An Efficient Hardware Architecture for a Neural Network Activation Function Generator

Daniel Larkin*, Andrew Kinane, Valentin Muresan, and Noel O'Connor

Centre for Digital Video Processing, Dublin City University, Dublin 9, Ireland.
{larkind,kinanea,muresanv,oconnorn}@eeng.dcu.ie

Abstract. This paper proposes an efficient hardware architecture for an elementary function generator that is suitable for use as the activation function in an artificial neural network (ANN). A spline-based approximation function is designed that provides a good trade-off between accuracy and ...., whilst also being inherently scalable and adaptable to other approxmations. This has been achieved by using a minimax polynomial and through optimisation of the placement of the splines by using a genetic algorithm. The approximation error of the proposed activation function compares favourably to all related research in this field. Multiplication circuitry is avoided through the use of distributed arithmetic thereby reducing the area overhead and power consumption of the hardware implementation. The proposed architecture is up to 2,751 times faster than a direct implementation of a sigmoid activation function on an ARM 9 processor with a comparable clock frequency.

1 Introduction

Artificial neural networks (ANN) have found widespread deployment in a broad spectrum of classification, perception, association and control applications [1]. However, practical ANN implementations for high dimensional data tasks, such as multimedia analysis, computer gaming, etc, create considerable demands for microprocessor cycles. This is due to the fact that such ANN require extremely high throughput, in addition to a large number of inputs, neurons and layers. The associated computational complexity is highly undesirable from real time operation and low power consumption perspectives. This poses considerable problems for constrained computing platforms (e.g. mobile devices) which suffer from limitations such as low computational power, low memory capacity, short battery life and strict miniaturisation requirements. An attractive solution to this is to design systems whereby ANN complexity can be addressed by offloading processing from the host processor to dedicated hardware for general purpose ANN acceleration.

There has been considerable research in both analog and digital hardware ANN implementations – [2][3][4]. Low complexity architectures have been favoured,

* The support of the Enterprise Ireland Informatics Initiative is gratefully acknowledged.

particularly for implementing the activation function. Low complexity approaches have the benefit of allowing reduced silicon area, but this typically comes at the expense of output performance. It is generally accepted that, as the resolution and precision of the inputs, weights and the activation function are reduced, so too is the ability of the ANN to act as a universal approximator [5][6]. However, in an era where large microprocessors now use half a billion transistors, the overall benefit of area savings in the order of tens to hundreds of transistors is questionable, particularly if the output performance and scalability is compromised. This observation has motivated us to design a high precision, power efficient hardware activation function generator, which is capable of accommodating multiple activation functions and their derivatives.

The rest of this paper is organised as follows: section 2 details related prior research in the area. Section 3 proposes the use of a minimax spline approximation scheme. Section 4 outlines an associated hardware architecture for this scheme. Section 5 details hardware synthesis results and power consumption estimates, whilst section 6 draws conclusions about the work presented.

2 Related Research

Given its desirable non linear characteristics and ease of differentiability, a Sigmoid based activation function, such as that defined in eqn. 1, is commonly used in neural networks [7]. However, a direct hardware implementation is not practical as it requires excessive logic, resulting in significant power loss.

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

Consequently, a number of approximations amenable to hardware implementation have been developed. Since a direct look up table (LUT) implementation uses excessive memory, approaches typically fall into the following broad categories: piecewise linear approximations [8][9][10][11][12][13], piecewise second order approximations [11] and combinatorial input/output mappings [14]. Furthermore, there is considerable variance within each category. For example, an A-Law companding technique is used in [8], a sum of steps approximation is used in [9], a multiplier-less piecewise approximation is presented in [10], a recursive piece multiplier-less approximation is presented in [13]. An elementary function generator capable of multiple activation functions using a first and second order polynomial approximation is detailed in [11]. Recently, a combinatorial approach has been suggested that considerably reduces the approximation error [14]. Our approach, which will be describe in detail in section 4, uses a minimax first order polynomial approximation. The use of a minimax polynomial has been suggested before in the context of a floating point activation function approximation [12]. However, we further minimise the maximum error and implement an area and power efficient architecture. Our approach produces a small approximation error, whilst being suitable for implementation of multiple activation functions.

2.1 Data representation and precision requirements

In function approximation there are two sources of error, the approximation method error and the data representation error resulting from the use of a finite number of bits. To minimise area and power consumption, the minimum number of bits should be chosen that results in an acceptable error. Hardware ANN reduced precision issues were previously explored in [5][6]. It was found that 10 precision bits were sufficient for multi-layer perceptrons (MLP) trained via back propagation [6]. Using fewer precision bits than this will effect the convergence speed for on-chip learning, and in some cases may completely prevent convergence. The number of integer bits should be chosen based on the range of the activation function. The functions we propose implementing (see section 5) have a maximum usable input integer range of ± 8 and a maximum usable output range of ± 1. For these reasons, we propose using 14 input bits (4 integer bits and 10 fractional bits) and 12 output bits (2 integer bits and 10 fractional bits). This scheme was also used by [11]. Data representation and precision used in this and other related research is shown in table 1. [DAN: VASSILIADIS SEEM TO HAVE DIFFERENT INPUT/OUTPUR RANGES AND DIFFERENT INTEGER/PRECISION BITS?]

	Input				Output			
	Total Bits	Range	Integer Bits	Fractional Bits	Total Bits	Range	Integer Bits	Fractional Bits
Myers et al [8]	16	[-8,8[	4	12	8	[0,1[	1	7
Alippi et al [9]	not discussed							
Amin et al [10]	8	[-8,8[	4	4	8	[0,1[	1	7
Vassiliadis et al [11]	14	[-4,4[	3	10	14	[0,1[	4	10
Faiedh et al [12]	Single precision floating point							
Basterretxea et al [13]	not discussed							
Tommiska-337 [14]	6	[-8,8[	3	3	7	[0,1[	0	7
Tommiska-336 [14]	6	[-8,8[	3	3	6	[0,1[	0	6
Tommiska-236 [14]	5	[-4,4[	2	3	6	[0,1[	0	6
Tommiska-235 [14]	5	[-4,4[	2	3	5	[0,1[	0	5

Table 1. Data widths and precision used in related research

3 Proposed Approximation Scheme

Polynomial approximating functions, such as a Taylor series (see eqn. 2 for a Taylor series expansion of a sigmoid function), can be used to represent any arbitrary continuous function. To reduce the order of the approximating polynomial, the input domain of the function can be sub-divided into smaller intervals. This allows a polynomial of much lower order to be used to approximate each of the of sub-intervals. The resulting composite function is known as a piecewise polynomial or spline. Using a spline-based activation function approximation offers the benefit that multiple activation functions can be accommodated by

merely changing the coefficients of the approximating polynomials. This implies that minimal extra hardware is required to support these additional functions.

$$Sigmoid_{Taylor} = \frac{1}{2} + \frac{1}{4} * x - \frac{1}{48} * x^3 + \frac{1}{480} * x^5 - \frac{17}{80640} * x^7 + \frac{31}{1451520} * x^9 ... \quad (2)$$

A Taylor series expansion itself is unsuitable for spline approximation as the approximation error is maximised at the boundaries of the spline i.e. large errors are present at the points where the splines are joined. [DAN: DOESN'T REALLY EXPLAIN WHY A TAYLOR IS UNSUITABLE AS OPPOSE TO ANOTHER TECHNIQUE – AREN'T THEY ALL PIECE-WISE? ALSO MAY CONFUSE THE READER INTRODUCING TAYLOR FOR EQN 2 AND THEN NOT USING IT, ALTHOUGH I SEE WHY YOU DID THIS.] Consequently, other approximating polynomials, such as least squares have been employed [11]. We propose using the Remez reduction algorithm to solve the minimax spline approximation [15] since this approach is novel in the context of fixed point ANN activation function approximation. A minimax polynomial minimises the maximum error over a range for a given order. As will be detailed in section 5, this greatly improves the approximating error relative to other polynomial approximating schemes such as least squares. We advocate using a first order minimax approximation, as this is capable of achieving an acceptable error (see section 2 DAN: YOU DON'T REALLY TALK ABOUT ACCEPTABLE ERROR IN SECTION 2 EXCEPT IN VERY HIGH LEVEL TERMS. I SUGGEST REPHRASING OR DELETING THIS REFERENCE TO SECTION 2) with a small number of approximating polynomials. This has the further benefit of avoiding higher order x^n operations. To further reduce the number of splines required to achieve a specified accuracy, we employ the common approach of range reduction that exploits inherent redundancies (e.g. symmetry, periodic behaviour, etc) so that fewer splines can be used to fully represent the function.

3.1 Optimisation of spline locations

The placement of the approximating polynomials on the input range clearly has a major bearing on the overall approximation error. The simplest approach is to evenly distribute the polynomials over the approximating range. However, astute placement can reduce the approximating error, although the potential search space for optimal placement is large. For example, when using 5 polynomials in a 0 to 8 range with a precision of 10^{-3}, there are in the order of 2^{17} possible combinations for the location of the polynomials. To address this large search space issue, we propose using a genetic algorithm (GA) to find the optimum location of the approximating polynomials. Unlike an exhaustive search, this solution is scalable even if the input range were to become extremely large (for example, double precision floating point). DAN: OK BUT WHY A GA SPECIFICALLY AS OPPOSE OT ANOTHER OPT APPROACH?

The GA was implemented using Matlab [16]. The fitness function firstly uses the Remez reduction algorithm [DAN: NEED A ONE LINER EXPLANATION OF WHAT'S HAPPENING HERE] to calculate the minimax polynomial

coefficients for each candidate in the population. Using these coefficients, the minimax spline approximation to the chosen activation function (e.g. Sigmoid) is constructed. The mean and maximum errors are then calculated from this approximation, using at least 10^6 samples. The GA explores the search space whilst attempting to minimise these mean and maximum error values.

As is usually the case, the GA needed extensive tuning through trial and error exploration of the different input parameters. Initial population sizes of 10 to 1,000 were considered, along with extensive investigation into different crossover functions and different mutation functions. Our best results were obtaind using arithmetic crossover and a multi-point non uniform mutation with 5 mutation points.

4 Hardware Architecture

STILL WORKING ON THIS SECTION

4.1 Range Reduction & Reconstruction

4.2 Calculation of Approximating Polynomials

Array Multiplier

Distributed Arithmetic Multiplication

5 Results - still needs work

5.1 Approximation Error

[DAN: NEED TO EXPLAIN HERE WHAT FIGURES IN THE TABLE MEAN AND WHY IT IS A FAIR COMPARISON. E.G. WHY IS FAIR TO COMPARE OUR APPROACH TO THAT OF [11] – DIFFERENT RANGES AND NOT CLEAR HOW MANY SEGMENTS THEY USE] We used the proposed sigmoid activation function approximation to compare our approach to previous research. We tested our method using 2 to 8 approximating polynomial segments. The maximum and average error results for these tests can be seen in table 2. When using 5 or more segments, our method outperforms all related research. Comparing our 4 segment implementation with the first and second order 4 segment approach of [11], our implementation gives an error that is almost 4 times smaller. The improvement obtained by using the genetic algorithm is apparent from comparing our five segment approach with the five segment approach of [12]. Our approach is 36% more accurate despite the fact that [12] employs a single precision floating point data representation.

[DAN CHANGE "OUR APPROACH" IN TABLE TO "PROPOSED APPROACH"]

Design	Range	Maximum Error	Average Error
Myers et al [8]	[-8,8[	0.0490	0.0247
Alippi et al [9]	[-8,8[	0.0189	0.0087
Amin et al [10]	[-8,8[	0.0189	0.0059
Vassiliadis et al (First Order) [11]	[-4,4[	0.0180	0.0035
Vassiliadis et al (Second Order) [11]	[-4,4[	0.0180	0.0026
Faiedh et al [12]	[-5,5]	0.0050	n/a
Basterretxea et al (q=3) [13]	[-8,8[	0.0222	0.0077
Tommiska (337) [14]	[-8,8[	0.0039	0.0017
Tommiska (336) [14]	[-8,8[	0.0077	0.0033
Tommiska (236) [14]	[-4,4[	0.0077	0.0040
Tommiska (235) [14]	[-4,4[	0.0151	0.0069
Our approach (2 segments)	[-8,8[	0.0158	0.0068
Our approach (3 segments)	[-8,8[	0.0078	0.0038
Our approach (4 segments)	[-8,8[	0.0047	0.0024
Our approach (5 segments)	[-8,8[	0.0032	0.0017
Our approach (6 segments)	[-8,8[	0.0023	0.0012
Our approach (7 segments)	[-8,8[	0.0017	0.0009
Our approach (8 segments)	[-8,8[	0.0013	0.0009

Table 2. Maximum and average Sigmoid Approximation errors

5.2 Hardware Implementation Results

The design was captured in Verilog HDL and synthesised using Synopsys Design Compiler and Synplicity Synplify Pro for a 90nm TSMC ASIC library and Xilinx Virtex 2 FPGA respectively. Power consumption estimates for the ASIC library were generated using Synopsys Prime Power. A summary of the synthesis results for a five segment implementation can be seen in table 3. The distributed arithmetic approach uses 15% less area than the array multiplier for an ASIC implementation.

[DAN: AGAIN DESCRIBE WHY THIS IS A VALID COMPARISON – REMEMBER REVIEWERS ARE UNLIKELY TO BE HARDWARE BODS]

	Area [Gates]	Max Frequency [MHz]	Average Power [mWatts]
ASIC - Array multiplier	1917	250	0.1423mW -this figure is a problem!!
ASIC - Distributed Arithmetic	1635	250	0.152mW
FPGA - Array multiplier	2800	40	n/a
FPGA - Distributed Arithmetic	?	40	n/a

Table 3. Synthesis Results

DISCUSS FPGA IMPLEMENTATION

COMPARE TO RELATED RESEARCH - ATTEMPT TO NORMALISE THEIR RESULTS

Both array multiplier and distributed arithmetic architectures complete processing within one clock cycle. This compares very favourable to a direct implementation of a sigmoid activation function on an ARM 9 processor, which we profiled (see table 4) as requiring a minimum of 395 clock cycles [DAN: PROVIDE MORE DETAIL - OPTIMISED IMPLEMENTATION, OR THROWN TOGETHER?]. Profiling also revealed that even a first order spline approximation requires a minimum of 55 clock cycles on an ARM 9 processor. [DAN: WHY IS THIS INTERESTING .. COS WE'RE SO MUCH BETTER THAN THAT AND YET ONLY NEED 1 CYCLE ... HAND HOLD THE READER THROUGH THIS EVEN AT THE RISK OF SOUNDING "OBVIOUS"]

Design	Floating Point Co-processor	Instruction count	Clock cycles
Direct implementation	not present	1,688	2,550
Direct implementation	present	114	395
1^{st} order spline approximation	not present	323	448
1^{st} order spline approximation	present	55	163

Table 4. Sigmoid activation function profiling on an ARM 920 Processor

6 Future work and conclusions

This paper presented a high precision, scalable hardware architecture for an activation function generator. It represents preliminary work toward a complete power efficient hardware ANN accelerator for mobile platform suitable for handling high dimensional data sets such as those used in multimedia applications. In the future, we plan to extend the number of activation functions supported, along with modifications to the genetic algorithm to compensate for any biases introduced from the rounding of the coefficients to a finite word length [DAN: NEED TO EXPLAIN OR REMOVE]. In addition, we also intend to conduct a thorough investigation on the suitability of using floating point representation.

References

[1] Benard Widrow, David E. Rumelhart, and Michael A. Lehr, "Neural networks: Applications in industry, business and science," *Communications of the ACM*, vol. 37, no. 3, pp. 93–105, Mar. 1994.

[2] G. Cauwenberghs and M. Bayoumi, *Learning on Silicon - Adaptive VLSI Neural Systems*, Kluwer Academic, 1999.

[3] David Zhang and Sankar K. Pal, *Neural Networks and Systolic Array Design*, World Scientific, New Jersey, 2002.

[4] U. Ruckert, "ULSI architectures for artificial neural networks," *IEEE Micro*, vol. 22, no. 3, pp. 10–19, May 2002.

[5] Sorin Draghici, "On the capabilities of neural networks using limited precision weights," *Neural Networks*, vol. 15, no. 3, pp. 395 – 414, Apr. 2002.

[6] J.L. Holt and J.N Hwang, "Finite precision error analysis of neural network hardwareimplementations," vol. 42, no. 3, pp. 280 – 291, Mar. 1993.

[7] Simon Haykins, *Neural Networks - A Comprehensive Foundation*, Prentice-Hall, 1999.

[8] D.J. Myers and R.A Hutchinson, "Efficient implementation of piecewise linear activation function for digital vlsi neural networks," in *Electronics Letters*, Nov. 1989, vol. 25, pp. 1662–1663.

[9] C. Alippi and G. Storti-Gajani, "Simple approximation of sigmoid functions: Realistic design of digital vlsi neural networks," in *Proceedings of the IEEE Int'l Symp. Circuits and Systems*, June 1991, pp. 1505–1508.

[10] H. Amin, K.M. Curtis, and B.R Hayes Gill, "Piecewise linear approximation applied to nonlinear function of a neural network," *IEE Proceedings Circuits, Devices and Systems*, vol. 144, pp. 313 – 317, Dec. 1997.

[11] S. Vassiliadis, Ming Zhang, and J.G. Delgado-Frias, "Elementary function generators for neural-network emulators," *IEEE Transactions on Neural Networks*, vol. 11, no. 6, pp. 1438 – 1449, Nov. 2000.

[12] H. Faiedh, Z. Gafsi, and K. Besbes, "Digital hardware implementation of sigmoid function and its derivative for artificial neural networks," in *Proceedings. of the 13th International Conference on Microelectronics*, Rabat, Morocco, Oct. 2001, pp. 189 – 192.

[13] K. Basterretxea, J.M. Tarela, and I. del Campo, "Approximation of sigmoid function and the derivative for hardware implementation of artificial neurons," *IEE Proceedings of Circuits, Devices and Systems*, vol. 151, no. 1, pp. 18–24, Feb. 2004.

[14] M.T. Tommiska, "Efficient digital implementation of the sigmoid function for reprogrammable logic," *IEE Proceedings Computers and Digital Techniques*, vol. 150, pp. 403 – 411, Nov. 2003.

[15] J.A. Pineiro, S.F. Oberman, J.M. Muller, and J.D. Bruguera, "High-Speed Function Approximation Using a Minimax Quadratic Interpolator," vol. 54, pp. 304 – 318, Mar. 2005.

[16] "The MathWorks - MATLAB and Simulink for Technical Computing," http://www.mathworks.com/.