Neural Networks, Vol. 5, pp. 971–977, 1992
Printed in the USA. All rights reserved.

0893-6080/92 $5.00 + .00

ORIGINAL CONTRIBUTION

Classes of Feedforward Neural Networks and Their Circuit Complexity

JOHN S. SHAWE-TAYLOR,[1] MARTIN H. G. ANTHONY,[2] AND WALTER KERN[3]

[1]Royal Holloway, University of London, [2]London School of Economics, University of London, and [3]University of Twente

(*Received* 12 *April* 1991; *accepted* 30 *March* 1992)

Abstract—*This paper aims to place neural networks in the context of boolean circuit complexity. We define appropriate classes of feedforward neural networks with specified fan-in, accuracy of computation and depth and using techniques of communication complexity proceed to show that the classes fit into a well-studied hierarchy of boolean circuits. Results cover both classes of sigmoid activation function networks and linear threshold networks. This provides a much needed theoretical basis for the study of the computational power of feedforward neural networks.*

Keywords—Neural networks, Boolean circuits, Circuit complexity, Communication complexity, Fan-in, Threshold networks, Feedforward, Circuit depth.

1. INTRODUCTION

There has been a great deal of recent research focussed on neural networks as a promising approach to artificial intelligence. The main attraction has been the possibility of using training instead of explicit programming in order to obtain the required performance of a system. The degree to which explicit programming is excluded varies from cases where a randomly generated network is presented with a representative sample of input/output pairs, to those in which some knowledge of the problem domain allows the design of network to be tailored to simplify the training complexity.

The other attraction of neural networks is their potential for highly parallel implementation. This possibility has been explored in the design of practical hardware systems but has not received much theoretical analysis. The purpose of this paper is to address this aspect of neural networks by linking them with the much studied boolean circuits which are the most natural theoretical model of massively parallel implementation.

In order to show how the classes of neural networks fit into the circuit complexity hierarchy, we must first consider them as boolean functions. This involves restricting the inputs to $\{0, 1\}$ vectors and thresholding the output of the whole network. Note that we allow real values for the activation of hidden nodes and we are not considering just simple linear threshold networks, though these networks do form subclasses of the classes we study. So that we may model the computation in our networks asymptotically, we need to put bounds on the maximum fan-in to nodes, the number of bits of accuracy used in storing the weight and activation values, and on the depth of the network. As with standard boolean circuit theory, these bounds are made in terms of the number of inputs to the network.

For each natural number k we define the class NN_k of problems to be those that can be solved by neural networks with monotonic activation functions (not all necessarily the same function) at the nodes, b bits of accuracy used in representing the weights and activation values, Δ maximum fan-in and height h, where $\log(\Delta) = O(\sqrt{\log n})$, $b \log \Delta = O(\log n)$ and $h \log(\Delta) = O(\log^k n)$, where $\log^k x$ denotes $(\log x)^k$. We show that for all k,

$$NC_k \subseteq NN_k \subseteq AC_k.$$

The paper is organized as follows. Section 2 introduces the terminology and definitions we require. These include circuit complexity and communication complexity. Section 3 gives a number of background results which will be used later. Included is a proof of the well-known equivalence between communication complexity and circuit depth. Section 4 shows how a neural network is converted to an equivalent boolean circuit, while Section 5 completes the complexity analysis and the proof of the main result. We finish with conclusions and directions for further research.

Requests for reprints should be sent to John Shawe-Taylor, Department of Computer Science, Royal Holloway and Bedford New College, University of London, Egham, Surrey TW20 0EX, UK.

2. DEFINITIONS

2.1. Networks

A *network* is a directed acyclic graph. The nodes of in-degree 0 are called *inputs*, and are labelled with a variable x_i or with a constant 0 or 1. The nodes of in-degree $k > 0$ are called *gates* and are labelled with a function on k inputs. The in-degree of a node is referred to as its *fan-in* and its out-degree as its *fan-out*. One node will be designated as the *output* node. A network specifies a function in a natural way. The *size* of the network is the number of gates and the *depth* is the maximum distance from an input to the output.

A *neural network* is a network for which each edge has an associated weight value. The functions associated with the nodes take the weighted sum of the inputs to the node and pass the result through an *activation function*. The activation function is a monotonically increasing function from the real numbers to the interval $[0, 1]$. Thus to specify the functionality of a neural network, we must specify the weights and the activation functions for each node. The activation function of the output node is a threshold function. By restricting the inputs to boolean values a neural network determines a boolean function, see McClelland and Rumelhart (1986).

A *boolean circuit* is a network for which the functions associated with the gates are the boolean functions AND or OR. We will also allow negation to appear on inputs. That is, each input will be paired with its negation. All circuits containing only AND, OR and NOT gates can be transformed into a circuit of the type we consider which is at most twice the size and has the same depth. This result is mentioned in Boppana and Sipser (1990) and a proof may be obtained along the lines of our Proposition 4.1. A boolean circuit is termed *binary* if each gate has two inputs. A boolean circuit represents a boolean function.

2.2. Complexity

Let N denote the natural numbers, $\{0, 1\}^n$ the set of binary strings of length n, and $\{0, 1\}^*$ the set of all finite binary strings. Let $f: \{0, 1\}^* \to \{0, 1\}$. We say that f is computed by a family of networks if the members of the family are indexed by the natural numbers such that the n-th network has n inputs and computes the function $f|\{0, 1\}^n$. We say that some parameter of the family of networks (e.g., size, fan-in, etc.) has complexity $c = c(n)$ if the n-th network has value $c(n)$ for that parameter. For example the *circuit complexity* of a function is the size complexity of the family of minimal boolean circuits with constant fan-in that compute the function. For more information on boolean circuit complexity we refer the reader to Boppana and Sipser (1990) and Wegener (1987).

The classes NC_i (AC_i) are defined to be those functions with polynomial circuit complexity, which can be computed by a family of boolean circuits of constant (unbounded) fan-in and depth $O(\log^i n)$.

The class NN_i is defined to be those functions which can be computed by a family of polynomially sized neural networks with weights and activation values determined to b bits of accuracy, fan-in equal Δ and depth h, satisfying $\log(\Delta) = O(\sqrt{\log n})$, $b \log \Delta = O(\log n)$ and $h \log(\Delta) = O(\log^i n)$.

2.3. Communication Protocols

Consider a function F of two inputs, $\mathbf{x} = (\mathbf{x}_1, \ldots, \mathbf{x}_n)$ and $\mathbf{y} = (\mathbf{y}_1, \ldots, \mathbf{y}_n)$, which takes values in some finite set. Assume that two computing agents, normally referred to as Alice and Bob, are given the values of $\mathbf{x}$ and $\mathbf{y}$, respectively. They are interested in determining the value of the expression $F(\mathbf{x}, \mathbf{y})$. We may assume that they both have unlimited computing resources and, of course, knowledge of F and ask only how many bits must pass between them in the worst case in order to determine the function value. In fact, we only require that one of the participants determines the value, while the other must know only that the value has been determined. In order to make the messages passed between Alice and Bob meaningful, they must agree beforehand on a system of rules to decide at each stage who should send the next message and what information it contains. Clearly after any particular sequence of bits has been passed between them, they must both know who is to send the next bit, though its value may affect who sends the next bit. This system of rules is called a *communication protocol*.

More generally, F need not be a function; the aim may simply be to determine any one value from a particular set of outputs corresponding to the inputs $\mathbf{x}$, $\mathbf{y}$. The definition of a communication protocol in this case is analogous.

The *complexity* of a protocol is the number of bits communicated in the worst case. The *trivial protocol* consists of either Alice or Bob sending all their bits to the other, thus allowing them to compute the value of the function. For more information on communication complexity the reader is referred to Lovász and Saks (1988).

The Difference Problem for a monotonic boolean function f is the problem in which, given two inputs $\mathbf{a}$ and $\mathbf{b}$ for which $f(\mathbf{a}) = 0$ and $f(\mathbf{b}) = 1$, one has to determine an index i for which $\mathbf{a}_i \neq \mathbf{b}_i$. This is the second type of problem described above, where *any* index i, such that $\mathbf{a}_i$ and $\mathbf{b}_i$ differ, will be a correct output.

In the next section we will present and prove a well-known result which states that a boolean function can be evaluated by a binary boolean circuit of depth t if

and only if the communication complexity of the Difference Problem for f is at most t, see Karchmer and Widgerson (1988).

3. PRELIMINARY RESULTS

We first show the trivial inclusion $NC_k \subseteq NN_k$, which follows from showing that we can simulate binary AND and OR gates with bounded fan-in neurons using just one bit of accuracy.

PROPOSITION 3.1. *The class NC_k is a subset of NN_k, for all k.*

PROOF. Given an arbitrary binary boolean circuit, we will convert it to an equivalent neural network with the same number of nodes and depth using binary neurons and one bit of accuracy. The underlying graph is the same as that used for the boolean circuit. The weights on all the lines will be 1. The activation functions for nodes which were AND gates in the boolean circuit will be a threshold function with threshold 1.5, while for an OR gate it will be a threshold function with threshold 0.5. Clearly the neural network computes the same function as the boolean circuit. Hence, if a function f lies in the class NC_k, there are circuits of polynomial size, constant fan-in and depth $O(\log^k n)$ which compute the function. We can therefore find neural networks with constant fan-in, a constant number of bits of accuracy and $O(\log^k n)$ depth which compute the function f. This implies f lies in the class NN_k, as required. ■

Next we consider the equivalence between communication complexity and circuit depth. We will give a proof of this result both for completeness and because we will need the result in the slightly stronger form presented here.

PROPOSITION 3.2. (*Karchmer & Wigderson, 1988*) *Let f be a monotonic boolean function on n inputs, and let $X \subseteq \{0, 1\}^n$. The function f can be evaluated correctly on inputs from X by a circuit of depth t if and only if the communication complexity of the Difference Problem for f on inputs from X is at most t.*

Proof. ($\Rightarrow$) We can assume the existence of a binary boolean circuit of depth t which Alice and Bob will use to direct the protocol. We will also assume that the two inputs to each of the gates have been labelled 0 and 1 in the same way on both Alice's and Bob's copy of the circuit. They use the information passed in the protocol to both trace the same path from the output node to an input which solves the difference problem. We assume inductively that Alice's input gives output 0 at the current node, while Bob's gives output 1. If the node is an AND node, Alice sends the label of an input to the node which also gives output 0. Bob's output at this node must be 1 and so we complete the inductive step. If the node is an OR node then Bob sends the label of an input to the node which also gives output 1. This will again complete the inductive step. After, at most t steps, they must reach an input node which solves the Difference Problem.

($\Leftarrow$) The proof is by induction on t. For $t = 0$ the result is trivial as X and the function f must be such that $f|_X(\mathbf{x}) = \mathbf{x}_i$ for some i. Suppose now that the result holds for values smaller than $t \geq 1$, that $X \subseteq \{0, 1\}^n$ and that the Difference Problem for $f|_X$ has communication complexity at most t. We must consider two cases depending on who sends the first bit in the protocol. As they are symmetrical, we will consider only the case where Alice sends the first bit. We will define two functions g and h which both agree with f on different subsets of X. The function g (h) will agree with f on the set X_g (X_h) composed of the union of $X_1 = f^{-1}(1) \cap X$ and the sets of inputs from X which when given to Alice cause her to send a 0 (1). Note that, however, the functions g and h are defined elsewhere on X, we will have $f = g \wedge h$. We claim that for $g(h)$ the communication complexity of the Difference Problem on inputs from X_g (X_h) is less than t. Both protocols use the protocol for f. This is possible since the two functions agree with f on their respective sets. However, the protocol for f is in both cases shortened by removing the transmission of the first bit. Since this bit is known to be 0 (1) for any input given to Alice from X_g (X_h), this bit does not need to be transmitted for Bob to continue with the protocol and for them both to correctly identify an index on which the inputs differ. By induction there are binary boolean circuits of depth at most $t - 1$ which compute g correctly on X_g and h correctly on X_h. By combining the outputs of these two circuits into one AND gate, we obtain a circuit of depth at most t which computes $g \wedge h$ for some extensions of g and h to X. By the above observation this agrees with f on X. ■

Our reason for considering the above result is in order to perform the following conversion from a linear threshold neuron to an equivalent boolean circuit.

PROPOSITION 3.3. *Let f be the boolean function computed by a linear threshold neuron, with positive weights specified to b bits of accuracy on Δ inputs. There is a boolean circuit of depth* $2 \log \Delta$ with binary OR gates and $2^{b+\log\Delta}$*-input AND gates and size at most*

$$2\Delta^{b+\log\Delta+1},$$

which computes f.

Proof. We first construct a protocol of maximum length $(b + \log \Delta) \log \Delta$, which solves the Difference Problem for the function f. This will imply the existence of a binary boolean circuit computing f and of depth at most $(b + \log \Delta) \log \Delta$. We will then observe that the circuit involves groups of layers all containing AND

gates. These can therefore be concentrated into single multiple input AND gates, reducing the depth significantly, and implying the required result.

The protocol assumes that Alice has been given an input $\mathbf{x}$ for which $f(\mathbf{x}) = 0$, while Bob has an input $\mathbf{y}$ satisfying $f(\mathbf{y}) = 1$. Bob and Alice agree to numbering the inputs to the neuron. The protocol involves bisecting the set of inputs always retaining a set for which Alice's weighted sum is less than Bob's. This is true at the outset for the complete set of inputs, while it is true of a single input if and only if that input is a solution to the Difference Problem. At each stage Alice computes her weighted sum of the first half of the current set and transmits this value to Bob using $b + \log \Delta$ bits. Bob computes the equivalent weighted sum for his input. If this is greater than the value transmitted by Alice, he transmits a 0 back indicating that Alice should continue with this as the new set of inputs. Otherwise, he returns a 1 implying that Alice's weighted sum on the other half of the inputs in the current set is less than Bob's and so this should be adopted as the current set. The bisection of the inputs takes $\log \Delta$ interactions, while the number of bits transmitted in a single interaction is $b + \log \Delta + 1$, giving the length of the protocol as $(b + \log \Delta + 1) \log \Delta$. This implies the existence of a binary boolean circuit of depth $(b + \log \Delta + 1) \log \Delta$ which computes the function f. The number of nodes in a binary circuit of depth h is easily seen to be bounded by 2^{h+1}, and therefore this circuit has at most $2\Delta^{b+\log\Delta+1}$ nodes. However, in the construction described in the proof of Proposition 3.2, bits transmitted by Alice translate into AND gates. Hence the transmission of $b + \log \Delta$ bits by Alice translates into a subtree of AND nodes of depth $b + \log \Delta$ with at most $2^{b+\log\Delta}$ inputs. By making these substitutions, we reduce the depth of the circuit significantly, obtaining a network of depth $2 \log \Delta$ with nodes having a fan-in of at most $2^{b+\log\Delta}$. The size of this network is at most the size of the binary circuit above, and is therefore bounded by

$$2\Delta^{b+\log\Delta+1},$$

as required. ■

PROPOSITION 3.4. *Let f be the boolean function computed by a linear threshold neuron, with positive weights specified to $2b$ bits of accuracy on $2^b\Delta$ inputs. Suppose the inputs to the neuron are grouped together in Δ groups of size 2^b, which are each ordered from $1, \ldots, 2^b$. Then there is a boolean circuit of depth $2 \log \Delta + 1$ with binary OR gates and $2^{2b+\log\Delta}$-input AND gates and size less than*

$$(2\Delta)^{2b+\log\Delta+1},$$

which correctly computes f for all binary vectors satisfying the restriction that those inputs in each group which are on are all of lower index than those that are off.

Proof. We again devise a protocol to distinguish inputs vectors, but now only input vectors satisfying the restriction that the on-inputs in each group are all of lower index than the off-inputs. The same protocol is used as before except that Bob and Alice agree to a subdivision of the inputs, which respects the grouping. In this way, after $\log \Delta$ interchanges, they will have identified a group for which Alice's weighted sum is less than Bob's. Since their inputs both respect the restriction, it will be sufficient for Alice to send the number between 1 and 2^b which is the largest index of an on-input in her group. The next input above this one can then be guaranteed to be a solution of the Difference Problem, since it must be off for Alice and on for Bob. Hence, the protocol has only been extended by b bits, all sent by Alice. In the binary circuit, this translates into b extra levels of AND gates, while in the reduced circuit (after collapsing the sub-trees of AND gates) it gives one extra level composed of AND gates with at most 2^b inputs. The depth of the binary circuit is therefore at most

$$(2b + \log \Delta + 1)\log \Delta + b,$$

and therefore the number of nodes in each circuit is at most

$$22^b\Delta^{2b+\log\Delta+1},$$

which is less than

$$(2\Delta)^{2b+\log\Delta+1}.$$

The result follows. ■

4. NETWORK CONVERSION

Having covered the preliminary results that will be needed in the network conversion, we can begin the main part of the reduction proof. We begin with a general neural network with given accuracy, fan-in and depth and convert it into an equivalent boolean circuit whose parameters can be computed from those of the original network. The conversion has three stages. The first is to eliminate negative weights (apart from threshold values) by moving them back through the network. The second stage involves replacing each neuron by 2^b threshold neurons, one for each possible output value of the original neuron. In the third stage we replace the threshold neurons by boolean circuits using the results of the last section. At each stage, the size, depth, and fan-in of the new circuit will be computed from the parameters of the previous stage.

4.1. Eliminating Negative Weights

This subsection shows how negative weights can be removed from a network in much the same way as negations can be moved to convert a general boolean circuit to one containing only AND and OR gates as in-

ternal nodes. The conversion at most doubles the number of nodes while not altering the fan-in or depth. The number of inputs is doubled. In the following proposition, the complement of a number $x \in [0, 1]$ is the number $1 - x$.

PROPOSITION 4.1. *For every neural network with negative weights there is a neural network with no negative weights which computes the same function, with twice as many nodes, and with each input paired with its complement.*

Proof. We prove the result by first duplicating all the nodes in the network in such a way that the pair of each node has a complementary output, whatever the input to the network. This process doubles the number of nodes and weights. We then show that if there are any negative weights they can be replaced by positive ones. Finally, any redundant nodes can be deleted.

To duplicate a non-input node v, we make a copy v' of v, which is connected to exactly the same nodes as v, but with weights of opposite sign. Furthermore, the activation function f' of v' is related to the activation function f of v by

$$f'(x) = 1 - f(-x).$$

Note that f' is as required a monotonically increasing function from the real numbers to the interval $[0, 1]$. Hence, if the output o_v of v is given by

$$o_v = f\left(\sum_{u:u\sim v} o_u w_{vu}\right),$$

then the output of v' is given by

$$\begin{aligned} o_{v'} &= f'\left(\sum_{u:u\sim v} -w_{vu}o_u\right) \\ &= 1 - f\left(\sum_{u:u\sim v} o_u w_{vu}\right) \\ &= 1 - o_v, \end{aligned}$$

as required.

We now show how to remove a negative weight. Let v be connected to z via a negative weight. In order to remove the negative weight w_{zv} we replace the $v \sim z$ connection with a $v' \sim z$ connection from v's complement to z having weight $-w_{zv}$. In addition, we change the activation function by shifting the argument of the function by the constant w_{zv}. Hence, the new output of z is

$$\begin{aligned} &f\left(\sum_{u:u\sim z, u\neq v} o_u w_{zu} - w_{zv}o_{v'} + w_{zv}\right) \\ &= f\left(\sum_{u:u\sim z, u\neq v} o_u w_{zu} - w_{zv}(1 - o_v) + w_{zv}\right) \\ &= f\left(\sum_{u:u\sim z, u\neq v} o_u w_{zu} + w_{zv}o_v\right) \\ &= o_z. \end{aligned}$$

In this way we can remove all the negative weights in the network. This completes the proof. ■

In view of this result, we assume from now on that all weights are positive. This will not affect our complexity results, since each neural network can be converted to one with positive weights in which the number of nodes is increased by at most a constant factor 2 and the maximum fan-in is unchanged.

4.2. Converting to Boolean Circuit

We first convert a neural network to an equivalent Linear Threshold Network, before completing the conversion to a boolean circuit using Proposition 3.4.

To convert a neural network to a Linear Threshold Network we first take 2^b copies of each of the internal (hidden) nodes, where b is the number of bits of accuracy. With b bits there are at most 2^b different values which can be represented. Let these values be $t_1, t_2, \ldots, t_{2^b}$ in increasing order between 0 and 1. The output of the i-th copy will be 1 if and only if the output of the original network node was greater than or equal to t_i. The node takes input from the 2^b copies of the nodes corresponding to those connected to it in the original neural network. The weight on the i-th line will be $(t_i - t_{i-1})w$, where w is the weight on the corresponding line in the neural network. Assuming the difference between two denotable values is also denotable, this implies that to denote the new weight will require $2b$ bits. The net input to each copy of the node is therefore

$$\sum_{u:u\sim v}\sum_{i=1}^{2^b}(t_i - t_{i-1})w_{vu}o_u^i = \sum_{u:u\sim v} w_{vu}o_u,$$

where o_u^i is the output of u threshold at t_i. Here, f_u is the activation function at node u in the neural network. Input lines from input nodes do not need to be duplicated and their weights do not need to be changed, since the inputs are already binary. The output node does not need to be duplicated since it is by definition a threshold node. Hence, the new network computes the same boolean function as the original neural network. The new linear threshold network has the same depth as the original network, the number of nodes has been increased by a factor 2^b, the fan-in has likewise increased by this factor, while the number of bits has doubled.

There is, however, one important property of the network. If, for a particular node in the original network, the node corresponding to output at least t_i is switched on for a particular input, then the nodes corresponding to lower denotable values will also be on.

We use Proposition 3.4 to convert each of the individual linear threshold neurons of the network created above. Note that, by the observation, the restriction required by the proposition is satisfied. Hence, for a neural network of depth h, maximum fan-in Δ and

accuracy of b bits, we obtain an equivalent boolean circuit with depth $h(2 \log \Delta + 1)$ with binary OR gates and $2^{2h+\log\Delta}$-input AND gates and size at most

$$N2^b(2\Delta)^{2h+\log\Delta+1},$$

where N is the number of nodes in the original neural network.

5. COMPLEXITY RESULTS

We are now ready to prove our main result.

THEOREM 5.1. *For each natural number k,*

$$NN_k \subseteq AC_k.$$

Proof. Let g by a function in $N N_k$. Then for each n there is a neural network which computes $g|\{0,1\}^n$ and has size $N(n)$, height $h(n)$, fan-in $\Delta(n)$ and $b(n)$ bits of accuracy such that $N(n) = O(\text{poly}(n))$, $\log(\Delta(n)) = O(\sqrt{\log n})$, $b(n) \log \Delta(n = O(\log n)$ and $h(n) \log(\Delta(n)) = O(\log^k n)$. By the above we can convert the neural network into a binary circuit of size

$$N(n)2^{b(n)}(2\Delta(n))^{2b(n)+\log\Delta(n)+1}.$$

Taking logarithms of this expression gives:

$$\log N(n) + b(n) + (2b(n) + \log\Delta(n) + 1) \times (\log\Delta(n) + 1) = O(\log n).$$

Hence, the circuit is polynomially sized. The depth of the circuit is

$$h(n)(2 \log\Delta(n) + 1) = O(\log^k n).$$

This implies that the function g lies in AC_k as required. ■

These results concern the exact representation, as boolean circuits, of neural networks with weights and outputs to a fixed accuracy of b bits. This can be regarded as in some sense approximating unlimited accuracy neural networks with boolean circuits, where the degree of approximation is determined by b. However, Raghavan (1988) has shown that a single linear threshold neuron with boolean inputs, unlimited accuracy on its weights, and fan-in Δ can be represented exactly by a linear threshold neuron with the same fan-in, but with weights constrained to be $\Delta \log \Delta$-bit integers. That is, the neuron can be replaced by one which computes the same function, has the same fan-in, but has bounded accuracy $\Delta \log \Delta$. Using this, we can obtain the following exact representation theorem for linear threshold networks.

THEOREM 5.2. *Suppose that a linear threshold feedforward neural network with n boolean inputs has height h, N nodes and maximum degree Δ. Then there is a boolean circuit with n inputs, depth $2h \log \Delta$, binary OR gates, AND gates of fan-in at most $\Delta^{2\Delta}$ and size at most*

$$2N\Delta^{\Delta\log\Delta+\log\Delta+1}$$

such that the circuit computes the same function of its inputs as does the neural network. ■

This leads to the following complexity result for sparse linear threshold networks.

THEOREM 5.3. *Let $0 < \epsilon < 1$ and let TN_k^i be the class of boolean functions $f: \{0, 1\}^* \to \{0, 1\}$ such that $f|\{0,1\}^n$ can be computed by a polynomially-sized feedforward linear threshold network with maximum fan-in $\Delta = O((\log n)^{1-\epsilon})$ and height h such that $h \log \Delta = O(\log^k n)$. Then*

$$NC_k \subseteq TN_k^i \subseteq AC_k.$$

Proof. The first containment follows as earlier, in the proof that $NC_k \subseteq NN_k$. For the second containment, we use the preceeding result. Let g be a function in $T N_k^i$. Then for each n there is a feedforward linear threshold network which computes $g|\{0,1\}^n$, with size $N(n)$, height $h(n)$ and fan-in $\Delta(n)$ such that $\Delta(n) = O((\log n)^{1-\epsilon})$ and $h(n)\log\Delta(n) = O(\log^k n)$. Now, by the previous result, we can convert this threshold network into a boolean circuit of size at most

$$2N(n)\Delta^{\Delta\log\Delta+\log\Delta+1}$$

and depth at most $2h \log \Delta$. Taking logarithms, we get

$$1 + \log N(n) + (\Delta \log \Delta + \log \Delta + 1)\log \Delta = O((\log n)^{1-\epsilon}\log\log n) = O(\log n).$$

Hence, the circuit has polynomial size. As before, the depth of the circuit is

$$2h \log \Delta = O(\log^k n).$$

Therefore g lies in AC_k, as required. ■

6. CONCLUSIONS

We have introduced a hierarchy of classes of neural networks and shown that they interleave the well-known boolean complexity classes NC_k and AC_k. The neural network class introduced has two significant limitations. The first is in the number of bits used to represent the real numbers involved and the second is in the fan-in to the nodes.

The limitation on the fan-in is sublinear in the number of inputs, but significantly larger than logarithmic. This appears to be a critical limitation on the computational power of the network.

The limitation on the number of bits is at best logarithmic, but decreases to the square root of the logarithm as maximum fan-in is used. This appears to be a fairly severe limitation on the expressibility of the numbers involved. This would seem a reasonable limitation not only for standard computing equipment but also for the biological neural networks, where synapse

accuracy does not appear to be very great or increase dramatically in more advanced warm blooded species.

There are several indications why the limitation on the number of bits is perhaps not as severe as we might suppose. The first is that in the proof that $NC_k \subseteq NN_k$, we require only one bit of accuracy to represent the Boolean circuit, leaving $O(\log n)$ bits in "reserve" in this constant fan-in example. More importantly, Theorem 5.3 shows that at least in the case of linear threshold circuits with fan-in still further restricted, the expressive power is no longer affected by increasing the number of bits indefinitely. It is an open question whether there is a restriction on the fan-in which would make the computational power of a neural network independent of the accuracy of the numbers involved, but in view of Theorem 5.3, this certainly does not seem an impossibility.

Our conclusions are therefore that, as in the case of threshold circuits, it appears to be the large fan-in nodes which significantly increase the computational power of neural networks rather than the detailed functions computed at a node. This is reinforced by our allowing any monotonic activation functions to be used in internal nodes in the classes discussed. We feel that to implement neural networks in hardware, more emphasis should perhaps be placed on processing large fan-ins in parallel rather than modelling traditional activation functions exactly.

REFERENCES

Boppana, R. B., & Sipser, M. (1990). The complexity of finite functions. In *Handbook of Theoretical Computer Science* (pp. 758–804). Elsevier Science Publishers.

Karchmer, M., & Widgerson, A. (1988). Monotone circuits for connectivity require super-logarithmic depth. *Proceedings 20th ACM Symposium on Theory of Computing*, 539–550.

Lovász, L., & Saks, M. (1988). Lattices, Möbius functions and communication complexity. *Proceedings 29th IEEE FOCS*, 81–90.

McClelland, J. L., & Rumelhart, D. (1986). *Parallel distributed processing, Vols. 1 and 2*. Cambridge, MA: MIT Press.

Raghavan, P. (1988). Learning in threshold networks. *Proceedings of 1988 Workshop on Computational Learning Theory*. San Mateo, CA: Morgan Kaufmann, 19–27.

Wegener, I. (1987). *The complexity of boolean functions*. Wiley-Teubner Series in Computer Science. Stuttgart: Teubner; Chichester: Wiley.