{n-k}_{LR}(a_{k+1}, \ldots, a_{n-1}, b_{k+1}, \ldots, b_{n-1}) \quad \text{for all } k \in \{0, \ldots, n-2\} \quad .$$

property 3:

$$P^n_{LR}(a_0, \ldots, a_{n-1-k}, 1, \ldots, 1, b_0, \ldots, b_{n-1-k}, 0, \ldots, 0) = P^{n-k}_{LR}(a_1, \ldots, a_{n-1-k}, b_1, \ldots, b_{n-1-k}) \quad \text{for all } k \in \{1, \ldots, n-1\} \quad .$$

Let S_n be a threshold circuit computing P^n_{LR}. We show how to eliminate one gate in S_n by exploiting the properties of P_{LR} given above. We assume that n is a power of 64. If it is not, use property 1 to obtain a threshold circuit such that the number of non-constant inputs to the circuit is the next lower power of 64.

[4] $\lfloor x \rfloor$ denotes the floor of x, which is $\lfloor x \rfloor = max\{y \in \mathbb{N} \cup \{0\} | y \leq x\}$.

Let g be a gate in S_n which does not have an output of a gate as one of its inputs. Then g computes the function

$$g(\underline{a},\underline{b}) = \begin{cases} 1, & \text{if} \quad \sum_{i=0}^{n} u_i a_i + \sum_{i=0}^{n} v_i b_i \geq t \\ 0, & \text{else .} \end{cases}$$

First, we need all the weights for $\underline{a}$ to have same sign and all the weights for $\underline{b}$ to have same sign, where $sign(x) : \mathbb{R} \to \{-1,+1\}$ is $+1$ for all $x \in \mathbb{R}^+ \cup \{0\}$ and -1 otherwise. More formally, we want

$$\begin{aligned} sign(u_i) &= sign(u_j) \qquad \text{for all i,j} \\ sign(v_i) &= sign(v_j) \qquad \text{for all i,j .} \end{aligned}$$

This can be achieved by setting at most $3n/4$ variables in $\underline{a}$ and at most $3n/4$ variables in $\underline{b}$ to constant zero. By property 1, the circuit computes $P_{LR}^{n/4}$ on the remaining non-constant variables. We renumber the remaining $m = n/4$ variables in $\underline{a}$, the $n/4$ remaining variables in $\underline{b}$ (we preserve the order) and the corresponding weights. Let $\alpha_1 = \sum_{i=0}^{m/2-1} u_i$, $\alpha_2 = \sum_{i=m/2}^{m-1} u_i$, $\beta_1 = \sum_{i=0}^{m/2-1} v_i$ and $\beta_2 = \sum_{i=m/2}^{m-1} v_i$. We consider four cases:

case 1: $sign(u_i) = +1$, $sign(v_i) = -1$ for all $i = 0,\ldots,m-1$

case 1.1: $|\beta_1| > \alpha_2 - t$

We set $a_0 = \cdots = a_{m/2-1} = 0$ and $b_0 = \cdots = b_{m/2-1} = 1$. By property 2 of P_{LR}, the circuit computes $P_{LR}^{m/2}$ on the remaining non-constant variables. It follows that

$$-|\beta_1| + \sum_{i=m/2}^{m-1} |u_i| a_i - \sum_{i=m/2}^{m-1} |v_i| b_i < t - \alpha_2 + \alpha_2 - \sum_{i=m/2}^{m-1} |v_i| b_i < t \quad .$$

Hence $g(\underline{a},\underline{b}) = 0$ for all possible values of $a_{m/2},\ldots,a_{m-1}$ and $b_{m/2},\ldots,b_{m-1}$.

case 1.2: $|\beta_1| \leq \alpha_2 - t$

We set $a_{m/2} = \cdots = a_{m-1} = 1$ and $b_{m/2} = \cdots = b_{m-1} = 0$. By property 3 of P_{LR}, the circuit computes $P_{LR}^{m/2}$ on the remaining non-constant variables. It follows that

$$\sum_{i=0}^{m/2-1} |u_i| a_i - \sum_{i=0}^{m/2-1} |v_i| b_i + \alpha_2 \geq \sum_{i=0}^{m/2-1} |u_i| a_i - |\beta_1| + |\beta_1| + t \geq t \quad .$$

Hence $g(\underline{a}, \underline{b}) = 1$ for all possible values of $a_0, \ldots, a_{m/2-1}$ and $b_0, \ldots, b_{m/2-1}$.

In case 1, there remain $2 \cdot m/2 = 2 \cdot n/8$ non-constant variables after the restriction.

case 2: $sign(u_i) = -1$, $sign(v_i) = +1$
We can treat this case in a similar manner as case 1.

case 3: $sign(u_i) = +1$, $sign(v_i) = +1$

case 3.1: $\beta_1 \geq t$
We set $a_0 = \cdots = a_{m/2-1} = 0$ and $b_0 = \cdots = b_{m/2-1} = 1$. By property 2 of P_{LR}, the circuit computes $P_{LR}^{m/2}$ on the remaining non-constant variables. Furthermore it follows that $g(\underline{a}, \underline{b}) = 1$ for all possible values of non-constant inputs.

case 3.2: $\alpha_2 \geq t$
We set $a_{m/2} = \cdots = a_{m-1} = 1$ and $b_{m/2} = \cdots = b_{m-1} = 0$. By property 3 of P_{LR}, the circuit computes $P_{LR}^{m/2}$ on the remaining non-constant variables. It follows that $g(\underline{a}, \underline{b}) = 1$ for all possible values of non-constant inputs.

After any of these restrictions, $2 \cdot n/8$ non-constant variables remain and the circuit computes $P_{LR}^{n/8}$. For the following restriction, we can assume $\beta_1 < t$ and $\alpha_2 < t$. In this case, the weights for the second half of the remaining inputs to g are small. So our aim will be to eliminate variables with large weights in g. Then, the sum of the remaining inputs to g will be too small to reach the threshold and the gate will output a constant for all possible values of non-constant inputs. In a first step, we set all inputs that contribute to α_1 and β_1 constant zero. The effect is that all weights of a's are small. We use another restriction to maintain small weights for non-constant b's. Then we set the inputs that have largest weights constant zero. We need to do this for at most $3/4$ of the remaining variables to let the gate output zero for all possible values of non-constant inputs.

case 3.3 $\beta_1 < t$, $\alpha_2 < t$
We set $a_0 = \cdots = a_{m/2-1} = b_0 = \cdots = b_{m/2-1} = 0$. By property 1 of

P_{LR}, the circuit computes $P_{LR}^{m/2}$ on the remaining non-constant variables. Let l=$m/2 = n/8$. Again, renumber the non-constant variables and corresponding weights of g, so that

$$g(\underline{a}, \underline{b}) = \begin{cases} 1, & \text{if} \quad \sum_{i=0}^{l-1} u_i a_i + \sum_{i=0}^{l-1} v_i b_i \geq t \\ 0, & \text{else} . \end{cases}$$

Let $\alpha_1' = \sum_{i=0}^{l/2-1} u_i$, $\alpha_2' = \sum_{i=l/2}^{l-1} u_i$, $\beta_1' = \sum_{i=0}^{l/2-1} v_i$ and $\beta_2' = \sum_{i=l/2}^{l-1} v_i$. Since $\alpha_1' + \alpha_2' = \alpha_2 < t$, we have $\alpha_1' < t$. If $\beta_1' \geq t$, case 3.1 applies and $\frac{l}{2} = n/16$ variables remain. Finally we consider weights such that $\alpha_1' < t$ and $\beta_1' < t$. In this case, we set $a_i = b_i = 0$ for $i = \frac{l}{2}, \ldots, l-1$ to eliminate the second half of the inputs (property 1 of P_{LR} applies). Then, by property 1 of P_{LR}, we set those $l/4$ remaining variables in $\underline{a}$ to zero that have maximal weights. We also eliminate those $l/8$ remaining variables in $\underline{b}$ with maximal weights. It follows that the overall sum of the remaining variables cannot reach t and $g(\underline{a}, \underline{b}) = 0$ for all possible values of non-constant inputs. There will remain at least $2\frac{l}{8} = 2\frac{n}{64}$ non-constant variables.

case 4: $sign(u_i) = -1, sign(v_i) = -1$

We can treat this case in a similar manner as case 3.

We have constructed a threshold circuit that has at least one gate less and computes $P_{LR}^{n/64}$.

We use this property of S_n to give an inductive proof of the lower bound. The inductive hypothesis is, that $size(S_n) \geq \lfloor \log_{64} n \rfloor$. Since we use the floor of $\log_{64} n$ in the bound, we can use induction on n for all n of the form $n = 64^m$ for some natural number m.

In the basis case, we have $n = 64$. Use property 1 to obtain a circuit that computes P_{LR}^2. Since, $P_{LR}^2(a_0, a_1, b_0, b_1) = a_0 \wedge b_1$, a circuit that computes P_{LR}^{64} consists of at least one threshold gate. Hence, the hypothesis holds for the induction basis. For the induction step, consider a threshold circuit S_n that computes P_{LR}^n. We show that, if the size of S_n is small, then we can construct a circuit$S_{n/64}$ with smaller size than possible. Suppose that $size(S_n) < \log_{64} n$. Construct a circuit $S_{n/64}$ that computes $P_{LR}^{n/64}$ by eliminating one gate in S_n. Then, $size(S_{n/64}) \leq size(S_n) - 1 < \log_{64} n - 1 = \log \frac{n}{64}$. This is a contradiction. Hence, $size(S_n) \geq \log_{64} n$.

In analog VLSI the area occupied by a subcircuit that implements a winner-take-all gate is comparable to that for a threshold gate. Hence the next theorem demonstrates a drastic gain in efficiency if one employs modules for computing winner-take-all in addition to threshold gates. It combines the minimal possible computation time of 2 with a *linear* total wire length.

Theorem 2.3. *P^n_{LR} can be computed by a feedforward circuit of depth 2, consisting of two winner-take-all gates and one threshold gate, with total wire length and area $O(n)$.*

Proof: Denote with $\underline{a} = (a_0, \ldots, a_{n-1})$ and $\underline{b} = (b_0, \ldots, b_{n-1})$ the two vectors of input features. We construct a circuit that consists of two winner-take-all gates in the first layer and one threshold gate in the second layer. One winner-take-all gate marks the position of the leftmost occurring feature in $\underline{a}$ and the other winner-take-all gate marks the position of the rightmost occurring feature in $\underline{b}$. In the second layer, a single threshold gate with linear weights can compute the value of $P^n_{LR}(\underline{a}, \underline{b})$.

Let $\underline{a}' = (a'_0, \ldots, a'_{n-1}) = WTA(w_0 \cdot a_0, \ldots, w_{n-1} \cdot a_{n-1})$ denote the output vector of a winner-take-all gate with the inputs $a_0, \ldots, a_{n-1}$ weighted by integer weights $w_0, \ldots, w_{n-1}$. Set the weights of the winner-take-all gate such that:

$$\underline{a}' = WTA((n+1) \cdot a_0, n \cdot a_1, (n-1) \cdot a_2, \ldots, 2 \cdot a_{n-1}, 1).$$

If $\underline{a} = (0, \ldots, 0)$, a'_n wins (*i.e.* a'_n is the only non-zero output of the gate). Otherwise, a'_i wins if and only if $i = min\{j | a_j = 1\}$, for $0 \leq i \leq n-1$. Furthermore, set the weights of the second winner-take-all gate such that:

$$\underline{b}' = WTA(2 \cdot b_0, 3 \cdot b_1, 4 \cdot b_2, \ldots, (n+1) \cdot b_{n-1}, 1).$$

If $\underline{b} = (0, \ldots, 0)$, b'_n wins. Otherwise, b_i wins if and only if $i = max\{j | b_j = 1\}$, for $0 \leq i \leq n-1$. A simple threshold gate with $\underline{a}'$ and $\underline{b}'$ as its inputs can be used to compute the value of $P^n_{LR}(\underline{a}, \underline{b})$:

$$P^n_{LR} = \begin{cases} 1, & \text{if } \sum_{i=0}^{n-1} (b'_i \cdot (i+1) - a'_i \cdot (i+1)) - b'_n \cdot (n+1) - a'_n \cdot (n+1) \geq 1 \\ 0, & \text{otherwise .} \end{cases}$$

If there is no feature a present, a'_n wins and the gate outputs 0. The same holds for the case that no feature b is present. Otherwise, since there is exactly one a'_i

and exactly one b'_j nonzero, if $a'_i = 1$ and $b'_j = 1$ and $i < j$, the weighted sum is above the threshold and the gate outputs 1. The sum is beyond the threshold for $i \geq j$ and the gate outputs 0.

Any gate can be implemented with linear wire length in our model. So the total wire length is $O(n)$. A similar VLSI layout uses linear area. ∎

In contrast to the threshold circuit of Theorem 2.1 just linear size integer weights are needed for this circuit.

3 Discussion

We have shown that the basic pattern recognition problem whether a certain local feature a occurs to the left of some other local feature b can be solved by circuits that require very little total wire length, and hence can potentially be implemented in analog VLSI. Furthermore it was shown that a circuit with $O(\log n)$ threshold gates can solve this problem, and that this number of threshold gates is asymptotically optimal. Finally it was demonstrated that the same problem can be solved more efficiently if winner-take-all gates are employed in addition to a threshold gate. This gives rise to the question which other concrete computational tasks can be carried out more efficiently by circuits that use winner-take-all gates besides (or instead of) threshold gates.

References

DasGupta, B., Schnitger G. (1996). Analog versus Discrete Neural Networks. *Neural Computation*, 8:819–842.

Lazzaro, J., Ryckebusch, S., Mahowald, M. A., Mead, C. A. (1989). Winner-take-all networks of $O(n)$ complexity. *Advances in Neural Information Processing Systems*, vol. 1, Morgan Kaufmann (San Mateo), 703–711.

Legenstein, R., Maass, W. (2001). Foundations for a circuit complexity theory of sensory processing. *Advances in Neural Information Processing Systems*, vol. 13, MIT Press, 259–265.

Legenstein, R., Maass, W. (2001b). Total Wire Length as a Salient Circuit Complexity Measure for Sensory Processing. Submitted for publication.

Maass, W. (2000). On the computational power of winner-take-all. *Neural Computation*, 12(11):2519–2536.

Mead, C. (1989). *Analog VLSI and Neural Systems*. Addison-Wesley (Reading, MA, USA).

Savage, J. E. (1998). *Models of Computation: Exploring the Power of Computing.* Addison-Wesley (Reading, MA, USA).

Thorpe, S., Fize, D., Marlot, C. (1996). Speed of processing in the human visual system. *Nature*, 381, 520–522.