NOLTA, IEICE

Paper

Robustness of hardware-oriented restricted Boltzmann machines in deep belief networks for reliable processing

Kodai Ueyoshi[1 a)], *Takao Marukame*[2], *Tetsuya Asai*[1], *Masato Motomura*[1], *and Alexandre Schmid*[2]

[1] *Graduate School of Information Science and Technology, Hokkaido University Kita 14, Nishi 9, Kita-ku, Sapporo, Hokkaido 060-0814, Japan*

[2] *Microelectronic Systems Laboratory, École Polytechnique Fédérale de Lausanne, CH-1015, Lausanne, Switzerland*

[a)] *ueyoshi@lalsie.ist.hokudai.ac.jp*

Received February 10, 2016; Revised April 25, 2016; Published July 1, 2016

Abstract: Remarkable hardware robustness of deep learning is revealed from an error-injection analysis performed using a custom hardware model implementing parallelized restricted Boltzmann machines (RBMs). RBMs used in deep belief networks (DBNs) demonstrate robustness against memory errors during and after learning. Fine-tuning has a significant impact on the recovery of accuracy under the presence of static errors that may modify structural data of RBMs. The proposed hardware networks with fine-graded memory distribution are observed to tolerate memory errors, thereby resulting in a reliable deep learning hardware platform, potentially suitable to safety-critical embedded applications.

Key Words: deep learning, restricted Boltzmann machines (RBMs), fault tolerance

1. Introduction

Deep learning (DL) algorithms have been devoted a growing attention, owing to impressive performances recently demonstrated in acceptedly difficult applications, *e.g.*, classification and prediction, exceeding the performances of conventional machine learning algorithms [1–3]. Nevertheless, DL algorithms require significant computational time and dissipate large amounts of power in order to reach a state where the data structure is optimized with a set of parameters that are suitable to obtain intelligent operations. Consequently, the execution of learning algorithms is considered excessively slow, specifically in the case of large-scale networks, *de facto* limiting the focus to small-scale models, or to the use of training sets of small size. Iterative updates of connection weights of the network use over the major part of the learning time. In practical cases, training of DL has been performed by cloud servers or high-performance graphics processing units (GPUs) [4]. Hence, efficient and reliable hardware systems are required to satisfy the growing needs of DL in big data analysis, autonomous system control and cognitive applications. Developments of field programmable gate array (FPGA)

DOI: 10.1587/nolta.7.395

or application specific integrated circuit (ASIC) technologies tailored to support DL algorihtms are expected to accelerate the studies and applications of DL and machine learning, in general [5–9].

Unsupervised DL algorithms aim at finding some structure in the data and clustering individual data into groups. The restricted Boltzmann machine (RBM) model is a prominent representative of unsupervised DL algorithms. In turn, RBMs can be used as building blocks of hybrid deep network, such as deep belief networks (DBNs). The hardware implementation of RBMs has been the focus of several recent studies, [4–8]. An original architecture supporting scalable and highly parallel RBM microelectronic systems has been developed, and forms the core hardware used in this study [9]. In RBM structures that are organized layer-by-layer, the stochastic process of the iterative learning efficiently reduces computational time, although all the learned data represented as connection weights and node biases should be stored in a suitable memory and updated during the learning sequence. During learning or feed-forward calculations phases, any error occurring in the memory results in degraded performance of DBN-based applications. Consequently, robustness to memory errors is expected in embedded and safety-critical applications which will make use future memory generations subjected to expectedly decreasing fabrication yield. Robustness to memory errors is also a factor of cost reduction by enabling low redundancy of memory at the integrated circuit and system levels [18, 19].

The structural construction of RBMs is expected to provide robustness against memory errors, owing to network topologies that are generally redundant. Earlier, studies have considered the potential impact of degradation or limitations in some of the parameters used in the representation of neural networks, *e.g.*, reduced bit-precision in [10], noise in the input set in [11], discretization noise and effect of event-driven simulation in [12]. However, the detailed fault-tolerance of DBNs assuming a highly scalable hardware system where memory or memory transfers may induce errors has never been studied. This work analyzes the dependence of accuracies to faults that are injected into a memory block that stores structural data consisting of weights and biases. The research presented in this paper is conducted in a general context aiming at developing reliable hardware made from unreliable components, specifically targeting the integration of machine learning towards its generalized usage in cheap consumer products. Hence, the final objective consists of developing hardware and algorithms that mutually support each other's potential defectiveness for all blocks of an integrated system, including the datapath and control, the memory and communication channels. In this specific context, we consider that the learning and the fine-tuning are processed on-chip, which also supports the assumption of future autonomous fault-tolerant machine-learning based processing.

This study focuses on the impact of structural data degradation that is stored in a faulty memory. Results of error-injection analysis after RBM learning and fine-tuning are reported, and demonstrate strong robustness against errors that originates from the hardware network construction. A hardware system processing the DBN and implemented on a FPGA platform, as well as its model are used to this purpose. Section 2 presents the developed hardware architecture supporting RBM integration. Section 3 presents the fault models that have been considered, the fault-tolerance analysis method that has been applied and discusses the results.

2. RBM hardware and modeling for DBNs

RBMs obey a stochastic neural network model consisting of two layers, namely a visible and a hidden layer. The network consists of an undirected graph model in which neurons in one layer are fully connected to neurons in the complementary layer. RBMs are tuned using three parameters, *i.e.*, the connection weights, visible biases, and hidden biases. The learning process is unsupervised and results in the update of the parameters. The equations representing the update originate from contrastive divergence (CD) learning [1, 2]. The RBM calculation flow consists of two repeating steps. In the first step, input data is delivered to the visible layer, and the hidden layer output is calculated using the visible layer's values as input. At this point, all visible layer and hidden layer combinations are calculated. In the second step, the visible layer is calculated, using sampling results from the hidden layer as input. The update equation is approximated by consecutive repetition of these steps. The

Algorithm 1: RBMs training pseudo-code

```
input  : RBM($v_1, ...., v_m, h_1, ...., h_n$), Learning
         rate $\alpha$
output: parameters $w_{ij}, b_i, c_j$
        ($i = 1, ...., m, j = 1, ...., n$)
1  init $w_{ij}$ = rand($-1/m \sim 1/m$), $b_i = c_j = 0$
2  $\mathbf{v}^{(0)} \leftarrow \mathbf{v}$
3  for $t = 0$ to $k$ do
4      if $t \neq 0$ then
5          for $i = 1$ to $m$ do
6              for $j = 1$ to $n$ do
7                  $\text{tmp}+ = {h_j}^{(t)} * w_{ij}$
8              $P({v_i}^{(t)}) = \text{sigmoid}(\text{tmp} + b_i)$
9              ${v_i}^{(t)} = P({v_i}^{(t)}) > \text{rand}(0 \sim 1)$
10     for $j = 1$ to $n$ do
11         for $i = 1$ to $m$ do
12             $\text{tmp}+ = {v_i}^{(t)} * w_{ij}$
13         $P({h_j}^{(t)}) = \text{sigmoid}(\text{tmp} + c_i)$
14         ${h_j}^{(t)} = P({h_j}^{(t)}) > \text{rand}(0 \sim 1)$
15 for $i = 0$ to $m$, $j = 0$ to $n$ do
16     $w_{ij}+ = \alpha * ({v_i}^{(0)} * P({h_j}^{(0)}) - {v_i}^{(k)} * P({h_j}^{(k)}))$
17 for $i = 0$ to $m$ do
18     $b_i+ = \alpha * ({v_i}^{(0)} - {v_i}^{(k)})$
19 for $j = 0$ to $n$ do
20     $c_j+ = \alpha * (P({h_j}^{(0)}) - P({h_j}^{(k)}))$
```

pseudo-code of the learning algorithm is presented in Algorithm 1. Lines 3 through 14 show the repeating steps processed to obtain the sample required for the update, and lines 15 through 20 are the update sequence.

This learning algorithm has been demonstrated in [9], and the corresponding architecture has been implemented on FPGA. This algorithm and architecture are used in the present study to perform the analysis of memory-error tolerance.

Deep networks are constructed by stacking RBMs in a layer-by-layer manner. A hidden layer that has completed learning is used as a visible layer for the RBM located in the next layer. The DBN configuration is completed by applying back-propagation to perform fine-tuning.

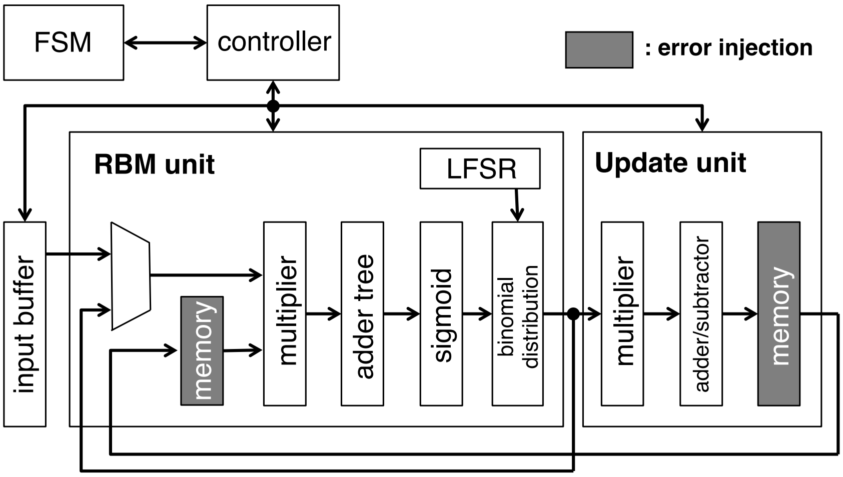
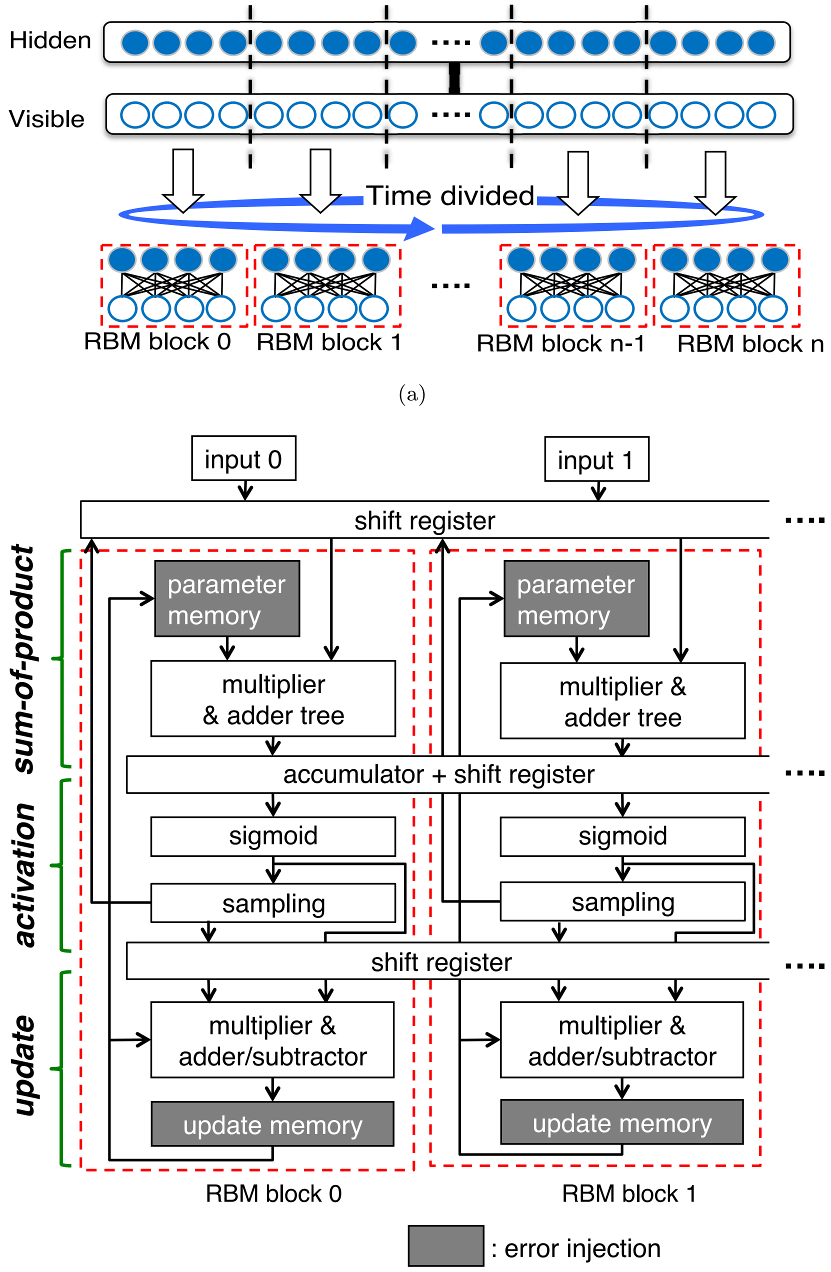


Fig. 1. Overall data-flow of the architecture. The memory blocks into which errors are injected are shown as gray-shaded blocks.

Figure 1 shows the overall data-flow and the proposed architecture. Input data is delivered from the input buffer to the RBM unit, where CD learning computation is repeatedly carried out. The learning update equation is simultaneously processed in the update unit, and the results stored in the unit's local memory. When a learning process is completed, the learning data is moved from the local memory of the update unit to the local memory of the RBM unit. These operations are controlled by a common finite-state machine (FSM) controller, and a linear feedback shift register (LFSR). Input data is assumed to consist of unsigned 8-bit fixed-point numbers represented as continuous values ranging from 0 through 1. Connection weights are signed 16-bit fixed-point numbers, and arithmetic unit results are rounded to signed 16-bit fixed-point numbers.

As an example, RBMs are constructed from blocks consisting of four hidden and four visible neurons, and their interconnections. Figure 2(a) shows a conceptual diagram of a RBM of size n neurons in both



(a)

(b)

Fig. 2. RBM hardware algorithm and corresponding implementation architecture. (a) Fine-graded RBM blocks (0 to n) defined from an original network with hidden and visible nodes. The example of unit network size of 4×4 (connections) is shown. (b) Block-diagram of the proposed architecture. RBM blocks consist of sum-of-product, activation, and update calculations, which are connected with shift registers. The parameter memories and update memories are modeled as a target of error injections, and are depicted as gray-shaded blocks.

the visible and hidden layers, which is partitioned into blocks of 4 neurons to reduce circuit resources allocated to connections between the two layers. Hence, the resulting number of connections in an RBM block is equal to 4×4. This block-partitioned architecture calculates all connections, thereby supporting full connectivity between the visible and hidden layers, by the means of a time-division processing that makes use of shift registers for its physical implementation, as well as additional control logic. The circuit details are reported in [9]. As a benefit of this approach, the number of connections to process, and thus the time and arithmetic complexity are linear with respect to the number of neurons.

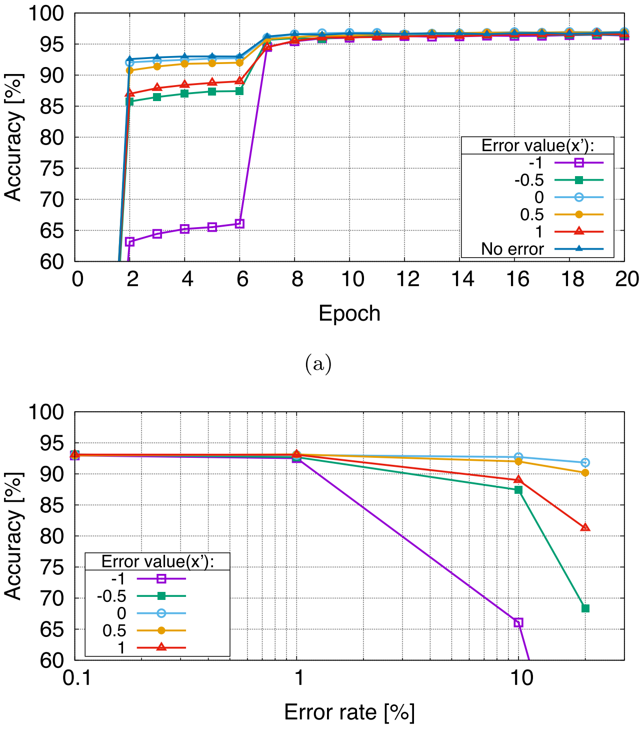
Figure 2(b) presents the detail block-diagram of the partitioned and parallel implementation of the algorithm, focused on the first two blocks. The block size, *i.e.*, the number of neurons in the layers that are processed by the block can be reconfigured to larger sizes such as 8, 16, and 32. The process is divided into three phases, and the hardware serving each phase is separated from the others by a shift register. These registers store and propagate data from or into each block, and also take the role of data storage in a pipelined realization. In the first *sum-of-products* phase, each RBM block multiplies all the inputs and connection weights in parallel, and calculates the respective outputs using an adder tree. This approach is possible because each RBM block is configured on a small scale. In each RBM block, a local memory saves parameters which are only used inside the specific block. The sigmoid calculation and binomial distribution calculation are performed in the next phase of *activation*. The approximate sigmoid calculation is obtained using a piecewise linear function. A binomial distribution calculation is also obtained from a dedicated circuit that determines the output as a binary value, logic 0 or 1, by comparing and thresholding the input with respect to random numbers delivered by the LFSR. In the last phase of *update*, the parameter update calculation is performed. The update unit has a completely parallel architecture to receive and process data that accommodates the parallel processing of RBM blocks. In addition, the architecture of the update unit is similar to the architecture of the sum-of-products unit, since both carry out processing over the entire set of parameters. The update unit contains a local memory that stores the update data for the RBM unit; these values are constantly updated during the learning process.

The structure of a DBN consists of multiple layers of artificial neural networks. Typically, a DBN consisting of three stages of RBMs represents an accepted configuration in character recognition applications, [1, 2]. DBNs exploit the probability values of RBMs in order to propagate extracted features from the 1st (256 nodes) / 2nd (256) / 3rd (256) layers to the final classifier consisting of 10 nodes supporting the selected ten-digit character recognition application. The proposed hardware architecture is used in an iterative way in order to emulate DBNs. As a first step, initial parameters are processed which pertain to RBMs in the 1st layer. Subsequently, a 2nd layer of RBMs is processed, using the outputs of the 1st layer as inputs. Layers are created on the top of another following this iterative procedure. Each RBM is trained over 20 epochs, in this study. After all layers are created, supervised learning is performed in the final classifier. In this process, external memory storage is required, which also is the key enabler to the emulation of DBN of arbitrary sizes.

3. Fault tolerance analysis

Defects are expected to occur in many components of processor systems, such as memory, control, datapath and communication channels. Many types of faults may occur, each causing a specific incorrect electrical behavior at the affected node, which may propagate to the output node as an observed error. Complementary metal oxide semiconductor (CMOS)-based integrated systems have experienced a golden age of high reliability until the advent of deep-submicron fabrication technologies. In these technologies, yield has decreased, and the necessity to find economically tractable solutions has fostered a revival of fault-tolerant circuits and yield enhancement techniques, as well as reliability studies. Exact numerical values of yield, reliability, error rates are classified data; nevertheless the relevance of the issue is clearly manifested by an abundant amount of scientific literature, *e.g.*, [13–17].

Fault models are defined from the way in which they modify the parameters of a network, *e.g.*,



(a)

(b)

Fig. 3. DBN classification accuracy as functions of (a) epochs and (b) error rate; faults are injected into the weights and biases that are stored in a memory after three sequences of RBM learning.

stuck-at logic-zero or stuck-at logic-one which modifies the weights and biases. Random noise sources in transistors are a critical factor affecting the reliability of a wide range of CMOS and memory technologies, causing bit-flippings that also depend on the device size, [18]. Soft errors are one of the major concerns in the reliability of large-scale integrations. Understanding the failure modes and quantifying the error rate under natural radiation are primarily crucial for the development of high-density static random access memories (SRAMs), [19]. As a main memory of large-scale computing systems, dynamic random access memories (DRAMs) have been widely used, and have shown a remarkable reliability from the appllication point of view, though mostly owing to the implementation of redundancy and error correction coding (ECC). In modern systems employing DRAMs, however, a growing memory error rate is observed, which depends on the device count; in addition, a significant amount of faults is observed in the memory controllers and transmission channels, [20]. The aforementioned types of faults may affect the neural network itself or the memory in which parameters are temporarily stored. Considering a digital processor that is used to compute the network emulation, faults affecting the parameters modify the calculation results and are static. A network should be trained to sustain faults occurring at any location of the data path. Studies must be carried out in order to identify how the parameters of a network should be modified such that the network may be capable of sustaining a certain amount of faults, *e.g.*, bit-precision, network connectivity and topology, number of neurons and layers. This paper focuses on the impact of faults altering structural data of RBMs which are stored in a memory, namely the weights and biases values. Potential memory defects are modeled into faults, which adverse effect over the learning process is studied.

3.1 Analysis using discontinuous faults

RBMs learn a set of parameters that are stored in a local memory during a pretraining phase. This structural data can be affected by memory errors that may occur at different timings, as shown earlier [21]. In this Section, the robustness of the hardware DBN model is analyzed by applying random fault injection into the weights and biases. Injected faults obey memory faults models that reflect the presence of realistic memory defects. The mixed national institute of standards and technology

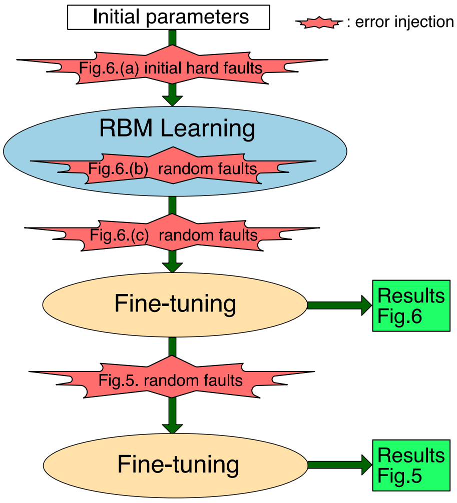
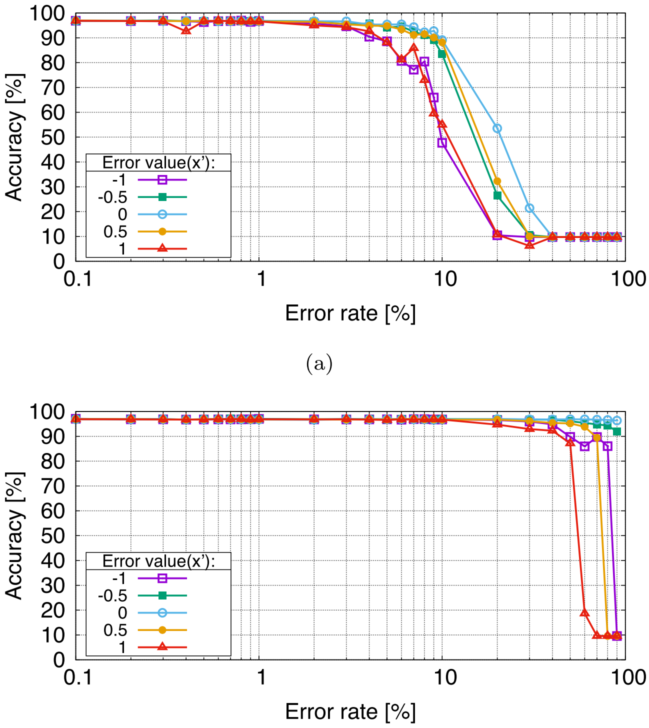


Fig. 4. Flow of hardware bit-flipping analysis, showing the time and sequence models of fault injection.

(MNIST) handwritten dataset is used to analyze classification accuracy [1]. As the parameters learned by the DBN are in the form of a matrix that is attached to each RBM, the fault-injection operation replaces a number of elements of the matrix with a substitute value that obeys the selected fault model. The replaced elements are randomly chosen, and multiple executions enable obtaining statistically relevant results, following the Monte Carlo methodology.

First, faults are injected after the pretraining procedure is completed. The analyzed classification accuracies of the DBN are presented as a function of the learning epochs (Fig. 3(a)) and considering various fault types and rates (Fig. 3(b)). The error rate is defined as the ratio of the number of parameters into which error is injected to the total number of parameters. A C-language model of the RBMs that accurately emulates the hardware that is also implemented on an FPGA is used. A Matlab model is used in the fine-tuning procedure, only. An improvement of the accuracy is observed when fine-tuning is applied after the 6th epoch. In our model, for each memory cell, a datum changes from some value (= x) into a wrong value (= x'), where the assumed error is caused from a mixture of possible memory fault models such as bit-vanishing (stuck-at-zero) and bit-flipping considered at cell and circuit levels. Bit-vanishing can be observed as the burst error in the memory controller or data transfer channels. Bit-flipping may occur when x-rays strike memory cells or when voltage noise is inherently generated by random telegraph noise. Bit-vanishing and bit-flipping of the sign bit cause a change of the sign of the affected value. In our analyses, bit-vanishing is modeled as a severe deletion of all bits storing a value at once, resulting in an incorrect stored value equal to '0'. Bit-flipping of the integer or precision bits creates incorrect random values, *e.g.*, 0.5. Finally, sign-bit errors are modeled by value changes to +1/-1. The *Error value* which parameterizes the results in Fig. 3 and Fig. 5 expresses the incorrect value x'. These fault models modify the electrical operation of the cell in an analog manner, for instance, considering tiny current leakage effects potentially due to the presence of unwanted metal extensions creating a high-impedance short circuit. The DBN maintains a good accuracy of 97% after 15 epochs, even when considering initial faults present in the memory, and in spite of the fact that the accuracy depends on the initial errors before the fine-tuning. From this analysis, memory-fault tolerance is observed with respect to faults caused by (1) vanishing nodes, modeled by a value equal to 0, *e.g.*, x=1 into x'=0, (2) reduction of the stored value, modeled by lowering the absolute value, *e.g.*, x=1 into x'=0.5, and (3) most significant bit (MSB) bit-flipping, modeled by a sign inversion (a positive sign is replaced by a negative sign, and *vice-versa*), *e.g.*, x=1 into x'=-1. This result suggests that a distributed cache memory that is affected by errors can still be used without any error correction, which is a result with fundamental practical impact.

Figure 4 presents the flow of the further analysis and specifies the time and sequence models of fault injection. In a next experimental phase, faults are injected into all the structural data of the DBN after the learning process, which includes fine-tuning. The accuracy is uniform under error



(a)

(b)

Fig. 5. DBN classification accuracy vs. error rate considering various error values. (a) Faults are injected into all the structural data after fine-tuning. (b) Accuracy after relearning with 10 epochs of fine-tuning.

rates up to 10% when the error model consists of x' set to 0, whereas the accuracy significantly reduces when x' is not at 0 (Fig. 5(a)). Figure 5(b) shows recovered accuracies that are obtained by applying a relearning computation sequence from the results shown in Fig. 5(a), specifically applying ten additional epochs of fine-tuning. Consequently, any value of memory faults up to the rate of 20% can be fixed by fine-tuning.

3.2 Hardware bit-flipping analysis

A bit-flipping model is considered for the RBM learning and the classification performance evaluation of the DBN in order to realistically emulate digital hardware faults. Taking the results of the previous analysis into account, a stochastic bit error is specifically considered. With increasing epochs in the normal RBM sequences, the sum-of-errors tends to decrease and be in a low saturated state, toward 20 epochs.

Figure 6 shows the analyzed performance applying various kinds of fault models, namely

(a) initial hard faults (injecting random bits) that are kept in learning as well as subsequently, and which supports the model of process errors that affect all types of memory, as well as design errors that may specifically be encountered in FPGA development; this model is applied in Fig. 6(a);

(b) temporal bit-flipping faults during learning of RBMs and which supports the model of soft errors occurring in the memory or communication channel, and that may be encountered due to radiation (SRAM), noise in transistors (SRAM, NOR-Flash), noise in the signal channel (all memories), data retention (NAND-Flash, resistive random access memory (ReRAM)), Read/Write error (magnetoresistive random access memory (MRAM)); this model is applied in Fig. 6(b);

(c) bit-flipping after learning of RBMs, *i.e.*, before supervised learning; this model is applied in Fig. 6(c).

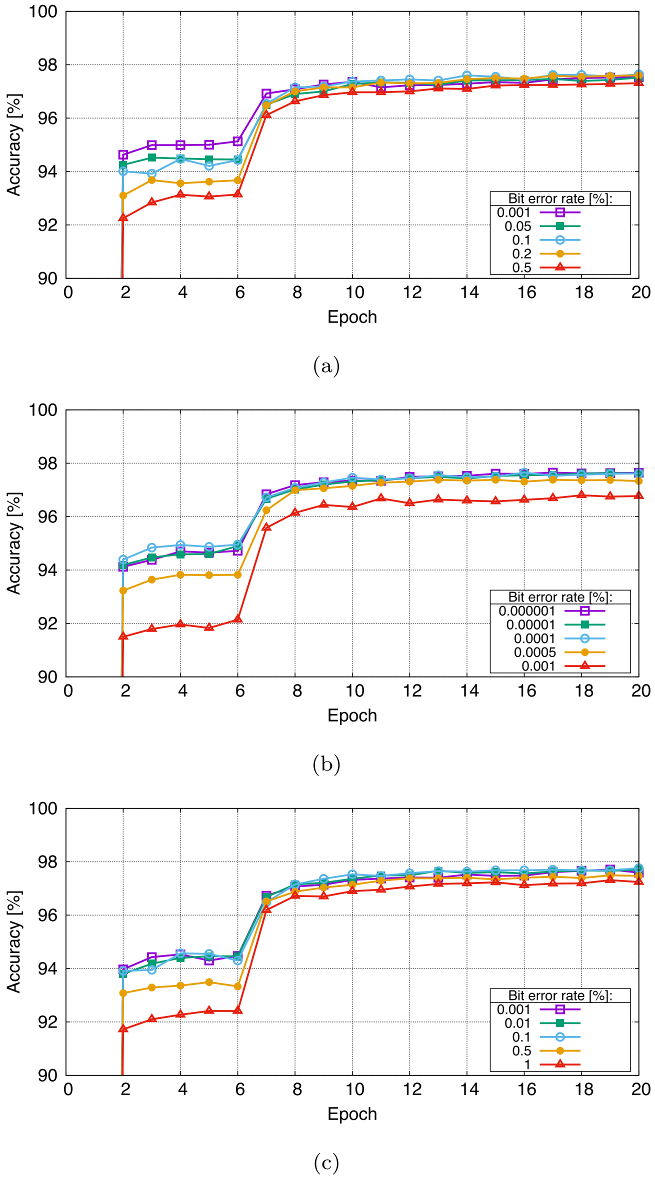


Fig. 6. Fault injection analysis based on a bit-flipping model for (a) initial hard faults, (b) random bit-flipping during learning, and (c) bit-flipping after learning.

These results show that the tolerance to static faults (Fig. 6(a) and 6(c)) is higher than the tolerance to dynamic faults (Fig. 6(b)). When faults are injected during the RBM learning sequences, different tendencies can be observed especially for higher error rates, resulting in much higher sum-of-error values than their initial values. Before starting fine-tuning at the 6th epoch, reasonably high accuracies with low error rates are obtained. An error rate of 0.0005% yields much reduced accuracy, while an error rate of 0.001% does not enable reaching a 90% accuracy, even while using fine-tuning (Fig. 6(b)).

The dependence of the accuracy with respect to the size of RBM blocks (4×4, 8×8, 16×16, 32×32 connections) is further investigated, considering that all parameters of one of the blocks are altered by bit-vanishing. These block errors may be encountered in the memory controller (DRAM, embedded Flash (eFlash)), or due to endurance issues (emerging non-volatile memories, embedded ReRAM (eReRAM) etc.). As presented in Fig. 7, a small network unit exhibits better performance before fine-tuning, whereas a larger-size block shows degradation. However, the accuracy of all network sizes consistently increase when fine-tuning is applied.

Table I summarizes the resource utilisation of the proposed architecture implemented on a commercial FPGA board (DE3 with an Altera Stratix III FPGA). The performance of the proposed

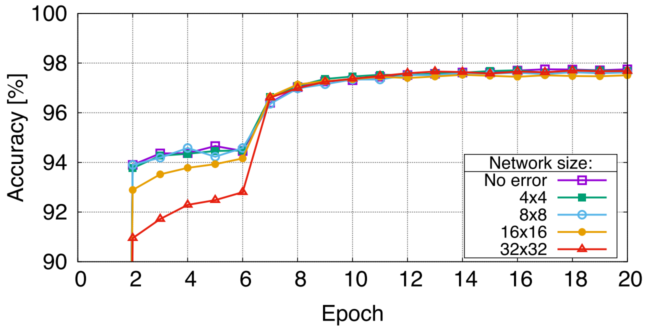


Fig. 7. Fault tolerance analysis dependence on the unit network size (4 × 4, 8 × 8, 16 × 16, 32 × 32) of RBM blocks. All parameters of one of the blocks are altered by bit-vanishing.

Table I. Resource utilization of binary RBMs on FPGA(Stratix III).

Resource	ALUT		Registers	Block memory	18x18 DSP
	Combinational	Memory			
RBM	189579(70.111%)	1024(0.757%)	26880(9.941%)	2490368(14.946%)	0(0.000%)
Controllers	236(0.087%)	7(0.005%)	97(0.036%)	0(0.000%)	0(0.000%)
Total	189815(70.198%)	1031(0.763%)	26977(9.977%)	2490368(14.946%)	0(0.000%)

Table II. Performance summary of RBM on FPGA.

	This work (2016)	Kim et al. (2009)	Ly & Chow (2010)		Kim et al. (2014)
Platform	EP3SL340	EP3SL340	XC2VP70		XC6VSX760
# of chips	1	1	1	4	1
Network	256 × 256	256 × 256	128 × 128	256 × 256	256 × 256
Clock [MHz]	50	200	100		80
GCUPS[a]	16.6(4 × 4)[b] 66.4(16 × 16)[b]	N/A	1.6	3.1	59.6
Benchmark	MNIST handwritten digit	N/A	MNIST handwritten digit		N/A
Accuracy	97%	N/A	N/A		N/A
Errortolerance	Evaluated	N/A	N/A		N/A

[a] CUPS = number of weight / time updates in one learning step.

[b] Size of one RBM block network (16 × 16 is assumed, for estimations).

architecture is compared with other FPGA implementations (Table II). Our hardware shows fast speed in terms of CUPS (connection updates per seconds), even though using a slower FPGA clock rate than other developments. Moreover, the speed can be adapted to unit RBM configurations, as a benefit of the linear scalability of the proposed architecture. The soft error analysis in Fig. 6(b) considers a bit error rate smaller than 10^{-3}%, whereas a typical value of soft error rate of SRAMs equals 10^3 FIT (Failure-In-Time, corresponding to an error rate equal to 10^{-6}%) for all process nodes, [22]. Only this work demonstrates memory-fault tolerance on the implemented hardware, indicating 97% classification accuracy even if the memory suffers from notable errors, larger than 0.1%, Figs. 6(a) and (c).

4. Conclusion

This study first quantifies the robustness of a DBN consisting of custom RBM hardware. in order to study the behavior of the network, errors are randomly injected into the connection weights prior to supervised DBN fine-tuning. The classification accuracy shows remarkable recovery under practical error rates affecting the memory, owing to the fine-tuning sequence. The achieved results confirm a strong robustness against memory errors. Hence, RBM-based DL systems can be implemented in hardware forming an architecture that is expected to be reliable by construction.

Acknowledgments

The authors thank Drs. Y. Higashi, Y. Nishi, Y. Mitani, and K. Muraoka of Toshiba Corporation for their invaluable support and suggestions.

References

[1] G. Hinton and R.R. Salakhutdinov, "Reducing the dimensionality of data with neural networks," *Science*, vol. 313, no. 5786, pp. 504–507, 2006.

[2] G. Hinton, S. Osindero, and Y. Teh, "A fast learning algorithm for deep belief nets," *Neural Computation*, vol. 18, no. 7, pp. 1527–1554, 2006.

[3] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, pp. 436–444, 2015.

[4] R. Raina, A. Madhavan, and A.Y. Ng, "Large-scale deep unsupervised learning using graphics processors," *Proc. 26th Annual International Conference on Machine Learning*, pp. 873–880, 2009.

[5] S.K. Kim, L.C. MacAfee, P.L. McMahon, and K. Olukotun, "A highly scalable restricted Boltzmann machine FPGA implementation," *Proc. International Conference on Field Programmable Logic and Applications*, pp. 367–372, 2009.

[6] S.K. Kim, L.C. MacAfee, P.L. McMahon, and K. Olukotun, "A large-scale architecture for restricted Boltzmann machines," *Proc. 18th IEEE Annual International Symposium on Field-Programmable Custom Computing Machines*, pp. 201–208, 2010.

[7] D. Ly and P. Chow, "High-performance reconfigurable hardware architecture for Restricted Boltzmann machines," *IEEE Transactions on Neural Networks*, vol. 21, no. 11, pp. 1780–1792, November 2010.

[8] L.W. Kim, S. Asaad, and R. Linsker, "A fully pipelined FPGA architecture of a factored restricted Boltzmann machine artificial neural network," *ACM Transactions on Reconfigurable Technology and Systems*, vol. 7, no. 5, pp. 1–23, 2014.

[9] K. Ueyoshi, T. Asai, and M. Motomura, "Scalable and highly parallel architecture for restricted boltzmann machines," *Proc. RISP International Workshop on Nonlinear Circuits, Communications and Signal Processing*, pp. 369–372, 2015.

[10] E. Stromatias, D. Neil, M. Pfeiffer, F. Galluppi, S.B. Furber, and S.-C. Liu, "Robustness of spiking Deep Belief Networks to noise and reduced bit precision of neuro-inspired hardware platforms," *Frontiers in Neuroscience*, vol. 9, article. 222, pp. 1–14, 2015.

[11] Y. Tang and C. Eliasmith, "Deep networks for robust visual recognition," *Proc. 27th International Conference on Machine Learning*, 2010.

[12] E. Neftci, S. Das, B. Pedroni, K. Kreutz-Delgado, and G. Cauwenberghs, "Event-driven contrastive divergence for spiking neuromorphic systems," *Frontiers in Neuroscience*, vol. 7, article. 272, pp. 1–14, 2014.

[13] R. Baumann, "The impact of technology scaling on soft error rate performance and limits to the efficacy of error correction," *IEDM Tech. Digest*, pp. 329–332, 2002.

[14] S. Borkar, "Designing reliable systems from unreliable components: The challenge of transistor variability and degradation," *IEEE Micro*, vol. 25, no. 6, pp. 10–16, 2005.

[15] M.A. Alam, K. Roy, and C. Augustine, "Reliability- and Process-variation aware design of integrated circuits — A broader perspective," *Proc. 2011 IEEE International Reliability Physics Symposium (IRPS)*, pp. 4A.1.1–4A.1.11, 2011.

[16] The International Technology Roadmap for Semiconductors, Edition 2013.

[17] R. Aitken, E.H. Cannon, M. Pant, and M.B. Tahoori, "Resiliency challenges in sub-10nm technologies," *Proc. 2015 IEEE 33rd VLSI Test Symposium (VTS)*, pp. 1–4, 2015.

[18] Y. Higashi, N. Momo, H. Momose, T. Ohguro, and K. Matsuzawa, "Comprehensive understanding of random telegraph noise with physics based simulation," *Proc. Symposium on VLSI Technology (VLSIT)*, pp. 200–201, 2011.

[19] J.L. Autran, S. Serre, D. Munteanu, S. Martinie, S. Semikh, S. Sauze, S. Uznanski, G. Gasiot, and P. Roche, "Real-time soft-error testing of 40nm SRAMs," *Proc. 2012 IEEE International Reliability Physics Symposium (IRPS)*, pp. 3C.5.1–3C.5.9, 2012.

[20] J. Meza, Q. Wu, S. Kumar, and O. Mutlu, "Revisiting memory errors in large-scale production data centers: Analysis and modeling of new trends from the field," *Proc. 2015 45th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pp. 415–426, 2015.

[21] T. Marukame, E. Calabrese, and A. Schmid, "Fault tolerance analysis of Restricted Boltzmann Machines in deep learning for embedded biosignal processing," *Proc. International Conference of the IEEE Engineering in Medicine and Biology Society*, August 2015.

[22] P. Shivakumar, M. Kistler, S.W. Keckler, D. Burger, and L. Alvisi, "Modeling the effect of technology trends of the soft error rate of combinational logic," *Proc. International Conference on Dependable Systems and Networks*, pp. 389–398, 2002.