Piecewise linear approximation applied to nonlinear function of a neural network

H.Amin
K.M.Curtis
B.R.Hayes-Gill

Indexing terms: Nonlinear approximation, Neural networks, Sigmoid function

Abstract: An efficient piecewise linear approximation of a nonlinear function (PLAN) is proposed. This uses a simple digital gate design to perform a direct transformation from X to Y, where X is the input and Y is the approximated sigmoidal output. This PLAN is then used within the outputs of an artificial neural network to perform the nonlinear approximation. The comparison of this technique with two other sigmoidal approximation techniques for digital circuits is presented and the results show that the fast and compact digital circuit proposed produces the closest approximation to the sigmoid function. The hardware implementation of PLAN has been verified by a VHDL simulation with Mentor Graphics running under the UNIX operating system.

1 Introduction

One of the important computational elements within a multilayer perceptron neural network is the nonlinear activation function [1, 2]. A commonly used type of nonlinear function is the sigmoid function which has the advantage of having a simple differential equation

$$y = \frac{1}{1 + e^{-x}} \tag{1}$$

$$\frac{dy}{dx} = y(1 - y) \tag{2}$$

Sigmoidal functions can easily be implemented in analogue circuits owing to the nonlinear characteristics inherent in CMOS devices. In digital implementations the sigmoidal approximation can be obtained from either a lookup table or a direct calculation [3]; for the latter, two different designs have been included in this paper for comparison with our proposed technique using a quadratic equation function [4], and using an A-law companding technique [5] based on the piecewise linear approximation technique.

In the lookup-table approach the y-value associated with each x-value is stored in memory and x is used as an address to the lookup table. Although this technique is simple to implement, it requires a large silicon area especially when higher precision is needed and hence it would be impractical to include one lookup table per processing element in a massively parallel neural network. In the case of the direct calculation approach, a quadratic equation is used to approximate the nonlinearity, with the expense of having to process two multiplications (i.e. a gain factor is multiplied by a squared value) and two additions. In the case of piecewise linear approximation the breakpoints of the curve are stored in memory, and linear interpolation is applied between the points to obtain the sigmoid function. In the third approach, an A-law companding technique for pulse code modulation is used to implement the piecewise linear approximation.

We present an efficient piecewise linear approximation technique where the multiply/add operations of the linear interpolation are replaced by a simple gate design, which leads to a very small and fast digital approximation of the sigmoid function. A comparison of the performances of the new technique with methods described in [4, 5] during the backpropagation learning process is presented.

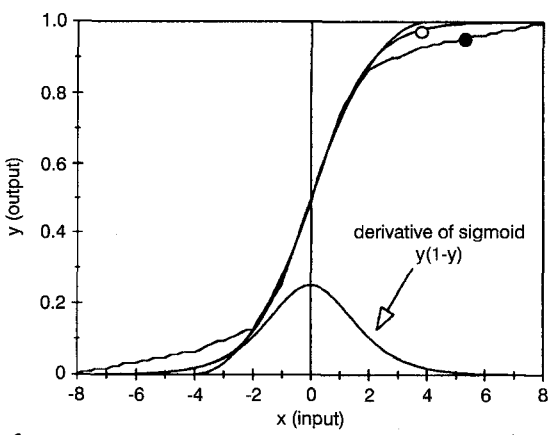

Fig. 1 *Sigmoidal approximation using A-law companding technique and quadratic equations*
——— quadratic equation
—○— sigmoid
—●— A-law

2 Approximation methods

Fig. 1 shows the sigmoid curve together with its approximations [4, 5]. As can be seen, the A-law technique does not give a good approximation to the sigmoid function when the input is either greater than 2

IEE Proceedings online no. 19971587
Paper received 23rd April 1997

The authors are with the Department of Electronic and Electrical Engineering, University of Nottingham, University Park, Nottingham NG7 2RD, UK

or less than –2. This can lead to a longer convergence time for the network because the gradient of the curve (i.e. line) is smaller than that of the sigmoidal nonlinear curve in that region.

For the quadratic equation technique the gradient of the curve is higher around a *low limit* (i.e. –4.04) and a *high limit* (i.e. 4.04), when compared with the sigmoid function. Although this would result in a faster convergence time for the network it can lead to it not being able to exit from a local minima since the derivative drops to zero above and below the *high limit* and *low limit*, respectively; in other words, the weights of the neurons entering such regions would not be updated. To overcome this problem the *low limit* and *high limit* range can be increased; this, however, could lead to a slower convergence of the network because the gradient of the curve would be reduced.

The piecewise linear technique, proposed here and plotted in Fig. 2, gives the closest approximation to the sigmoid function. Since the gradient of the curve (i.e. line) when the input is either between 2.375 to 5 and –2.375 to –5 is slightly lower than that of the sigmoid curve the convergence time for the network will be slightly longer, but much shorter than the A-law technique.

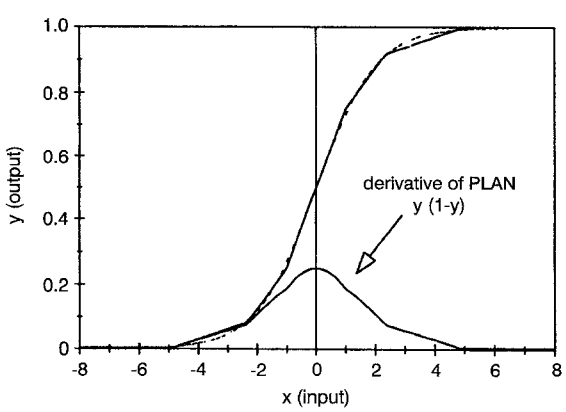

Fig. 2 *Sigmoidal approximation using piecewise linear technique*
- - - - - sigmoid
——— PLAN

Our proposed new implementation of the PLAN approximating technique is simple to implement in digital circuits as the complexity of the computations can be significantly reduced by choosing the gradient of the lines so as to eliminate the need for multipliers.

Furthermore, a modification to the derivative of the output was made [6] to ensure that the network would be able to leave local minima; i.e. derivative = ε, when the calculated derivative is less than ε, where ε is a small value equal to the derivative of the output of a neuron that is close to the solution (i.e. either 0 or 1), for a neural network that uses a sigmoid function. The value of 0.005 for ε was used in the simulations presented in the results Section.

This modification would prevent the neurons from becoming inactive in the presence of local minima since the derivative would not be zero at any time but would be small enough not to make the network unstable in the vicinity of the solution. This method is particularly useful for the quadratic equation and the PLAN techniques because, for these cases, the neurons reach the saturation for input values greater than 4.04 and 5, respectively, when compared with the sigmoid function (Figs. 1 and 2).

3 Piecewise linear approximation of nonlinearities

Fig. 2 shows an approximation to the sigmoid function using the PLAN technique. Since the graph is symmetrical about the line $Y = 0.5$ [3] (or when $X = 0$), the piecewise calculations need only be performed on the absolute value of the input $/X/$ to obtain the output value Y for both regions; that is, for the negative input values the calculated Y is subtracted from 1 to give its mirror image. A piecewise representation of the sigmoid function is given in the following Table.

Here, the Zs are flags used in the hardware implementation to indicate which range the input belongs to. Each output, in this case, represents a straight line between two breakpoints. The significance of this piecewise representation is that the gradient of the lines are chosen such that these multiplications can be replaced by simple shift operations; for example, if $X = 2$, then X is shifted 3 times to the right ($0.125_{10} = 0.001_2$) and then added to 0.625 (0.101_2), and for $X = -2$ the same result is subtracted from 1. These shift and add operations, however, can be totally removed and replaced with a simple logic design by performing a direct transformation from input to sigmoidal output (i.e. DTXY).

Table 1: Implementation of PLAN

Operation	Condition	Flags			
		Z_1	Z_2	Z_3	Z_4
$Y = 1$	$/X/ >= 5$	0	0	0	1
$Y = 0.03125 * /X/ + 0.84375$	$2.375 =< /X/ < 5$	0	0	1	0
$Y = 0.125 * /X/ + 0.625$	$1 =< /X/ < 2.375$	0	1	0	0
$Y = 0.25 * /X/ + 0.5$	$0 =< /X/ < 1$	1	0	0	0
$Y = 1 - Y$	$X < 0$				

4 Implementation of PLAN using DTXY

Fig. 3 shows a diagram of our new digital implementation of PLAN which realises the operation in Table 1. The input is first two's complemented if it is a negative value, then a magnitude comparator is used to set the flags according to Table 1. Based on these flags the appropriate operation will be performed in the DTXY component as indicated in Fig. 3. The result will be subtracted from 1, and the final value Y(sig) will be equal to either Y or $1 - Y$ according to the sign bit of the input. The last equation, $1 - Y$ is a necessary operation since it is needed to obtain the derivative of the output nodes (i.e. eqn. 2).

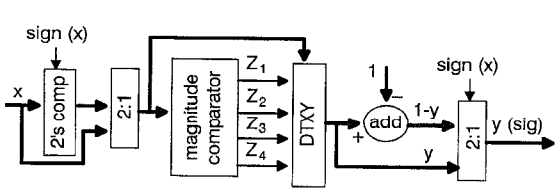

Fig. 3 *Digital architecture of PLAN using DTXY*

4.1 Magnitude comparator

Fig. 4 shows the implementation of the magnitude comparator, where on the left-hand side the ranges of $/X/$ (i.e. column 2 of Table 1) are given both in real and binary values. As can be seen, the digital

implementation is small and requires no storage of the breakpoints.

input /x/ register		
real value	binary value sign A B C D. E F G ...	
0.0	0 0 0 0 0. 0 0 0 ...	
0.99	0 0 0 0 0. 1 1 1 ...	} Z_1
1.0	0 0 0 0 1. 0 0 0 ...	
2.3749	0 0 0 1 0. 0 1 0 ...	} Z_2
2.375	0 0 0 1 0. 0 1 1 ...	
4.9	0 0 1 0 0. 1 1 1 ...	} Z_3
5.0	0 0 1 0 1. 0 0 0 ...	
15.9	0 1 1 1 1. 1 1 1 ...	} Z_4

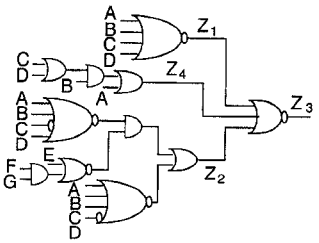

Fig. 4 *Implementation of magnitude comparator*

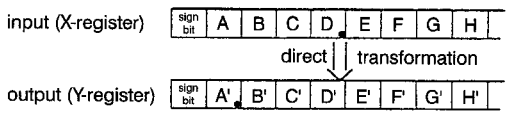

Fig. 5 *Direct tranformation from X-register to Y-register*
The capital letter indicates the bits in the registers and the dot indicates where the decimal point is placed

4.2 Direct transformation

Based on Table 1 the system must perform a shift operation on the X-register where the input values are moved by two, three or five times to the right, depending on which range the input belongs to. This means that if a shift register is used the system would require five clock cycles for the worst case, excluding the addition operation. These operations however can be removed altogether by applying a direct transformation from the X-register to the Y-register (see Fig. 5). The aim is to discover and recreate regularities that occur among bits of the X-register (the input value) and the Y-register (the output value) when the shift/add operation is performed on the input to get the approximated sigmoidal output. This is shown in the following Sections where the direct transformation is obtained for each range separately and then they are combined to get the transformation of all ranges in a single design.

4.2.1 First range (0 = < /X/ < 1, Z1 = 1, Y = 0.25* /X/ + 0.5): X is shifted twice to the right and added to $0.5_{10}(0.1_2)$. The maximum value for X in this range is $0.99.._{10}$ $(0.11.._2)$ and when it is shifted twice it would give $0.0011.._2$, and adding 0.1_2 gives $Y = 0.1011.._2$. Therefore,

$$A' = 0, \quad B' = 1, \quad C' = 0,$$
$$D' = E, \quad E' = F, \quad F' = G, \ldots$$

note that the capital letters with the prime sign are bits of the y-register, as indicated earlier.

4.2.2 Second range (1 = < /X/ < 2.375, Z_2 = 1, Y = 0.125 * /X/ + 0.625): Here X is shifted three places to the right and added to $0.625_{10}(0.101_2)$; in this case the value before the decimal point would be either 1 or 2, and thus both are considered.

	ABCD.EFGH
1.9375 $\longrightarrow$	0001.1111
$1.9375 * 0.125 \longrightarrow$	0.0011111+
add to 0.625 $\longrightarrow$	0.1010000
Y $\longrightarrow$	0.1101111
	ABCD.EFGH
2.3125 $\longrightarrow$	0010.0101
$2.3125 * 0.125 \longrightarrow$	0.0100101+
add to 0.625 $\longrightarrow$	0.1010000
Y $\longrightarrow$	0.1110101

For both cases the first bit after the decimal point in the y-register is always 1 (B′ = 1), this is because the corresponding bits for the shifted X and 0.625 are 0 and 1, respectively, and no carry would be produced from the next bits. The second bit after the decimal point is also 1 (C′ = 1). Only the third bit after the decimal point (i.e. D′) would be changed, which is the same as bit C in the X-register, while the remaining bits will be the same as the corresponding bits in the X-register; therefore

$$A' = 0, \quad B' = 1, \quad C' = 1, \quad D' = C,$$
$$E' = E, \quad F' = F, \quad G' = G, \ldots$$

4.2.3 Third range (2.375 = < /x/ < 5, Z3 = 1, Y = 0.03125* /X/ + 0.84375): A similar analogy can be drawn for this region which gives the following:

$$A' = 0, \quad B' = 1, \quad C' = 1, \quad D' = 1,$$
$$E' = (C \text{ XNOR } D), \quad F' = \text{NOT } D,$$
$$G' = E \quad H' = F, \quad I' = G, \ldots$$

where NOT indicates an inverse operation.

4.2.4 Fourth range (/X/ > = 5, Z4 = 1, Y = 1): In this range all the bits in Y-register are 0 except A′ = 1.

4.3 Final design

Fig. 5 shows the final design of the direct transformation from X to Y (i.e. DTXY) obtained from the preceding analysis. Referring to Fig. 3, the propagation delay for obtaining Y(sig) using DTXY is

$$t_{PLAN} = 2t_{add} + t_{mc} + t_{dt} + 2t_{mux}$$

where t_{mux} is a two gate delay required for a 2:1 multiplexer and t_{mc} and t_{dt} are the propagation delay of the magnitude comaparator and the direct transformation, respectively; these last two operations give a total of seven gate delays. There are two delays for the addition operation, one is the two's complement operation and the other is the calculation of $1 - Y$. These addition operations can be performed by either employing an adder that is resident in every neuron or using separate adders that can be constructed within the PLAN. The first approach is generally more desirable as it results in a more efficient design in terms of silicon area with the price of increasing the complexity of the control system. If this approach is used t_{add} would be equal to one clock cycle, otherwise t_{add} would depend on what type of adder is used. For example, for a ripple carry parallel adder $t_{add} = 2Nt_g$, where t_g is one gate delay and N is the number of bits of inputs.

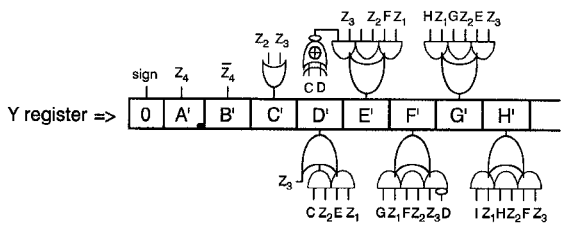

Fig. 6 *Hardware implementation of direct transformation from /X/ to Y register*

The following Section describes how the last addition operation (i.e. $1 - y$) can be eliminated, which would result in an even less complex and faster design.

5 Faster Implementation of PLAN using DTXY

Further simplification can be applied to eliminate the 'add', in Fig. 3, by obtaining the 1 - Y value separately and in parallel with the operation that leads to obtaining the Y register as shown in Fig. 6. In this case, Y and 1 - Y are obtained from $+X$ and $-X$, respectively. The 'DTXY for y' is the same as the 'DTXY' in Figs. 3 and 6; while 'DTXY for 1-y' can be implemented in the same manner as for Fig. 6, the result is given in Fig. 8. Here small letters are the bits of the two's-complemented X-register and primed small letters correspond to the bits of the 1 - Y register. Note that the same flags, generated from the circuit in Fig. 4, are used for both operations and the propagation delay for $t_{mc} + t_{dt}$ will still be equal to six gate delays. The total delay will then be

$$t_{PLAN} = t_{add} + t_{mc} + t_{dt} + 2t_{mux}$$

Once the input is two's complemented, Y(sig) would be obtained within a 11 gate delay (i.e. $t_{mc} + t_{dt} + 2t_{mux}$).

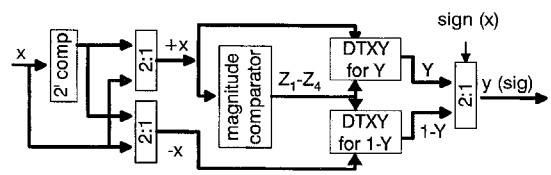

Fig. 7 *Digital architecture of PLAN using DTXY*

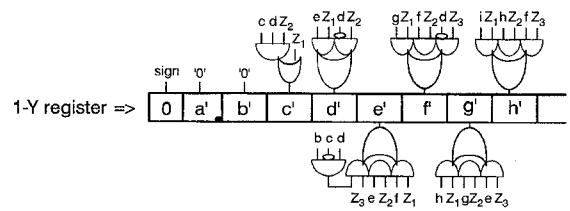

Fig. 8 *Hardware implementation of direct transformation from –X to 1–Y register i.e. DTXY for 1-Y*

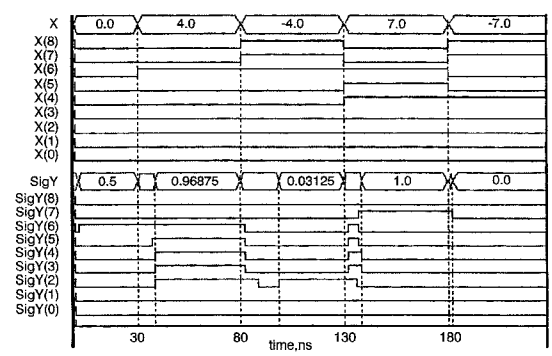

Fig. 9 *Simulation results of VHDL models for PLAN created using Mentor Graphics*
Decimal point for input values is at X (4) and for approximated sigmoid values at Sig Y (7)

6 VHDL simulation

A VHDL simulation tool [7] running under Mentor Graphics was used to test PLAN. First the design was described at a purely behavioural level, according to the conditions in Table 1. Then the design in Fig. 3 was simulated, where the magnitude comparator and the direct transformation blocks were described at the structural gate level and the rest at the behavioural level. Finally the design of Fig. 7 was simulated at a purely structural gate level. All the simulations were successful and produced the same results. Fig. 9 shows the simulation results of the VHDL structual model for PLAN; the real values of X and Sig Y in the graph were added manually to indicate their values at different times. Note that the decimal point for X is located after bit $X(4)$ and for Sig Y after Sig $Y(7)$ and the bit 8 represents the sign bit for both cases. The delay of 1 ns was arbitrarily assigned for each gate and a 10 ns delay was chosen for the two's complement operation (i.e. adder). The worst-case delay for the PLAN was found to be 21 ns (i.e. when $X = -4.0$), of which 10 ns belongs to t_{add} and 11 ns is the result of $t_{mc} + t_{dt} + 2t_{mux}$. This result is in agreement with the previous analysis for t_{PLAN}.

7 Results

All the three nonlinearity approximating techniques were used in the backpropagation learning process to solve three different problems: parity check, decoder problem, and handwritten character recognition. These problems were chosen as platforms for comparison.

7.1 Parity check problem

In this problem a 3-bit and 4-bit parity check were performed by a neural network using different nonlinearity approximation techniques. The simulation results showed that the network whose nonlinearities were approximated by PLAN (Fig. 3) gave the closest result (i.e. a comparable number of iterations) to the networks which used the sigmoidal function. The results are shown in Table 2. The learning rate and the momentum term were 0.3 and 0.7, respectively, and the network was stopped when the preset error was below 0.001, or when the number of iterations reached 6000. The figures in the parenthesis represent the neural network configurations, that is (input: hidden: output).

Table 2: Simulation results showing number of iterations for parity check problems

	Sigmoid	PLAN	A-law	QET1	QET2
Iterations (3:5:1)	1073	1281	1839	780	775
Iterations (4:8:1)	643	960	1667	512	509
Iterations (4:5:1)	1049	1107	1540	failed	3081

From the Table, for some cases the neural networks that used the quadratic equation technique (QET1) converge faster than those that used the sigmoid function. However, as described earlier they fail to learn for more complex problems. The last entry in the QET1 column shows the lack of convergence of the network. Here due to the smaller number of hidden nodes the network could not solve the problem, probably because a critical local minima in the error surface has been created. This problem was solved by applying a modification to the derivative of the output of the neurons, as presented earlier. The last column shows the result of the quadratic equation technique using a modified derivative (QET2).

The performances of neural networks using PLAN and the A-law approximation techniques are also in agreement with the earlier predictions. The modified derivative had no, or little, effect on these networks and hence are not included in the Table.

7.2 Decode problem

This is the problem of mapping a set of orthogonal input patterns to a set of orthogonal output patterns through a small set of hidden nodes which is equal to $\log_2 N$, where N is the number of output (and input) nodes [8]. Table 3 shows the results of simulating two problems using the different nonlinear approximation techniques. The first was an 8-bit decoder problem (i.e. $N = 8$) with three hidden nodes and eight patterns, and the second a 16_bit decoder problem (i.e. $N = 16$) with four hidden nodes and 16 patterns. The learning rate and the momentum term were 0.05 and 0.9, respectively, and the preset error was 0.002.

Table 3: Simulation results showing iterations for decode problem

	Sigmoid	PLAN	A-law	QET1	QET2
Iterations for N=8	5160	6008	9500	5140	5123
Iterations for N=16	3400	3671	4455	failed	4740

The results follow the same trend as the parity check problem. Again it can be seen that PLAN results in an efficient number of iterations during learning.

7.3 Handwritten character recognition

Several handwritten character recognition problems were examined with the neural networks employing the different nonlinear approximating methods, and the result of the simulations is presented in Table 4. In this simulation, the learning rate and the momentum term were 0.05 and 0.9 respectively; and the preset-error was 0.001 and the final iteration was set to 150. There were 260 training patterns (i.e. 10 versions of each handwritten character) and 130 test patterns.

Table 4: Simulation results of handwritten character recognition problem

	Sigmoid	PLAN1	PLAN2	A-law	QET1	QET2
Iterations	63	150	69	93	150	113
Generalisation	85.4%	63.8%	85.4%	84.5%	46%	80.8%

The modified derivative, described earlier, proved to be extremely useful for the networks which used PLAN and QET nonlinearities. In both cases, the efficiencies of the network, in terms of the convergence and generalisation, was improved substantially. These are shown as PLAN2 and QET2, respectively.

8 Conclusion

Three different implementations of the sigmoid function for digital circuits were compared. One of the methods, based on the piecewise linear approximation PLAN, was developed by ourselves and the other two were a quadratic equation technique [4] and an A-law companding technique [5]. In the PLAN method, the sigmoid function was approximated by a combination of straight lines. A substantial reduction in the complexity of computing the nonlinear approximations was made by choosing the gradient of the lines such that all the multiplications were replaced by simple shift operations. This shift/add operation, of the line interpolation, has been implemented with a simple gate design so as to directly map the input values to sigmoidal outputs. This resulted in a very small and fast digital approximation of the sigmoidal function. The system will require only 11 gate delays to perform the entire computation for the sigmoidal approximation on the two's-complemented input. The simulation results showed that, of the three methods, our new approximation technique offered the best performance, since it was the closest to the sigmoid function. The feasibility of the hardware implementation of PLAN was tested using a VHDL simulation, where both behavioural and structural gate models were used to describe the circuit designs. A modification to the derivative of the output of the neurons was applied which eliminated the saturation of the neurons in the presence of local minima.

9 References

1 LIPPMANN, R.P.: 'Introduction to computing with neural nets', *IEEE ASSP Mag.*, 1987, **4**, (2), pp. 4–22
2 HUSH, D.R., and HORNE, B.G.: 'Progress in supervised neural networks: what's new since Lippmann ?', *IEEE Signal Process. Mag.*, 1993, **10**, (1), pp. 8–39
3 NORDSTROM, T., and SVENSON, B.: 'Using and designing massively parallel computers for artificial neural networks', *J. Parallel Distrib. Comput.*, 1992, **14**, pp. 260–285
4 SAMMUT, K.M., and JONES, S.R.: 'Implementing nonlinear activation function in neural network emulatiors', *Electron. Lett.*, 1991, **27**, (12), pp. 1037–1038
5 MAYERS, D.J., and HUTCHINSON, R.A.: 'Efficient implementation of piecewise linear activation function for digital VLSI neural networks', *Electron. Lett.*, 1989, **25**, (24), pp. 1661–1662
6 FAHLMAN, S.E.: 'An imperical study of learning speed in back-propagation networks'. Technical report CMU–CS–88–162, Department of Computer Science, Carnegie Mellon University, Pittsburgh, PA, June 1988
7 LIPSETT, R., SCHAEFER, C., and USSERY, C.: 'VHDL: hardware description and design' (Kluwer, 1989)
8 RUMELHART, D.E., HINTON, G.E., and WILLIAMS, R.J.: 'Learning internal representations by error propagation' *in* RUMELHART, D.E., and McCLELLAND, J.L. (Eds.): 'Parallel distributed processing: explorations in the microstructures of cognition' (MIT press, Cambridge, MA, 1986), pp.318–362