Neuron Fault Tolerance in Spiking Neural Networks

Theofilos Spyrou*, Sarah A. El-Sayed*, Engin Afacan*,
Luis A. Camuñas-Mesa†, Bernabé Linares-Barranco†, Haralampos-G. Stratigopoulos*
*Sorbonne Université, CNRS, LIP6, Paris, France
†Instituto de Microelectrónica de Sevilla (IMSE-CNM), CSIC y Universidad de Sevilla, Sevilla, Spain

***Abstract*—The error-resiliency of Artificial Intelligence (AI) hardware accelerators is a major concern, especially when they are deployed in mission-critical and safety-critical applications. In this paper, we propose a neuron fault tolerance strategy for Spiking Neural Networks (SNNs). It is optimized for low area and power overhead by leveraging observations made from a large-scale fault injection experiment that pinpoints the critical fault types and locations. We describe the fault modeling approach, the fault injection framework, the results of the fault injection experiment, the fault-tolerance strategy, and the fault-tolerant SNN architecture. The idea is demonstrated on two SNNs that we designed for two SNN-oriented datasets, namely the N-MNIST and IBM's DVS128 gesture datasets.**

I. Introduction

Deep Neural Networks (DNNs) for modern applications, e.g. computer vision, speech recognition, natural language processing, etc., comprise a multitude of layers of different types, i.e., convolution, pooling, fully-connected, etc., tens of millions of synaptic weight parameters, and they perform a myriad of operations in a single forward pass. From a hardware perspective, this poses great challenges such as energy-hungry data movement, large memory resources, speed of computation, and scalability. In this regard, there is a need for Artificial Intelligence (AI) hardware accelerators that can support high-dimensional and computationally-intensive AI workloads. While the dominant AI hardware accelerators today are Graphics Processing Units (GPUs) and Field-Programmable Gate Arrays (FPGAs), over one magnitude of cost-energy-performance improvements can be obtained by Application Specific Integrated Circuits (ASICs) [1], [2].

A major preoccupation nowadays is the trustworthiness of AI systems, which involves privacy, avoidance of unfair bias, explainability, resilience to adversarial attacks, dependability, etc. The issue of AI hardware accelerator dependability, which involves reliability, availability, safety, etc., has been overlooked since it is often tacitly assumed that Neural Networks (NNs) in hardware inherit the remarkable fault-tolerance capabilities of biological NNs. This capability stems from massively parallel architectures and overprovisioning. However, recent fault injection experiments in AI hardware accelerators have shown that they can be highly vulnerable to hardware-level faults especially when those are happening after training [3]–[11]. These experiments demonstrate that equipping AI hardware accelerators with a preventative fault tolerance strategy is a crucial requirement for mitigating risks in AI systems.

In this context, standard fault-tolerance techniques for regular Very Large-Scale Integrated (VLSI) circuits can be employed, such as Triple Modular Redundancy (TMR) and Error Correction Codes (ECCs) for memories. However, efficiency can be largely improved by exploiting the architectural particularities of AI hardware accelerators and targeting only those fault scenarios that have a measurable effect on performance [12]. One approach is to perform re-training to learn around faults, but this requires access to the training set and extra resources on-chip, thus it is impractical at chip-level.

In this work, we propose a cost-effective neuron fault-tolerance strategy for Spiking Neural Networks (SNNs). SNNs constitute the third generation of NNs that aim at bridging the gap between the biological brain and machine learning in terms of computation speed and power consumption [2], [13], [14]. In SNNs, communication occurs through spike trains and the information is coded by considering the temporal correlation between spikes, thus reproducing the efficiency observed in the brain. In principle, SNNs can be used for the same applications as conventional level-based Artificial Neural Networks (ANNs), having major advantage for recurrent architectures due to their pseudo-simultaneity property.

Fault injection experiments showing the vulnerability of SNNs to hardware-level faults have been presented in [6], [7], [11]. A built-in self-test strategy is proposed for a biologically-inspired spiking neuron in [7]. To the best of our knowledge, this is the first paper proposing a generic network-level neuron fault tolerance strategy for SNNs.

More specifically, we designed two deep convolutional SNNs for the N-MNIST [15] and IBM's DVS128 gesture [16] datasets. We developed a fault injection framework for analysing the criticality of neuron fault types and locations. Fault injection is accelerated by modeling neuron faults at behavioral-level and by embedding the faulty SNN in a GPU. For the fault modeling part, we rely on the findings of a recent work that performed transistor-level fault simulation at the neuron-level and collected all types of neuron faulty behaviors [11]. Next, we leverage the findings from our fault injection experiments to develop a cost-effective fault tolerance strategy consisting in multiple layers of protection. We propose passive fault-tolerance based on dropout [17] to nullify the effect of certain faults, and several active fault tolerance techniques to detect and recover from the remaining faults.

The rest of the paper is structured as follows. In Section II, we describe the two case studies. In Section III, we describe the neuron model used in our designs. Section IV presents the fault injection framework, followed by the fault injection experiment results in Section V. In Section VI, we present the neuron fault tolerance strategy and the hardware architecture. Finally, Section VII concludes the paper.

II. Case Studies

For our experiment, we designed two deep convolutional SNNs for the classification of the N-MNIST [15] and IBM's

978-3-9819263-5-4/DATE21/©2021 EDAA

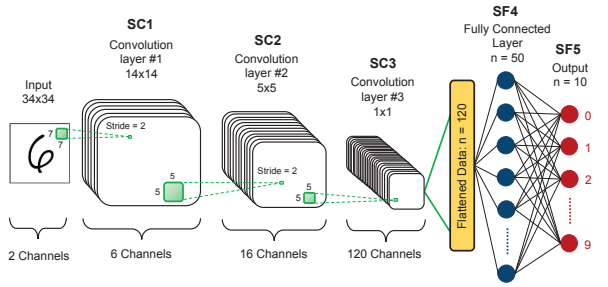

Fig. 1: Architecture of the SNN for the N-MNIST dataset.

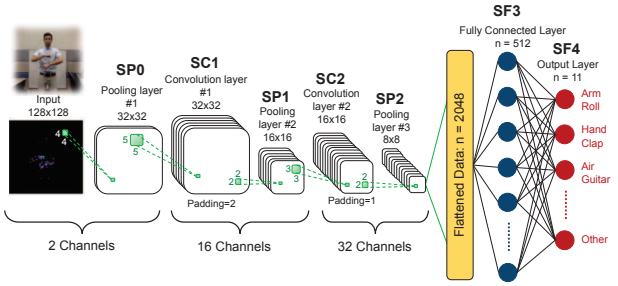

Fig. 2: Architecture of the SNN for the IBM's DVS128 gesture dataset.

DVS128 gesture [16] datasets. Both SNNs are modeled with Python using primitives in the open-source Spike LAYer Error Reassignment (SLAYER) [18] and PyTorch [19] frameworks, and run on a GPU accelerator. They are trained using batch learning with a variation of back-propagation. The winning class is selected after the neuron which is triggered the most, i.e. produces the highest number of spikes.

A. N-MNIST SNN

The N-MNIST dataset is a neuromorphic, i.e., spiking, version of the MNIST dataset, which comprises images of handwritten arithmetic digits in gray-scale format [15]. It consists of 70000 sample images that are generated from the saccadic motion of a Dynamic Vision Sensor (DVS) in front of the original images in the MNIST dataset. The samples in the N-MNIST dataset are not static, i.e. they have a duration in time of 300ms each. The dataset is split into a training set of 60000 samples and a testing set of 10000 samples. The SNN architecture is inspired from the LeNet-5 network [20] and is shown in Fig. 1. The classification accuracy on the testing set is 98.08%, which is comparable to the performance of state-of-the-art level-based DNNs.

B. Gesture SNN

The IBM's DVS128 gesture dataset consists of 29 individuals performing 11 hand and arm gestures in front of a DVS, such as hand waving and air guitar, under 3 different lighting conditions [16]. Samples from the first 23 subjects are used for training and samples from the last 6 subjects are used for testing. In total, the dataset comprises 1342 samples, each of which lasts about 6s, making the samples 20x longer than of those in N-MNIST. Due to computation limitations of the neuromorphic simulation, we trimmed the length of the samples to about 1.5s. We used the architecture proposed in [18], shown in Fig. 2. The network performs with an 82.2% accuracy on the testing set, which is acceptable considering the shortened samples of the dataset and the shallower architecture compared to the architecture in [16].

III. The Spike Response Model

The two SNNs are designed using a generalized form of the ubiquitous Integrate-and-Fire (I&F) neuron model, known as the Spike-Response Model (SRM) [21]. The state of the neuron at any given time is determined by the value of its membrane potential, $u(t)$, and this potential must reach a certain threshold value, ϑ, for the neuron to produce an output spike. The membrane potential of a neuron j in layer l is calculated as:

$$u_j^l(t) = \sum_i \omega_{i,j}^{l-1,l}(\varepsilon * s_i^{l-1})(t) + (v * s_j^l)(t) \tag{1}$$

where $s_i^{l-1}(t)$ is the pre-synaptic spike train coming from neuron i in the previous layer $l-1$, $s_j^l(t)$ is the output spike train of the neuron, $\omega_{i,j}^{l-1,l}$ is the synaptic weight between the neuron and the neuron i in the previous layer $l-1$, $\varepsilon(t)$ is the *synaptic kernel*, and $v(t)$ is the *refractory kernel*.[1]

In Eq. (1), the spiking action of the neuron is described in terms of the neuron's response to the input pre-synaptic spike train and the neuron's own output spikes. The incoming spikes by the neurons in the previous layer are scaled by their respective synaptic weights and fed into the post-synaptic neuron. The response of the neuron to the input spikes is defined by the synaptic kernel $\varepsilon(t)$ which distributes the effect of the most recent incoming spikes on future output spike values, hence introducing temporal dependency. For our experiments, we use the form [18]:

$$\varepsilon(t) = \frac{t}{\tau_s} \cdot e^{(1-\frac{t}{\tau_s})} \cdot H(t) \tag{2}$$

where $H(t)$ is the unit step function and τ_s is the time constant of the synaptic kernel. The second term in Eq. (1) incorporates the refractory effect of the neuron's own output spike train onto its membrane potential through the refractory kernel. The form used here is:

$$v(t) = -2\vartheta \frac{t}{\tau_{ref}} \cdot e^{(1-\frac{t}{\tau_{ref}})} \cdot H(t) \tag{3}$$

where τ_{ref} is the time constant of the refractory kernel.

IV. Fault Injection Framework

A. Fault Modeling

We treat the SNN as a distributed system where neurons are discrete entities that can fail independently. We use the neuron fault model recently proposed in [11] to describe faults at behavioral-level and make fault simulation for deep SNNs traceable. This fault model is generated by performing detailed transistor-level simulations at neuron-level and collecting all types of faulty behaviors. In this way, the fault model becomes independent of the hardware implementation, which helps us draw general conclusions. The most systematic faulty behaviors in [11] are adapted to our neuron model of Section III as follows:

[1]Eq. (1) holds for any neuron no matter the type of the layer; however, in convolutional layers neurons are arranged in a 3-D representation, i.e., width $\times$ height $\times$ channels, and, thereby, we either need to consider a flat indexing or 3-D indexes.

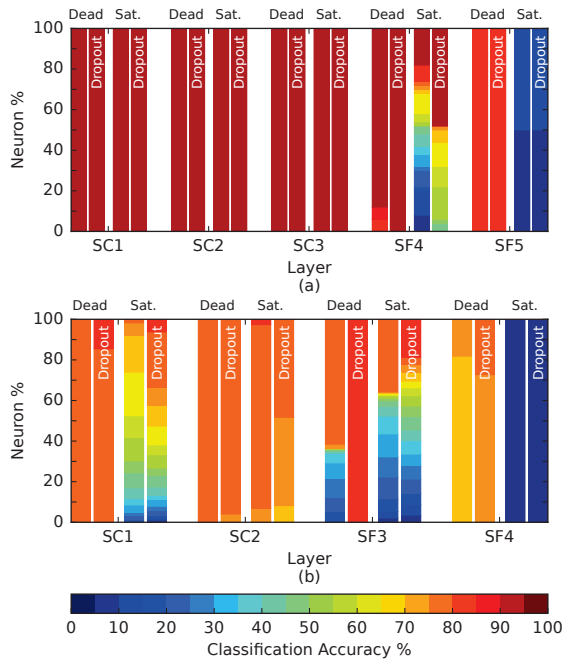

Fig. 3: Effect of neuron faults on classification accuracy with and without dropout: (a) N-MNIST SNN; (b) gesture SNN.

1) Dead Neuron: A fault in the neuron that leads to a halt in its computations and a zero-spike output. This fault is modeled by forcing the output spike train to be always low.

2) Saturated Neuron: A fault that causes the neuron to be firing all the time, even without any external stimuli. This fault is modeled by skipping the computations and forcing the output to be high at every time step.

3) Neuron timing variations: A fault that results in timing variations of the output spike train, i.e. time-to-first-spike and firing rate. This parametric fault is modeled by varying the value of τ_s in Eq. (2).

B. Fault simulation

The fault injection framework is built on top of the SLAYER [18] and PyTorch [19] frameworks. Fault injection and simulation are performed by customizing the flow of computations in the frameworks according to the behavioral modeling of the faults described in Section IV-A. Fault simulation acceleration is achieved first by describing the SNN at behavioral-level and performing fault injection at this level, and second by mapping the faulty SNN on a GPU.

V. Fault Injection Results

We consider fault injection in an already trained SNN and we examine whether the fault impacts inference. The metric used is the classification accuracy drop of the faulty network with respect to the baseline classification accuracy of the fault-free network, computed on the testing set. Herein, we show single fault injection results, but in Section VI-D we will show how the proposed neuron fault tolerance strategy responds effectively to multiple neuron fault scenarios.

The effect of dead and saturated neuron faults on the classification accuracy is shown in Figs. 3(a) and 3(b) for the N-MNIST and gesture SNNs, respectively. The x-axis shows the different layers and for each layer we show four columns each corresponding to a fault type: dead, dead with dropout, saturated, saturated with dropout. Dropout is the proposed passive fault tolerance strategy and how fault injection results change using dropout will be explained in Section VI-A. Pooling layers SPX in the gesture SNN aggregate regions of spikes of their previous convolutional layers and do not contain any neurons, thus they are excluded from the analysis. A column is a colored bar possibly separated into chunks of different colors. Each chunk of the bar corresponds to a specific classification accuracy according to the color shading shown at the bottom of Fig. 3, and the projection on the y-axis shows the percentage of neurons for which the fault results in this classification accuracy.

The effect of neuron timing variations is shown in Figs. 4(a) and 5(a) for the N-MNIST and gesture SNNs, respectively. For a given layer, we vary τ_s of one neuron at a time. Figs. 4(a) and 5(a) demonstrate the per-layer average, minimum, and maximum classification accuracy observed across all faulty neurons for τ_s values expressed in % of the nominal value.

The following brief observations can be made regarding the effect of different neuron fault types:

1) Dead Neurons: Dead neuron faults may impact classification accuracy only for the neurons in the last hidden and output layers. In the output layer, a dead neuron implies always misclassifying samples of the class corresponding to the neuron.

2) Saturated Neurons: Saturation neuron faults, on the other hand, may impact classification accuracy for neurons at any layer, as shown in the gesture SNN. In the output layer, a saturated neuron implies always selecting the corresponding class of the saturated neuron, thus samples from all other classes are always misclassified. We also observe that the effect of saturated neuron faults is magnified for layers that have a smaller number of outgoing synapses, i.e., compare SC1 and SC2 layers in the gesture SNN, where the synapses connecting SC1 and SC2 are much less than those connecting SC2 and SF3.

3) Neuron timing variations: Neuron timing variations have an impact only for the last hidden and output layers, thus for simplicity Figs. 4 and 5 exclude all other layers. For the N-MNIST SNN, large variations in τ_s must occur to observe a drop in the classification accuracy of no more than 10%, i.e. more than 80% reduction for the hidden layer SF4 and more than 50% reduction or 100% increase for the output layer SF5. For the gesture SNN, we observe that this fault type can seriously affect the output layer SF4, while the last hidden layer contributes to significant classification accuracy drop only when τ_s reduces by more than 50%. A smaller τ_s implies a narrower synaptic kernel in Eq. (2), i.e., a decreased integration time window, thus reducing the value that the membrane potential can reach. As a result, the spiking probability is reduced and at the extreme the neuron could end up as a dead neuron.

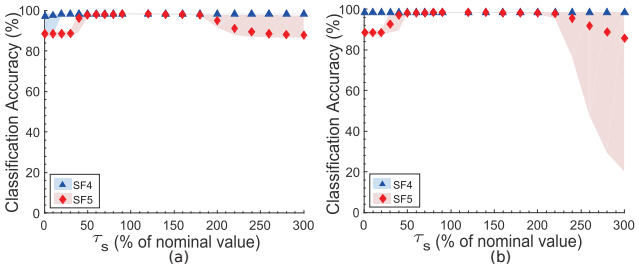

Fig. 4: Effect of neuron timing variations for the N-MNIST SNN: (a) without dropout; (b) with dropout.

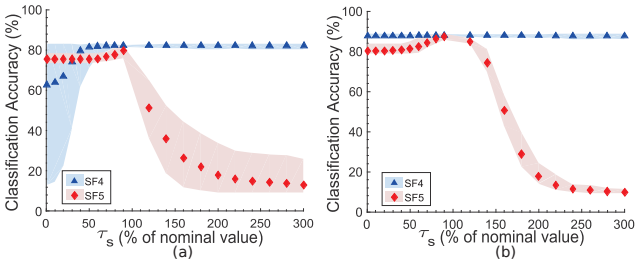

Fig. 5: Effect of neuron timing variations for the gesture SNN: (a) without dropout; (b) with dropout.

Similarly, it can be argued that a higher τ_s increases the spiking probability and at the extreme the neuron could end up as a saturated neuron.

The general conclusions are that saturation neuron faults are the most lethal, and that the impact of all fault types may be severe for the last hidden and output layers.

VI. Neuron Fault Tolerance Strategy

A. Passive fault tolerance using dropout

As a first step, we aimed at implementing a passive fault tolerance such that the SNN is by construction capable of withstanding some faults without any area and power overheads. To this end, we discovered that training the SNN with dropout [17] can nullify the effect of dead neuron faults and neuron timing variations in all hidden layers. In this way, active fault tolerance, which implies area and power overheads, gets simplified since it will need to focus solely on saturation neuron faults in the hidden layers and on all fault types only for the output layer.

The dropout training technique was originally proposed to prevent overfitting and reduce the generalization error on unseen data. The idea is to temporarily remove neurons during training with some probability p, along with their incoming and outgoing connections. At test time, the final outgoing synapse weights of a neuron are multiplied by p. For a network with n neurons, there are 2^n "thinned" scaled-down networks, and training with dropout combines exponentially many thinned network models. The motivation is that model combination nearly always improves performance, and dropout achieves this efficiently in one training session.

For the N-MNIST SNN we used p=10% in the input and SC1 layers, 20% in layers SC2 and SC3, and 50% in layer SF4. Training with dropout resulted in a slight improvement in the classification accuracy from 98.08% to 98.31%. For the gesture SNN we used p=50% only in layer SF3. In this case, dropout increased significantly the classification accuracy from 82.2% to 87.88%.

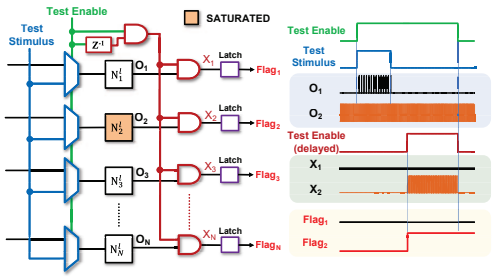

Fig. 6: Offline self-test scheme.

The beneficial effect of dropout on passively nullifying the effect of dead neuron faults is shown for each layer in Figs. 3(a) and 3(b) for the N-MNIST and gesture SNNs, respectively. For example, this is made largely evident by comparing the classification accuracy in the presence of dead faults for layer SF4 of the N-MNIST SNN and layer SF3 for the gesture SNN. The beneficial effect on nullifying the effect of neuron timing variations for the last hidden layer even for extreme variations of τ_s is shown in Figs. 4(b) and 5(b) for the N-MNIST and gesture SNNs, respectively. As can be seen, variations in τ_s from 1% to 300% have now no effect.

The reason behind this result is that dropout essentially equalizes the importance of neurons across the network, resulting in more uniform and sparse spiking activity across the network. Therefore, if a neuron in a hidden layer becomes dead or shows excessive timing variations, this turns out to have no effect on the overall classification accuracy. On the contrary, dropout may magnify the effect of saturation neuron faults, i.e. layer SF3 of the gesture SNN. Finally, we observe that dropout does not compensate for faults in the output layer since in this layer there is one neuron per class and any fault will either overshadow this class or cause it to dominate the other classes.

B. Active fault tolerance in hidden layers

As explained in Section VI-A, active fault tolerance in hidden layers needs only to address neuron saturation. We propose two self-test schemes for neuron saturation detection, namely an offline scheme that can run during idle times of operation and an online scheme that can run concurrently with the operation. Regarding the faulty recovery mechanism, we propose the "fault hopping" concept to simplify the hardware implementation, and we propose two recovery mechanisms at neuron-level and system-level.

1) Offline self-test: The offline self-test scheme is illustrated in Fig. 6. Neuron saturation is declared based on the neuron's activity in the absence of an input. A multiplexer is assigned to every neuron to switch between self-test and normal operation modes. During normal operation, neurons are receiving inputs from the previous layer through synapses, processing them and propagating them to the next layer. When the test enable signal is on, a short internally-generated current pulse is applied to all the neurons simultaneously as a test stimulus, thus test time for the complete network is very short. The neuron outputs are

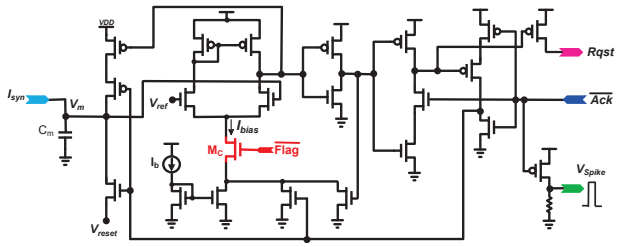

Fig. 7: I&F neuron design showing the recovery operation at neuron-level.

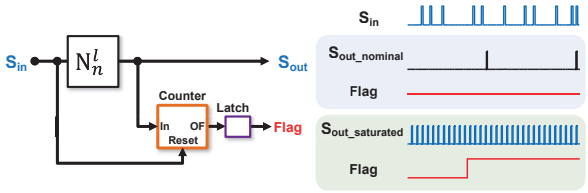

Fig. 8: Online self-test scheme.

paired with a delayed version of the test enable signal through an AND gate. This is to ensure that any activity detected is uncorrelated with the input of the neuron and is indeed a result of saturation. The output of an AND gate going high indicates neuron saturation. This is captured by the latch which raises an error flag signal. A simulation is shown in Fig. 6 using the I&F neuron shown in Fig. 7. This neuron is designed in the AMS $0.35\mu m$ technology, and was originally proposed in [22] as part of a neuromorphic cortical-layer processing chip for spike-based processing systems. This self-test scheme adds a multiplexer, an AND gate, and a latch per neuron, thus the area overhead of the test circuitry is relatively small compared to a single neuron. It can detect aging-induced errors, possibly with some latency.

2) Online self-test: The online self-test scheme is illustrated in Fig. 8. It is applied on a per-neuron basis and takes advantage of the temporal dependency between the input and output of a spiking neuron. In particular, we count the number of spikes a neuron produces after every single input spike using a counter whose reset port is connected to the input of the neuron. In fault-free operation, the neuron needs to integrate multiple input spikes before it can produce a spike of its own, hence the counter is always reset, and the flag signal stays at zero. On the other hand, a saturated neuron will produce spikes with higher frequency than usual, causing the counter to overflow before an incoming spike resets it again. A latch is set when overflow happens and an error flag is raised and maintained. Based on our simulations, 2^3 uncorrelated spikes clearly indicate saturation, thus it suffices to use a 3-bit counter. Fig. 8 shows a simulation using the I&F neuron of Fig. 7. This online self-test scheme entails an area overhead comprised of a counter and a latch per neuron. All neurons are monitored individually and neuron saturation is detected in real-time.

3) Recovery Mechanism: The recovery mechanism is activated once neuron saturation is detected. We propose the concept of "fault hopping" where the critical saturation neuron fault is artificially translated to a dead neuron fault. The network is repaired since a dead neuron fault has no effect after dropout.

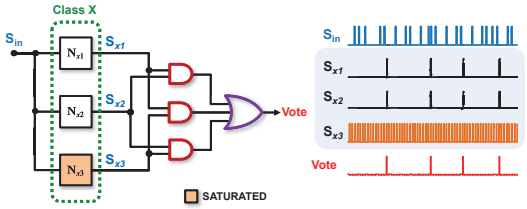

Fig. 9: TMR at the output layer.

This approach leads to an elegant hardware implementation and saves significant costs as opposed to the standard approach, which is to duplicate or triplicate neurons, or provision the SNN with spare neurons that are kept "fresh" and switch the connections of a detected saturated neuron to a spare neuron [12]. We propose two recovery mechanisms based on the concept of "fault hopping", at neuron-level and at system-level.

Neuron-level recovery is implemented by switching-off the saturated neuron. For example, for the I&F neuron in Fig. 7, this can be achieved by connecting a single extra transistor M_C in the tail part of the comparator inside the neuron. This transistor is controlled by the neuron error flag signal. When the neuron gets saturated, the biasing connection of the comparator is suddenly ceased, which deactivates the neuron tying its output to zero. The area overhead is only one transistor per neuron and an auxiliary advantage is that faulty neurons get deactivated; thus, they stop consuming power.

System-level recovery is implemented by setting the outgoing synapse weights of the saturated neuron to zero. In this way, the saturated spike train gets trapped and does not propagate to neurons in the next layer. Typically, the communication between neurons in SNNs is performed by a controller that implements the Address-Event-Representation (AER) protocol [23]. AER controllers perform multiplexing/demultiplexing of spikes generated from or delivered to all neurons in a layer onto a single communication channel. Rather than delivering the actual spike, the controller encodes the address of the neuron that spiked and translates it into the addresses of the destination neurons, and then the weights corresponding to every synaptic connection are loaded accordingly. By leveraging this operation, the system-level recovery approach is based on equipping the controller with the ability to recognize the neuron error flag and update the outgoing synaptic weights to zero. This system-level recovery mechanism has a minimal area overhead since it is reused across all neurons. However, compared to the neuron-level recovery mechanism, saturated neurons stay on continuing consuming power.

C. Active fault tolerance in the output layer

As for the most critical output layer, we propose to directly use TMR for a seamless recovery solution from any single fault type. In particular, a group of three identical neurons vote for the decision of a certain class, as shown in Fig. 9. The voter is a simple 4-gate structure that propagates the output upon which the majority of neurons agree. This means that a faulty neuron in the group, be it dead, saturated or showing excessive timing variations, is outvoted and bypassed. Performing TMR only in the last layer will result in negligible increase in the

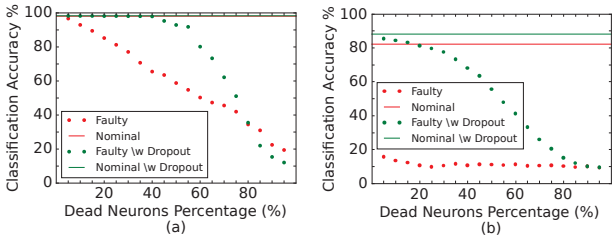

Fig. 10: Fault tolerance for multiple fault scenarios: (a) N-MNIST SNN; (b) gesture SNN.

area and power overhead and a reasonable overhead to pay to ensure strong fault tolerance. The reason is that the number of neurons in the output layer is typically small compared to the size of the whole network. For example, in the N-MNIST SNN, the output layer neurons account for about 0.57% of the neurons in the whole network. This percentage gets even less for the more complicated gesture SNN, where the output layer represents around 0.04% of the total number of neurons.

D. Multiple fault scenario

So far, we have discussed fault tolerance considering a single fault assumption. Moreover, our experiments have shown that neuron timing variations start having an effect when the neuron approaches a dead or a saturated one, and our proposed fault tolerance strategy suggests turning a saturated neuron into a dead one. Hence, all faults eventually fold back to a dead neuron fault, arising the question of what percentage of dead neuron faults can the network withstand. Figs. 10(a) and 10(b) show the classification accuracy as a function of the percentage of dead neurons considering the last hidden layer, which is the most critical amongst all hidden layers, for the N-MNIST and gesture SNNs, respectively. Fig. 10 shows the baseline nominal classification accuracy with and without dropout and how the classification accuracy drops with the increase of dead neuron rate. As the results show, the SNNs employing dropout can withstand larger rates of dead neurons. More specifically, the N-MNIST dataset does not lose any classification accuracy with a dead neuron rate of up to 40%. As for the gesture SNN, the classification accuracy drops faster, but it is still able to perform with over 80% classification accuracy at a dead neuron rate of 20%, which corresponds to 102 neurons.

VII. Conclusions

We presented a cost-effective neuron fault tolerance strategy for SNNs. It leverages findings from large-scale fault injection experiments corroborated on two different deep convolutional SNNs. The fault-tolerance strategy is a two-step procedure. In a first preparatory step, the SNN is trained using dropout which makes some neuron fault types for some layers passive. In a second step, we perform active fault tolerance to detect and recover from the remaining neuron faults in all layers. For hidden layers, we propose offline and online fault detection schemes, a "fault hopping" concept to simplify the error recovery mechanism, and two different neuron-level and system-level recovery mechanisms. For the small output layer we simply use TMR. In terms of future work, we are planning to extend the fault tolerance strategy to include synapse faults.

Acknowledgments

T. Spyrou has a fellowship from the Sorbonne Center for Artificial Intelligence (SCAI). This work has been partially funded by the Penta HADES project and by Junta de Andalucía under contract US-1260118. L. A. Camuñas-Mesa was funded by the VI PPIT through the Universidad de Sevilla.

References

[1] N. P. Jouppi et al., "In-datacenter performance analysis of a tensor processing unit," in *ACM/IEEE International Symposium on Computer Architecture*, 2017.
[2] M. Davies et al., "Loihi: A neuromorphic manycore processor with on-chip learning," *IEEE Micro*, vol. 38, no. 1, pp. 82–99, 2018.
[3] G. Li et al., "Understanding error propagation in deep learning neural network (DNN) accelerators and applications," in *International Conference for High Performance Computing, Networking, Storage and Analysis*, 2017.
[4] B. Reagen et al., "Ares: A framework for quantifying the resilience of deep neural networks," in *ACM/ESDA/IEEE Design Automation Conference*, 2018.
[5] J. J. Zhang et al., "Fault-tolerant systolic array based accelerators for deep neural network execution," *IEEE Design & Test*, vol. 36, no. 5, pp. 44–53, 2019.
[6] E. Vatajelu et al., "Special session: Reliability of hardware-implemented spiking neural networks (SNN)," in *IEEE VLSI Test Symposium*, 2019.
[7] S. A. El-Sayed et al., "Self-testing analog spiking neuron circuit," in *International Conference on Synthesis, Modeling, Analysis and Simulation Methods and Applications to Circuit Design*, 2019.
[8] F. F. d. Santos et al., "Analyzing and increasing the reliability of convolutional neural networks on GPUs," *IEEE Transactions on Reliability*, vol. 68, no. 2, pp. 663–677, 2019.
[9] M. A. Neggaz et al., "Are CNNs reliable enough for critical applications? An exploratory study," *IEEE Design & Test*, vol. 37, no. 2, pp. 76–83, 2020.
[10] L.-H. Hoang et al., "FT-ClipAct: Resilience analysis of deep neural networks and improving their fault tolerance using clipped activation," in *Design, Automation & Test in Europe Conference*, 2020, p. 1241–1246.
[11] S. A. El-Sayed et al., "Spiking neuron hardware-level fault modeling," in *IEEE International Symposium on On-Line Testing and Robust System Design*, 2020.
[12] C. Torres-Huitzil and B. Girau, "Fault and error tolerance in neural networks: A review," *IEEE Access*, vol. 5, pp. 17322 – 17341, 2017.
[13] W. Maass, "Networks of spiking neurons: The third generation of neural network models," *Neural Networks*, vol. 10, no. 9, pp. 1659–1671, 1997.
[14] L. A. Camuñas-Mesa et al., "Spiking neural networks and their memristor-CMOS hardware implementations," *Materials*, vol. 12, no. 17, pp. 2745, 2019.
[15] G. Orchard et al., "Converting static image datasets to spiking neuromorphic datasets using saccades," *Frontiers in Neuroscience*, vol. 9, 2015, Article 437.
[16] A. Amir et al., "A low power, fully event-based gesture recognition system," in *IEEE Conference on Computer Vision and Pattern Recognition*, 2017.
[17] N. Srivastava et al., "Dropout: A simple way to prevent neural networks from overfitting," *Journal of Machine Learning Research*, vol. 15, no. 1, pp. 1929–1958, 2014.
[18] S. B. Shrestha and G. Orchard, "SLAYER: Spike layer error reassignment in time," in *Advances in Neural Information Processing Systems*, 2018, pp. 1412–1421.
[19] A. Paszke et al., "Pytorch: An imperative style, high-performance deep learning library," in *Advances in Neural Information Processing Systems 32*, H. Wallach et al., Ed., pp. 8024–8035. Curran Associates, Inc., 2019.
[20] Y. LeCun et al., "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.
[21] W. Gerstner, "Time structure of the activity in neural network models," *Phys. Rev. E*, vol. 51, pp. 738–758, 1995.
[22] R. Serrano-Gotarredona et al., "A neuromorphic cortical-layer microchip for spike-based event processing vision systems," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 53, no. 12, pp. 2548–2566, 2006.
[23] S.-C. Liu et al., *Event-based neuromorphic systems*, John Wiley & Sons, 2014.