Efficient and robust bitstream processing in binarized neural networks

S. Aygun, E.O. Gunes and C. De Vleeschouwer

In the neural network context, used in a variety of applications, binarized networks, which describe both weights and activations as single-bit binary values, provide computationally attractive solutions. A lightweight binarized neural network (BNN) system can be constructed using only logic gates and counters together with a two-valued activation function unit. However, BNNs represent the weights and the neuron outputs with only one bit, making them sensitive to bit-flipping errors. Binarized weights and neurons are manipulated by the utilization of bitstream processing with regard to stochastic computing (SC) to cope with this error sensitivity. SC is shown to provide robustness for bit errors on data while being built on a hardware structure, whose implementation is simplified by a novel subtraction-free implementation of the neuron activation.

Introduction: Emerging technologies such as nanoscale devices pave the way for a better trade-off between computation, power, and area. However, these hardware systems can be vulnerable to stuck-at-fault and bit-flip errors. Therefore, it is crucial to design fault-robust and noise-robust algorithms. *Stochastic Computing* (SC) has introduced a hardware efficient bitstream computational paradigm that manipulates data in the form of non-stationary Bernoulli sequences [1] to provide the same level of significance to all bits, and thereby be robust to bit-flipping errors [2]. Moreover, SC has the capability to reduce the hardware resource utilization by using simple logic gates instead of the sophisticated arithmetic units [3–6].

In the context of neural networks, now widely used in a large variety of applications, binarized networks, describing both weights and activations as binary values, provide computationally attractive solutions with competitive model prediction accuracy. However, in binarized neural networks (BNN) [7], the 1-bit representation of the weights is vulnerable to soft errors that are inherent to high-throughput nanoscale environments. Therefore, this letter proposes to adopt bitstream encoding and processing to handle the operations of the neural network. The bitstream processing paradigm using random streams is also named stochastic computing in literature. In terms of computational unit, BNNs involve a *sign* function at each neuron in hidden layers, to obtain $+1$ or -1 activation values. Moreover, each weight is restricted to $+1$ or -1 during the training. Thus, multiplication and summation operations are reduced to simple digital logic elements, like an XNOR gate and a counter, respectively.

A few studies have recently investigated the use of bitstream processing in the context of neural networks [2, 4–6]. Most of the preceding bitstream processing neural networks operate at either full precision or 2-or-more bit quantization of the weights and activations. Stochastic streams were first considered in the context of BNNs by Hirtzlin et al. [8], who proposed an improved training strategy by introducing input data streams into the network during training. For the MNIST Fashion dataset [9], they increased their stochastic network accuracy closer to the deterministic one. They employed the XNOR gate, counter, and the subtraction unit of activation for the hardware construction. In contrast, we propose a computational framework that derives the neuron activation from the accumulation unit, and thereby avoids the subtraction generally implemented in the neuron activation unit. In the simulation section, we underline how our solution is robust to errors injected into the images and network weights, which is the main theme of this letter.

Background: In bitstream processing, the numbers are represented by a binary encoding scheme, before being processed as a bit flow by simple logic hardware, to implement numerical operations. In this work, arguments in italic fonts are used to denote scalar numbers, while bold italic fonts denote their corresponding binary stream. Let x be a scalar value. Its binary stream representation is composed of N binary elements $\boldsymbol{x} = \{x_1, \ldots, x_N\}$ where $N \in \mathbb{Z}^+$, $N \geq 2$ define the *stream size*, and any m^{th} element out of N, x_m, is either binary *1* or binary *0*. In a bitstream, let K be the number of *1*s denoted by $K = \sum_{m=1}^{N} x_m$. Unipolar encoding (UPE) and bipolar encoding (BPE) are generally considered to encode any scalar into a bitstream, but the fractional number encoding is essential in neural network applications. A positive-only fractional number x, in the range $[0, 1]$, is encoded via UPE such that $x = \frac{K}{N}$. However, in BPE, signed number x, in $[-1, +1]$, is represented as $(x+1)/2 \equiv K/N$, so that decoding $\boldsymbol{x}$ stream is given represented by $x = \left[\left(\sum_{m=1}^{N} x_m\right) - \left(N - \left(\sum_{m=1}^{N} x_m\right)\right)\right]/N$. In the BPE stream, a majority of *1*s, i.e. *high values*, is an indication of the positive sign, whereas a majority of *0*s is an indication of the negative sign. Thus, BPE is superior to UPE in terms of sign estimation robustness. In both the schemes, at encoding, a Bernoulli sequence can be considered to convert the scalar number into a random sequence for perfect randomness of each bit [3]. In this work, data are represented using the BPE format in the proposed framework.

Proposed Framework: In this study, the acronym BNN refers to the conventional deterministic binarized neural network [7]. In contrast, the "BSBNN" acronym denotes the bitstream processing BNN. We first revisit how BNN works, and then introduce the proposed BSBNN implementation.

In any deterministic neural network, $S_m^{(n)}$ denotes the m^{th} neuron pre-activation related to the n^{th} hidden layer (HL). Pre-activation as $S_m^{(n)} = (x_1 \times w_{m,1}^{(n)}) + \cdots + (x_p \times w_{m,p}^{(n)})$ is computed with the multiplied and accumulated values of preceding neuron (or input) values, x, and the weights, w, where p is the number of neurons in the $(n-1)^{th}$ layer. In the hidden layers of the deterministic BNN, decimal weight and neuron output values, restricted to ± 1 during training, are mapped into 1-bit binary values such as $(+1)_{10} \equiv (1)_2$ and $(-1)_{10} \equiv (0)_2$. This makes it possible to implement bitwise multiplication using a single XNOR gate. Table 1 compares decimal and bitwise multiplications and demonstrates how decimal multiplication is degraded into a simple logic operation via XNOR gates.

Table 1: Multiplication in decimal and binary values.

Multiplication in Decimal	Multiplication in Binary (XNOR)	Decimal vs. Binary Equity
$+1 \times +1 = +1$	*1⊙1 = 1*	$(+1)_{10} \equiv (1)_2$
$-1 \times -1 = +1$	*0⊙0 = 1*	
$+1 \times -1 = -1$	*1⊙0 = 0*	$(-1)_{10} \equiv (0)_2$
$-1 \times +1 = -1$	*0⊙1 = 0*	

However, in deterministic BNN, accumulation is performed by counting the population of *1*s. For this operation, a modulus (MOD) digital counter circuit is utilized. Let T be the total number of states of the counter. Including a *zero* initial accumulation value, the counter can count up to the final state, $(T-1)$. The term *popcount*, short for population count, is generally used to denote the basic counter logic hardware [7, 8]. Fig. 1 depicts an example of an asynchronous counter based on the D-type flip flops for popcount. Binary inputs to be accumulated are fed sequentially to the first flip-flop clock input, and the base-2 output, $(Q_3 Q_2 Q_1)_2$, gives the total number of *1*s in the input binary word. It naturally resets to zero when the total number of *1*s presented to the input has reached a value that is the number of flip-flops (ffs) to the power of two, i.e. $2^{(\#\ of\ ffs)}$.

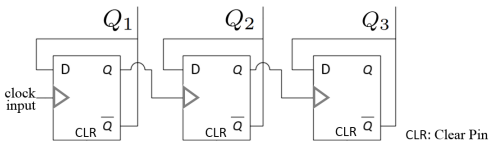

Fig. 1 $T = 2^3 = 8$ *states asynchronous counter example utilizing 3 D-type ffs.*

When considering our BSBNN framework, we implement multiplication and accumulation operations using bitstream sourced logic elements. Distinct from the prior art [8], bitstreams are performed not only to represent the image pixel values but also to define all neuron outputs and weights, that are handled throughout hidden layers. Thus, a -1 weight value in deterministic BNN is converted into $\boldsymbol{w} = \{0, 0, \ldots, 0_N\}$ for bitstream processing. Likewise, $+1$ is converted into $\boldsymbol{w} = \{1, 1, \ldots, 1_N\}$ for bitstream processing. Our experiments show that this way of representing weights and hidden values improves the bit-flip data error robustness.

Next, the implementation of the proposed BSBNN bitstream operations is detailed. In bitstream processing, the multiplier is either an AND gate for UPE bitstreams or an XNOR gate for BPE bitstreams. Since the weights can be $+1$ or -1 in the binarized network case, BPE streams are adopted, and the multiplication is implemented by the XNOR gate. Fig. 2 depicts an example of XNOR-based multiplication for $\boldsymbol{x}$ and $\boldsymbol{w}$ BPE bitstreams.

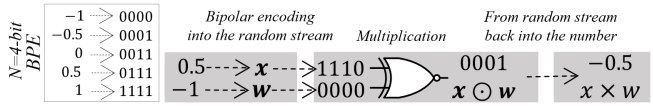

Fig. 2 *An illustration of number representation and bitwise multiplication. BPE in $N = 4$-bit is given on the left, and the places of each bit can be random.*

In BSBNN, the neuron activation can be computed directly by processing the concatenation of all multiplication bitstreams, denoted as $\left[(\boldsymbol{x_1} \odot \boldsymbol{w}^{(n)}_{m,1}), \ldots, (\boldsymbol{x_p} \odot \boldsymbol{w}^{(n)}_{m,p})\right]$, where $(\boldsymbol{x_i} \odot \boldsymbol{w}^{(n)}_{m,i})$ represents the multiplication stream related to the i^{th} neuron from the preceding layer. The concatenated bitstream is inputted to a counter, which decides if the output of the neuron should be activated or not. It depends on whether the concatenated bitstream contains a majority of *1*s. Our proposed solution uses a counter that resets itself and forwards a positive neuron activation as soon as the number of *1*s observed in the concatenated bitstream has attained a threshold, corresponding to the half size of this concatenated bitstream. Since the total number of states is not always a power of two, the counter combines a popcount with a specific masking logic to reset the popcount once it reaches the predefined hard-coded value T [10]. This concept is denoted as *truncated MOD-T counter* in the following. The term *truncated* refers to the fact that the counter resets before the maximal value of the popcount is reached, while *MOD-T* indicates that T is the modulus value of the counter. In practice, at every increment, the current total sum, $(Q_3Q_2Q_1)_2$ is presented to the bit masking logic to check whether the threshold is attained. The counter truncation is exemplified in Fig. 3 for the popcount presented in Fig. 1. After the output, $(Q_3Q_2Q_1)_2$ would have gone through 5 states, from $(000)_2$ to $(100)_2$, and it resets itself by setting the clear pin. To reset after 5 states, the output of the popcount is masked by a gate that outputs a $(1)_2$ only if $Q_1 = (1)_2$, $Q_2 = (0)_2$, $Q_3 = (1)_2$. This can be implemented with an AND gate, as shown in Fig. 3. As *1*s are present in majority in the input binary stream, whenever the 6^{th} high value is seen on the input, the masking logic outputs $(1)_2$.

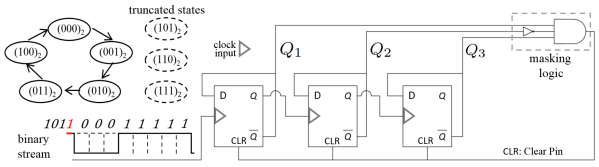

Fig. 3 *Popcount presented in Fig. 1 is truncated from $T = 8$ states into $T = 5$. It outputs a $(1)_2$ and resets itself, whenever the 6^{th} occurrence of high value is observed in the input bitstream.*

As depicted in Fig. 4, our proposal to use a truncated MOD-T counter simplifies the hardware in charge of turning the concatenated bitstream into a neuron activation. Former solutions, given in Fig. 4*a*, simply use a register to accumulate the number of *1*s counted in each of the input products and then rely on a subtraction to compare the accumulated sum with the pre-encoded threshold. In contrast, our proposed solution shown in Fig. 4*b* parses the concatenated products bitstream and embeds the pre-encoded threshold T directly in the truncated MOD-T counter. It does not need any extra register nor subtraction other than the masking logic.

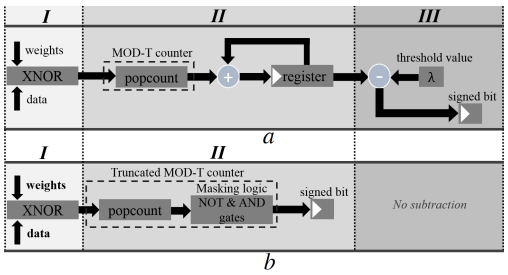

Fig. 4 *Architecture of prior bitstream processing BNN and the proposed model.*
a Hardware units of prior art by Hirtzlin et al. [8]. Region I: multiplication, Region II: population count and cumulative addition, Region III: decision of activation by subtraction
b Architecture of our proposed framework. The signed bit is 0 by default and is set to 1 if and only if the counter reaches a pre-defined threshold that has been hard coded in the masking logic. Region I: multiplication, Region II: population count & masking logic = truncated MOD-T counter

In Fig. 5, a numerical example is provided to clarify the overall proposal. By assuming this is the input layer of a neural network, input values are exemplified as 1, 0.5, 0.5, and weight values are 1, −1, 1. The overall concatenated bitstream size is 12 bits because there are three neurons, and each stream is 4 bits. The threshold is $T - 1 = 6$, as $T = 5$ including the *zero*th state, indicates $(101)_2$ hard-coded value in base-2 representation. To control the neuron activation bit, i.e. signed bit, the masking logic is performed at the output of the truncated MOD-5 counter that has been shown in Fig. 3.

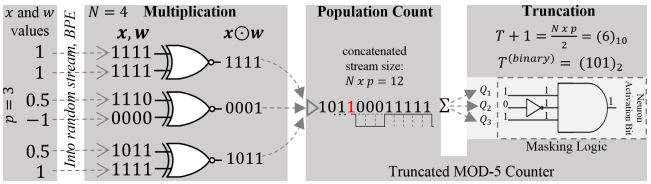

Fig. 5 *Input layer example of BSBNN with $p = 3$ neurons; x denotes the normalized grayscale image pixel values while w denotes the weight values. N is the length of a bitstream used to represent $\boldsymbol{x}$ and $\boldsymbol{w}$ values in BPE, and T corresponds to the modulo value of truncated MOD-T counter.*

To conclude, hardware units in deterministic BNN and BSBNN are compared in Table 2. In summary, it reveals that BSBNN does not require subtraction for the activation operation while using bitstream processing.

Table 2: Hardware units in BNN and BSBNN.

	Deterministic BNN	BSBNN
Multiplication	XNOR Gates	XNOR Gates
Addition	Popcount	Truncated MOD-T Counter
Activation	Subtraction Unit	-

Network on the Simulation: After the discussion on how BSBNN can be implemented efficiently, with reduced hardware logic, we analyse how BSBNN performs compared to BNN in case of input or model bit-flip errors. This section simulates our proposed network, using the MNIST handwritten digit dataset [11], and presents the obtained results. Fig. 6 presents the accuracy obtained on the MNIST digit dataset by previous bitstream-based neural network architectures as a function of the bitstream length. Most studies in the literature use a multi-layer perceptron with 2 HL, as has been considered in this study.

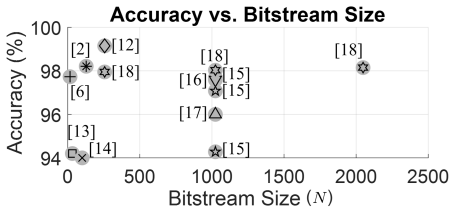

Fig. 6 *Previous efforts on the stochastic bitstream processing neural networks.*

During our simulations, we first consider a single model. It is obtained using deterministic BNN in the training phase. The forward path of the network uses binarized weights. However, the weight update is performed considering a floating-point version of the weights, which are then binarized to define the forward path. The network is made of four layers, each of them including 784-1024-1024-10 neurons, respectively. Squared hinge loss is chosen, which outperforms cross-entropy loss in terms of validation accuracy. An exponential learning rate decay is applied in the range of $[3 \times 10^{-7}, 3 \times 10^{-3}]$. The batch size is set to 100. Dropout is implemented with $p = 0.2$ for the input layer, and with $p = 0.5$ for hidden ones. The best performing model is saved by considering the best validation accuracy, 98.43%, at the 968^{th} epoch over 1000 epochs. The deterministic BNN test accuracy is 98.27%.

From a data representation point of view, traditional binary data in BNN is known to be quite vulnerable to bit-flipping errors. This is attributed to it being quite sensitive to errors affecting the most significant bits of input data and the 1-bit only weights. In contrast, we expect increased robustness from the bitstream processing paradigm. Hence, bit-flip error injection has been considered to measure the robustness of the proposed bitstream processing framework to errors that can happen during memory

accesses of images and weights [4, 8]. Since binary symbol errors affect the weights and pixel values differently, our study investigates those two kinds of errors separately.

For the case in which errors affect pixel values, some binary symbols representing the pixels are randomly selected. The pixels of test images are manipulated in 8-bit traditional binary representation for deterministic BNN. For BSBNN, the image data is represented in $N = 8$ (BSBNN8) or $N = 16$ (BSBNN16) sized streams. Fig. 7*a* depicts how network accuracy decreases with the percentage of flipped symbols. We observe that the conventional processing paradigm suffers from dramatic accuracy losses while the BSBNN achieves remarkable robustness. Regarding binary weights flipping, we observe in Fig. 7*b* that BNN and BSBNN8 achieve close performance, while BSBNN16 demonstrates a non-negligible, but significantly reduced sensitivity to weight bit errors.

As the second round of simulations, we have investigated how accounting for the errors and the processing paradigm at training time affects the robustness to errors at the test time. Therefore, different models have been trained with different bit-flipping rates (5, 10, or 20%) at training. This is done using either the conventional or the bitstream processing computational paradigm, and the errors affect either the image pixels or the weights. The resulting models are tested with the same computational paradigm than the one used at training, and their test accuracy is reported in Fig. 7*c* and *d*, as a function of the bit-flip percentage at test time. In Fig. 7*c*, we observe that injecting errors on image pixels at training increases the robustness to errors at test time by a large margin (about 30% accuracy improvement at 30% error rate) and that the bitstream processing paradigm is significantly more robust to errors than the conventional one. With regard to the errors affecting weights, we observe in Fig. 7*d* that accounting for errors is largely beneficial when they are also present at test time but the performance is penalized in absence of errors at test time.

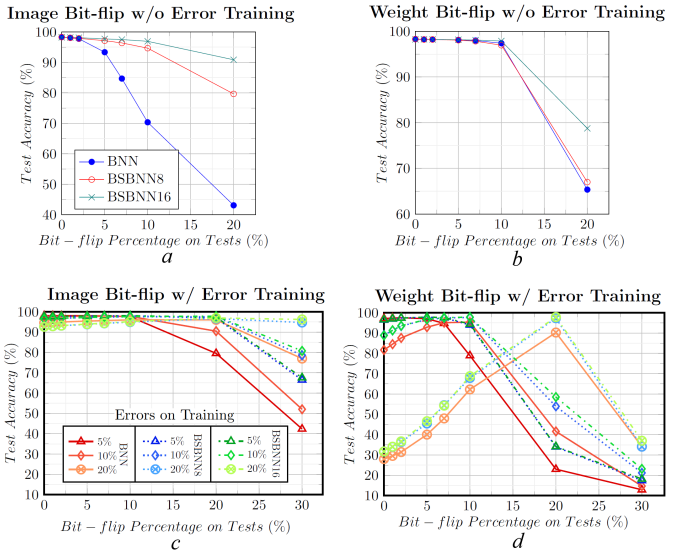

Fig. 7 *Bit-flip soft error injections applied to BNN, BSBNN8, and BSBNN16. Error injection is defined in terms of the percentage of flipped binary symbols.*
a BNN model trained without error is tested for image bit-flips
b BNN model trained without error is tested for weight bit-flips
c&d Models are trained in the presence of (image or weight) errors, with the processing paradigm used at test time.

To compare the complexity of the conventional and bitstream-based computational paradigms, a hardware simulation framework was constructed in *MATLAB Simulink* with design primitives from *Xilinx System Generator*. For both BNN and BSBNN hardware simulations, the input layer data were copied from *MATLAB* workspace. BPE-based random bitstreams were prepared on the software side, and they adopted a Bernoulli distribution for BSBNN following the procedure in [4]. Parallel co-simulation was performed by feeding the hardware model associated with the feed-forward hidden layers. For single neuron hardware shown in Fig. 4, the total count of the look-up table and flip-flop logic was 101 for $N = 8$-bit BSBNN, and 147 for BNN, which represents a gain of approximately 30%, according to the design synthesis in the *Vivado* tool setting the target device as *Zynq-7000*.

Conclusion: In this letter, a stochastic bitstream processing binarized neural network is presented. The first contribution of the study is proposing subtraction-free activation using the truncated MOD-T counter in the presence of bitstream usage. The second contribution is underlining data error robustness of bitstream processing over the deterministic approach. Injecting random bit-flips into the data of deterministic BNN decreases the accuracy values significantly. Stochastic bitstream representation of image pixels and weights provides noticeable robustness for soft errors on data.

Acknowledgment: This Ph.D. dissertation is supported by ITU BAP with grant MDK-2018-41532. The study is being collaboratively continued in UCLouvain, ICTEAM during Sercan Aygun's Ph.D.-related research visit in the group lead by Prof. C. De Vleeschouwer.

S. Aygun and E. O. Gunes (*Department of Electronics and Communication Engineering, Istanbul Technical University (ITU), 34469, Istanbul, Turkey*)

E-mail: ayguns@itu.edu.tr

C. De Vleeschouwer (*ICTEAM, Université Catholique de Louvain (UCLouvain), B-1348, Louvain-la-Neuve, Belgium*)

S. Aygun: also, with the Department of Computer Engineering, Yildiz Technical University, Istanbul, Turkey

References

1 Miller, A.J., Brown, A.W., Mars, P.: 'Adaptive logic circuits for digital stochastic computers' *Electronics Letters*, 1973, **9**, (21), pp. 500-502, doi:10.1049/el:19730370
2 Li, B., Najafi, M.H., Lilja, D.J.: 'Low-cost stochastic hybrid multiplier for quantized neural networks' *ACM, J. Emerg. Technol. Comput. Syst.*, 2019, **15**, (2), doi:10.1145/3309882
3 Alaghi, A.: 'The logic of random pulses', Ph.D. thesis, University of Michigan, 2015
4 Canals, V., Morro, A., Oliver, A., *et al.*: 'A new stochastic computing methodology for efficient neural network implementation', *IEEE Trans. on Neural Networks and Learning Sys.*, 2016, **27**, (3), pp. 551-564, doi:10.1109/TNNLS.2015.2413754
5 Ren, A., Li, Z., Ding, C., *et al.*: 'SC-DCNN: Highly-scalable deep convolutional neural network using stochastic computing', *ACM SIGARCH Comp. Arch. News*, 2017, **45**, (1), pp. 405-418, doi:10.1145/3093337.3037746
6 Ardakani, A., Leduc-Primeau, F., Onizawa, N., *et al.*: 'VLSI implementation of deep neural network using integral stochastic computing' *IEEE Trans. on VLSI Sys.*, 2017, **25**, (10), doi:10.1109/TVLSI.2017.2654298
7 Courbariaux, M., Hubara, I., Soudry, D., *et al.*: 'Binarized neural networks: training deep neural networks with weights and activations constrained to +1 or -1', 2016, *arXiv*: 1602.02830
8 Hirtzlin, T., Penkovsky, B., Bocquet, M., *et al.*: 'Stochastic computing for hardware implementation of binarized neural networks' *IEEE Access*, 2019, **7**, doi:10.1109/ACCESS.2019.2921104
9 Xiao, H., Rasul, K., Vollgraf, R.: 'Fashion-MNIST: a novel image dataset for benchmarking machine learning algorithms', 2017, *arXiv*:1708.07747
10 Godse, A.P. and Bakshi, U.A.: 'Digital and linear integrated circuits', *Technical Publications*, Chapter-5, "Counters" 2009, ISBN:9788184316827
11 LeCun, Y., Cortes, C., Burges, CJ.: 'MNIST handwritten digit database', *AT&T Labs*, 2010, [Online]. Available: http://yann.lecun.com/exdb/mnist
12 Liu, Y., Wang, Y., Lombardi, F., Han, J.: 'An energy-efficient stochastic computational deep belief network', 2018 DATE Conf., Dresden, Germany, pp. 1175-1178, doi:10.23919/DATE.2018.8342191
13 Sanni, K., Garreau, G., Molin, J.L., Andreou, A.G.: 'FPGA implementation of a Deep Belief Network architecture for character recognition using stochastic computation', 49^{th} Annual Conf. on Info. Sciences and Sys., Baltimore, MD, USA, 2015, pp. 1-5, doi:10.1109/CISS.2015.7086904
14 Babu, A.V., Lashkare, S., Ganguly, U., Rajendran, B.: 'Stochastic learning in deep neural networks based on nanoscale PCMO device characteristics', *Neurocomputing*, 2018, **321**, pp. 227-236, doi:10.1016/j.neucom.2018.09.019
15 Li, B., Najafi, M.H., Lilja, D.J.: 'An FPGA implementation of a Restricted Boltzmann Machine classifier using stochastic bit streams', IEEE 26^{th} ASAP Conf., Toronto, Canada, 2015, doi:10.1109/ASAP.2015.7245709
16 Kim, K., Kim, J., Yu, J., *et al.*: 'Dynamic energy-accuracy trade-off using stochastic computing in deep neural networks', 53^{rd} DAC, Austin, TX, USA, 2016, doi:10.1145/2897937.2898011
17 Xie, Y., Liao, S., Yuan, B., *et al.*: 'Fully-parallel area-efficient deep neural network design using stochastic computing' *IEEE Trans. on Circ. and Sys. II: Expr. Br.*, 2017, **64**, (12), pp. 1382-1386, doi:10.1109/TCSII.2017.2746749
18 Liu, Y., Liu, S., Wang, Y., *et al.*: 'A Stochastic computational multi-layer perceptron with backward propagation', *IEEE Trans. on Comp.*, 2018, **67**, (9), pp. 1273-1286, doi:10.1109/TC.2018.2817237