Towards Hardware Acceleration of Neuroevolution for Multimedia Processing Applications on Mobile Devices

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

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

Abstract. This paper addresses the problem of accelerating large artificial neural networks (ANN), whose topology and weights can evolve via the use of a genetic algorithm. The proposed digital hardware architecture is capable of processing any evolved network topology, whilst at the same time providing a good trade off between throughput, area and power consumption. The latter is vital for a longer battery life on mobile devices. The architecture uses multiple parallel arithmetic units in each processing element (PE). Memory partitioning and data caching are used to minimise the effects of PE pipeline stalling. A first order minimax polynomial approximation scheme, tuned via a genetic algorithm, is used for the activation function generator. Efficient arithmetic circuitry, which leverages modified Booth recoding, column compressors and carry save adders are adopted throughout the design.

1 Introduction

Artificial neural networks (ANN) have found widespread deployment in a broad spectrum of classification, perception, association and control applications [1]. However, finding an appropriate network topology and an optimal set of weights represents a difficult multidimensional optimisation problem. Ideally, the topology should be as small as possible, but large enough to accurately fit the training data. Failure to find a suitable configuration will cause poor generalisation ability with unseen data and/or excessive execution time. One possible solution to this issue is to use a genetic algorithm to evolve an optimum topology and/or weights. This approach is also sometimes known as Evolutionary Artificial Neural Networks (EANN) or Neuroevolution (NE) [2][3]. As well as reducing the requirement for trial and error design exploration for the ANN, the approach is more robust at avoiding local minima and has the scope for finding a minimal topology [2]. A minimal topology is hugely beneficial since fewer neurons and synaptic connections lead to reduced computation, which in turn means higher throughput and lower power consumption.

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

However, even with minimal topologies, when processing high dimensional datasets, for example multimedia data, the NE process may not meet system requirements (throughput and/or power consumption). In the case of multimedia data, poor performance is likely to be a consequence of the unavoidable combination of a requirement for extremely high throughput for real time processing and large complex ANN topologies caused by a high number of inputs. The associated computational complexity is highly undesirable from a real time operation and low power consumption perspective. 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.

One possible solution to NE complexity issues, is to offload the computational burden from the host processor to a dedicated hardware accelerator. Although a general consensus emerging in recent times is that the viability of dedicated hardware ANN processors are questionable [4]. However, due to the aforementioned throughput and power consumption issues, ANN hardware acceleration still provides an attractive and viable solution particularly in the context of constrained computing platforms. This has motivated us to design an efficient and flexible hardware ANN accelerator, which is suitable for NE tasks. We have chosen to investigate the widely used open source Neuro Evolving Augmented Topologies (NEAT) software library [3]. Our profiling has revealed the computational burden of the ANN evaluation is suitable for hardware off load, whilst the genetic algorithm routines, which use moderate computational resources can reside in software. This scalable co-design methodology combines the reconfigurability of software with the speed of hardware [5]. This also facilitates application re-usability, where the core could be re-deployed for a number of applications. It should be noted that hardware acceleration will not be applied to the genetic algorithm itself.

The rest of this paper is organised as follows: Section 2 details related prior research in the area. Section 3 discusses NE hardware architecture design challenges and choices. Section 4 outlines the hardware implementation of the proposed architecture. Section 5 details hardware synthesis results and power consumption estimates. Future work is outlined in Section 6, whilst Section 7 draws conclusions about the work presented.

2 Related Research

There are many approaches to NE, differing principally in the choice of genome encoding scheme and the operators chosen for mutation and crossover. NEAT is an example of a direct encoded node based NE approach. Each genome consists of a number of link and neuron genes. Fig. 1 shows an example genome and the principal constituent elements of each gene type. NEAT uses a genome historical marking scheme (shown as the *innovation number* in Fig. 1), which avoids many of the problems associated with other NE approaches. This scheme allows meaningful topology crossover to occur, and furthermore, it avoids any computational overhead of topology analysis when producing the valid offspring. Taking

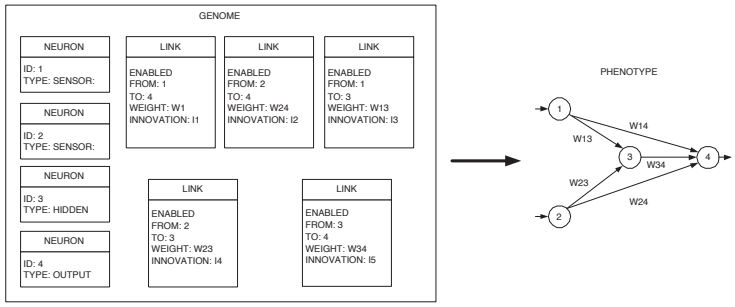

Fig. 1. Example of an NEAT genome and resultant phenotype

inspiration from biological evolution, NEAT introduces the concept of *complexification*, whereby processing starts with a minimal topology with no hidden nodes and each input is connected to an output node. Each successive generation systemically elaborates the complexity by adding neurons and/or synaptic connections. This is attractive in the context of our interest in mobile devices since minimal topologies are favoured, thus reducing computational complexity for a given problem. During complexification, topology innovation occurs. However, adding structure will typically reduce the fitness function, since smaller structures optimise faster. Consequently, there is little opportunity for the new topology to survive [3]. Again taking inspiration from biological evolution, where an innovation needs time to reach its potential, NEAT uses a process of *specification*. This allows genomes to compete only against their own species and gives the new offspring a better chance of survival. The combination of these features, allows NEAT to outperform other NE approaches [3], and for this reason, we have chosen it for further investigation.

2.1 ANN Hardware Implementations

There has been considerable research in analog and digital hardware ANN implementations. Analog implementations typically have the benefit of high speed operation, and smaller area compared to a digital implementation. However an analog design has a number of drawbacks including susceptibility to component process variation, electrical noise and environmental conditions, along with resolution issues for the weights and activation function. A digital hardware implementation, on the other hand, does not suffer from these drawbacks, and furthermore allows ease of design and computational accuracy. For these reasons we have adopted a digital design approach.

A digital implementation typically stores the network topology and/or synaptic weights in memory. These values are then later retrieved and processed in discrete chunks by the parallel processing elements (PE). This is advantageous because any network size and potentially any topology can be handled. A PE usually consists of multiply accumulate circuitry [6] and sometimes depending on the configuration, an activation function generator. The number of processing elements implemented is a design space trade off between area, power and performance. This design space extends from a single PE to a PE for each neuron or

even a PE for each synaptic calculation. A discussion of the PE implementation challenges is given in Section 4.

One of the principal challenges for ANN acceleration using time shared PE's is how to get the ANN data from slow, bandwidth limited memories to the high speed PEs in a fast, bandwidth efficient manner. Systolic array architectures have been the pervasive choice for bridging this gap [6]. It is well established that systolic arrays offer many benefits for ANN with regard to using memory bandwidth effectively by maximising data reuse, in addition to permitting highly regular PE control logic. However, as will be demonstrated in Section 3, topologies with sparse synaptic connections considerably reduce the efficiency of systolic array approaches, in addition to increasing memory requirements. Furthermore, as sparse topologies result in fewer synaptic calculations, it could be exploited to reduce power consumption. Despite this, the literature contains very few hardware implementations, which attempt to exploit topology sparseness The literature rather focuses on dense connectivity neural algorithms.

Our work provides a hardware ANN accelerator extension for NEAT, using an architecture which is suitable for any evolved topology, as well as having the ability to exploit sparse connectivity. The intended target of this work is as an accelerator for mobile devices, consequently the focus is on power efficiency, rather than ultra high performance acceleration.

3 Proposed Hardware Architecture

Unlike a conventional ANN, an evolved network can have any topology, potentially with a mixture of forward synaptic connections, recurrent synaptic connections and looped current synaptic connections. Furthermore, owing to the complexification process, NEAT will naturally favour sparse topologies. These factors have important consequences for the hardware architecture. The efficiency of systolic array architectures is dramatically reduced when operating on sparse neural topologies. This is because the data flow through the systolic array is frequently interrupted and thus the throughput benefit of multiple PEs is not being achieved. To overcome this, the PE can be either disabled for that synaptic calculation or have a weight of zero. However both solutions are undesirable. Disabling the PE leads to additional control logic for each individual PE. Whilst storing weights with a zero value leads to increased memory sizes, which further exacerbates power consumption issues, particularly as memory power consumption disproportionately increases with size. The systolic array efficiency is further reduced due to the presence of recurrent synaptic connections. This causes unpredictable feedback synaptic connections from other neurons, causing further data flow interruptions. However more importantly, as systolic array architectures favour layered ANN, where the inputs to each PE layer are well defined, it is not a trivial matter to dynamically reconfigure the PE inputs for the evaluation of alternative topologies, which contain recurrent links. This situation would be necessary for an application where NEAT is evolving in real-time, such as artificial intelligence for computer gaming [7]. Whilst it is possible to modify the genetic algorithm so that only forward synaptic connections are added, this compromises the quality of the evolved ANN solution.

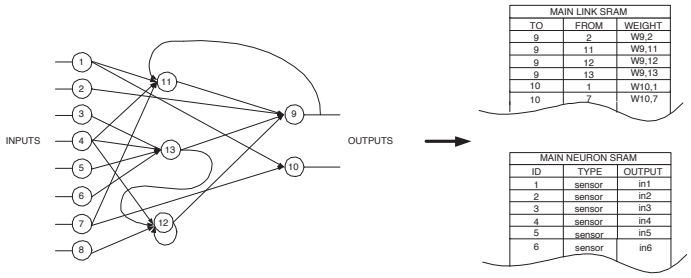

Fig. 2. NEAT phenotype to hardware memory mapping

These factors have motivated us to explore alternative architectures rather than a traditional systolic array approach. Examining the NEAT genome data structures, it is clear that the essential information in the link and neuron genes, can be mapped easily to hardware memories, as is demonstrated from the simple ANN in figure 2. Essentially we propose "parsing" through the *LINK* memory to retrieve the relevant weights and using the "*LINK→FROM*" field as the index to retrieve the output from the appropriate neuron in the *NEURON* memory. Once the values are retrieved the multiply accumulate operation is performed. This operation is repeated for all synapses associated with that neuron, before the activation function is calculated. The process then repeats for all neurons.

To increase throughput, two PEs operating in parallel are used, as can be seen from the proposed architecture in Fig. 3. The PE datapath is 64 bits and each PE has access to the memory (via a memory controller), thus two PEs were chosen so as to give a good trade off between throughput and bus addressing complexity. Each PE is equipped with a local "*LINK*" SRAM. This is loaded with the incoming synaptic connections for that neuron. To reduce stalling, the synaptic connections for neuron $N + 1$ are prefetched, whilst in parallel the PE processes neuron N. The PE datapath is 64 bits wide because 4×16 bit entries from the local *LINK* SRAM are retrieved in a burst. Parallel processing is possible since the "*LINK→WEIGHT*" values are processed sequentially and they do not have interdependencies. The decision to use 4 parallel arithmetic units per PE was chosen as a trade off between throughput, bus width size and to minimise the complexity of the hierarchical memory system.

Unless a fully connected topology is being evaluated, the 4 "*NEURON→ID*" addresses decoded from "*LINK→FROM*" will not necessarily be contiguous. Therefore if a single "*NEURON*" SRAM is used (such as in Fig. 2), only one "*NEURON*" address could be processed per clock cycle. However to maximise the performance of the multiple arithmetic units in the PE datapath, valid data needs to be available on each clock cycle. To overcome this issue, we propose partitioning the neuron memory into 8 smaller SRAMs, as can be seen in Fig. 3. To further increase the probability that all data units can be retrieved within one clock cycle, we propose using a data cache to maximise data reuse. The cache provides backup should two or more data requests occur for the same "*NEURON*" SRAM bank. This could occur based on the connectivity of the topology, in particular if the sparse synapses were modulo 8 apart.

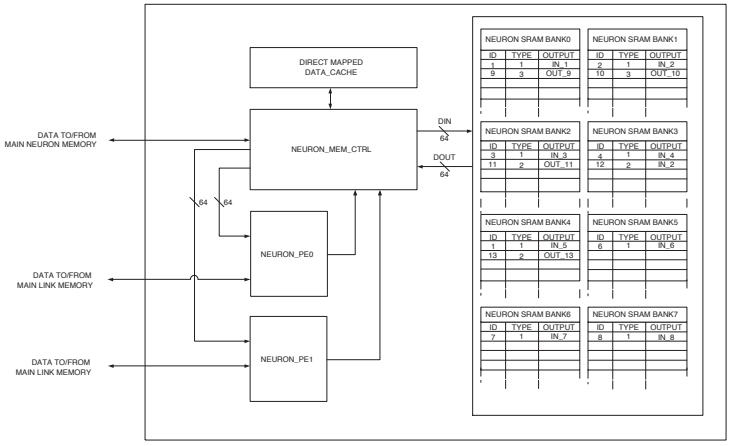

Fig. 3. Simplified block diagram of the proposed NE hardware accelerator datapath

When the memory control logic receives a request for 8 new values from the 2 PEs, the cache is firstly examined to see if it can fulfil these requests. Should cache misses occur, the "*NEURON→ID*" is decoded to indicate the relevant SRAM bank. If 2 or more requests attempt to access the same SRAM bank, the data must be retrieved over multiple clock cycles, otherwise all requests can be handled in one clock cycle. In the worst case scenario, when none of the data is present in the cache and all requested data is located in the same "*NEURON*" SRAM bank, one multiply accumulate operation occurs per clock cycle. On the other hand, if all the requested data is either in the cache or different Neuron SRAM banks, 8 multiply accumulates occur per clock cycle. For a fully connected feed forward ANN, the performance of this proposed system will be similar if not identical to a systolic array with the same number of arithmetic units, and we believe, the system will also statistically outperform a systolic array architecture when processing a mixed forward and recurrent sparse connection topology. Performing this comparison is targeted as future work.

4 Hardware Implementation

A fundamental digital implementation design decision is what format to use for the data representation There are at least three popular approaches for digital ANN – stochastic, fixed point and floating point. Stochastic digital ANN implementations encodes the value of a signal using the probability of a given bit in a stochastic pulse stream being a logic 1 [8]. This has the benefit that many common arithmetic operations require very simple logic [9]. Whilst beneficial from an area perspective, there are issues concerning representation variance. Furthermore, due to the increased latency from the inherently serial operation, a higher clock frequency is required to match the throughput from a more parallel fixed point implementation. The substantially increased clock frequency is of considerable concern in the clock tree network from a power consumption perspective, particularly in deep sub micron technologies. Floating point implementations on the other hand offer a wide dynamic range suitable for the typical distribution of ANN weight values, however it has a considerable area overhead

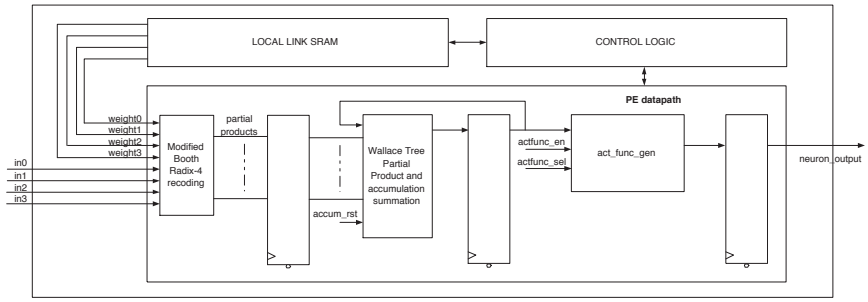

Fig. 4. Simplified block diagram of the Neuron PE

for arithmetic operators. Consequently we have chosen a fixed point representation, as we believe it offers a reasonable trade off between area, power and performance.

The width of the datapath in a fixed point implementation is a vital design decision. To minimise area and power consumption, the minimum number of bits should be chosen, which will result in an acceptable error. Reduced fixed-point precision hardware ANN issues were explored in [10]. It was found that 10 precision bits were sufficient for multi-layer perceptrons trained via back propagation [10]. Using fewer precision bits than this will effect the convergence speed, and in some cases may completely prevent convergence. Related to this, input/output standardising (sometimes called scaling) is advised in most cases. Standardising the inputs will results in a smaller integer range. This leads to fewer bits switching and consequently power savings. For these reasons, we propose using 16 bits in a 6.10 fixed point representation (6 integers bits and 10 fractional bits) throughout the design, with the input data standardised to a range of -1 to 1. The 6 integer bits allows net accumulation values to grow to levels, which maximally exploit the resolution achievable from the proposed activation function generator. The remainder of this section will discuss the architecture implementation issues and decisions.

4.1 PE Implementation

The function of the PE is to generate the neuron weighted sum of inputs, this clearly requires multiply accumulate circuitry. We have also chosen to add the activation function generator to each PE (see Fig. 4). An alternative approach is to time share a single activation function generator between multiple PEs. This is typically the approach adopted when using a systolic array architecture due to the highly regular processing. For our proposed architecture, control logic must be designed to ensure only a single PE has control of the activation function generator at any clock cycle. However as we are currently using only two PEs and that the area overhead for the activation generator is not considerable ($< 2,000$ gates), we chose to integrate the activation function logic into the PE.

In a system where the ANN weights are static, canonic signed digit representation and multiplier-less distributed arithmetic architectures can be employed in the generation of the weighted sum. However, static weights are clearly not suitable for NE. Fortunately, owing to the prevalence of the sum of products

operation in signal processing algorithms and matrix arithmetic, there has been considerable research on efficient hardware implementations. Consequently, we have chosen a fused multiply add approach as can be seen in Fig. 4 [11]. The number of partial products has been halved by using modified Booth radix 4 coding and the accumulation step is merged with the addition of the partial products using a Wallace tree [11]. Furthermore, the generation of the two's complement for the modified Booth algorithm uses a simple inversion for the one's complement and delays adding the additional one, until the Wallace tree stage, thereby reducing the critical path. The final sum and carry are then added using an efficient carry propagate adder.

Each PE has ≈ 3KB of local SRAM to store "*LINK→FROM/WEIGHT*" data, providing enough storage for the details of over 1000 synaptic connections. Obviously the amount of memory can be adjusted based upon the timing constraints of the main memory and connectivity characteristics of a particular application. The PE control logic, which governs access to the local SRAM and the control signals for the PE datapath is outlined in Algorithm 1.

Algorithm 1: Neuron PE datapath control flow

1 ***MEM_SETUP setup***;
  Load PE SRAM with "LINK→FROM/WEIGHT" data for first "LINK→TO" neuron;
  Regular processing starts once loaded;
  In parallel, prefetch "LINK→FROM/WEIGHT" data for the next neuron;
2 ***LINK_DECODE Stage***;
  PE requests 4 "LINK→FROM/WEIGHT" entries from local SRAM;
3 ***INPUT_FETCH Stage***;
  Using the 4 retrieved "FROM" addresses, the PE requests these values from the "NEURON" SRAM memory banks;
  The "WEIGHT" values are set up on the inputs to the partial product generation logic;
4 ***PP_GEN Stage***;
  With the input values returned from the "NEURON" SRAM banks, the partial product generation logic is enabled;
5 ***ACCUM Stage***;
  Partial products and the previous accumulation are added in the Wallace Tree compressor;
6 ***ACT_FN Stage***;
  If necessary the activation function generator is enabled;
7 ***WRITE_BACK Stage***;
  If the activation function is enabled, output is written to memory;

We have recently proposed an efficient hardware architecture for an activation function generator [12], which improves the approximation error upon prior research. Our approach uses a minimax polynomial spline approximation, with a genetic algorithm (GA) leveraged to find the optimum location of the approximating polynomials. The GA typically improved the approximating error by 30% to 60% relative to an even distribution of the approximating polynomials. Using a spline-based approach has the benefit that multiple activation functions can be accommodated by merely changing the coefficients of the approximating polynomial. This is beneficial from an evolutionary perspective and uses minimal extra hardware to support the additional functions.

4.2 Cache Design

To minimise the area and control logic overhead a direct mapped cache implementation has been chosen [13]. The selection of the cache size represents an

important design trade off, a larger cache will have fewer cache misses, but will have a larger area. In our initial design investigation we use 2 parallel PEs each with 4 arithmetic units, as a result we believe a cache size of 64 blocks will be appropriate. However, further optimisations should be possible with the cache size by tailoring it to the statistics of the ANN topology for a particular application, e.g. video analysis. Each block consists of 2 elements, the neuron address and the neuron data. A benefit of using a direct mapped cache is that logic for a cache block replacement strategy is not required. When a value from the activation function generator is written back to SRAM, a stall signal is issued and if necessary the cache is updated. This avoids any potential data mismatches and the need for additional storage to hold a "block dirty" value in the cache [13].

5 Results

Prior to hardware implementation, we carried out profiling on two classical ANN/GA problems, the XOR problem and the double pole balancing problem, using the NEAT software library. Despite the fact that both evolved topologies resulted in a small number of neurons, the evaluation of the ANN was the clear computational hot spot. Additionally, it is fair to assume that as the number of neurons in the genome increases (for example in multimedia tasks), this computational hot spot will only worsen.

The hardware design was captured in Verilog HDL and synthesised using Synopsys Design Compiler using a 90nm TSMC ASIC library. Power consumption was estimated from a gate level netlist with a 1.2 volt voltage source. A summary of the preliminary synthesis and power results can be seen in table 1. It should be noted that the area figures do not take into account the *NEURON* and *LINK* memory storage elements. The design was synthesised using a 200MHz and a 300MHz clock frequency, these are typical speeds of mobile microprocessors. As would be expected, the higher clock frequency design has a marginally larger area. Power estimates were generated using the data derived from the "Winning" topology for the XOR and double pole balancing tests generated in software. Owing to the fact that the topologies were small, little discernible difference was noted in the average power between the two data sets. We believe these power consumption figures are appropriate for deployment on a mobile device. Under optimum conditions, our proposed hardware calculates 8 multiply accumulate operations per clock cycle and requires 1 clock cycle for the calculation of the activation function. This compares favourably with modern mobile processors, which typically achieve a sustained 1 multiply accumulate operation per clock cycle. Comparison of results with other ANN implementations is difficult, as no other approach from the outset attempts to accelerate the ANN evaluation within neuroevolution and exploit the associated sparse topologies. Normalisation of results will be necessary to give a fair comparison and this is targeted as a future work item.

Table 1. Preliminary synthesis results

	Frequency [MHz]	Area [Gates]	Average Power [mW]
Proposed architecture	200	52,186	42.27
Proposed architecture	300	55,726	69.55

6 Future Work

The current bottleneck in the design is retrieving the topology from memory for the multiple PEs. Alternative micro-architectures are currently being investigated, which could give throughput benefits and improve the scalability of the architecture. The cache module warrants further investigation, in particular the effects of different caches sizes, architectures and different input datasets from NEAT. Before integration with the NEAT software library begins, a comparative power consumption study between fixed point, reduced word length floating point and stochastic implementations is planned. Benchmarking will be necessary to compare the performance (throughput and power consumption) of the dedicated hardware core to a software implementation running a general multimedia processing task such as face detection in video sequences.

7 Conclusions

The computational complexity associated with Neuroevolution for multimedia applications on mobile devices is highly undesirable from a throughput and power consumption perspective. This paper has proposed a viable hardware architecture for accelerating the neural network genome calculation. The architecture is flexible enough to process any ANN topology, whilst still providing a good trade off between area, power and throughput.

References

[1] Widrow, B., Rumelhart, D.E., Lehr, M.A.: Neural Networks: Applications in Industry, Business and Science. Communications of the ACM **37** (1994) 93–105

[2] Fogel, D.B., Fogel, L.J., Porto, V.W.: Evolving neural networks. Biol. Cybern. **63** (1990) 487–493

[3] Stanley, K.O., Miikkulainen, R.: Evolving neural networks through augmenting topologies. Evol. Comput. **10** (2002) 99–127

[4] Omondi, A.R.: Neurocomputers: A Dead End? International Journal of Neural Systems **10** (2000) 475–481

[5] Reyneri, L.: Implementation issues of neuro-fuzzy hardware: going toward hw/sw codesign. IEEE Transactions on Neural Networks **14** (2003) 176–194

[6] Kung, S., Hwang, J.: A Unified Systolic Architecture for Artifical Neural Networks. Journal of Parallel and Distributed Computing **6** (1989) 358–387

[7] Stanley, K., Bryant, B., Miikkulainen, R.: Real-time neuroevolution in the NERO video game. IEEE Transactions on Evolutionary Computation **9** (2005) 653–668

[8] Gaines, B.: Stochastic Computing Systems, Advances in Information Systems Science. Plenum Press New York (1969)

[9] Brown, B.D., Card, H.C.: Stochastic Neural Computation I: Computational Elements. IEEE Transactions on Neural Networks **50** (2001) 891–905

[10] Holt, J., Hwang, J.: Finite precision error analysis of neural network hardware implementations. IEEE Transactions on Computers **42** (1993) 280 – 291

[11] Koren, I.: Computer Arithmetic Algorithms. A K Peters Ltd (2001)

[12] Larkin, D., Kinane, A., Muresan, V., O'Connor, N.: An Efficient Hardware Architecture for a Neural Network Activation Function Generator. In: International Symposium on Neural Networks, Chengdu, China (2006)

[13] Hennessy, J.L., Patterson, D.A.: Computer Architecture: A Quantitative Approach. 3rd edn. Morgan Kaufmann Publishers (2003)