Improving the Robustness of Neural Networks to Noisy Multi-Level Non-Volatile Memory-based Synapses

Manon Dampfhoffer, Joel Minguet Lopez, Thomas Mesquida, Alexandre Valentian, Lorena Anghel

Improving the Robustness of Neural Networks to Noisy Multi-Level Non-Volatile Memory-based Synapses

Manon Dampfhoffer
INAC-Spintec, Univ. Grenoble Alpes, CEA, CNRS, Grenoble INP
Grenoble, France
manon.dampfhoffer@cea.fr

Joel Minguet Lopez
CEA-Leti
Univ. Grenoble Alpes
Grenoble, France
joel.minguetlopez@cea.fr

Thomas Mesquida
CEA-List
Univ. Grenoble Alpes
Grenoble, France
thomas.mesquida@cea.fr

Alexandre Valentian
CEA-List
Univ. Grenoble Alpes
Grenoble, France
alexandre.valentian@cea.fr

Lorena Anghel
INAC-Spintec, Univ. Grenoble Alpes, CEA, CNRS, Grenoble INP
Grenoble, France
lorena.anghel@phelma.grenoble-inp.fr

***Abstract*—The implementation of Artificial Neural Networks (ANNs) using analog Non-Volatile Memories (NVMs) for synaptic weights storage promises improved energy-efficiency and higher density compared to fully-digital implementations. However, NVMs are prone to variability, resulting in a degradation of the accuracy of ANNs. In this paper, a general methodology to evaluate and enhance the accuracy of neural networks implemented with non-ideal multi-level NVMs is presented. A hardware fault model distinguishing two types of errors, namely static and dynamic, capturing the variability of NVMs is proposed. Considering various neural networks, it is shown that error-aware training highly increases the robustness to errors compared to a standard, error-agnostic, training. Moreover, Recurrent NNs (RNNs) and Spiking NNs (SNNs) are found to be inherently more robust to dynamic errors than Convolutional NNs (CNNs). In addition, new insights on the adaptability of neural networks to noisy multi-level NVMs are presented, which could further improve their robustness in this context. The methodology aims at providing tools for hardware-software co-design, paving the way for a broader use of multi-level NVM-based synapses.**

***Index Terms*—neuromorphic computing, non-volatile memories, robustness, fault tolerance, spiking neural networks, convolutional neural networks, recurrent neural networks**

I. Introduction

Artificial Neural Networks (ANNs) rely on a large number of computations and data transfers, making their implementation in embedded systems, with stringent power, energy and area constraints, particularly challenging. Neuromorphic computing, such as bio-inspired algorithms and architectures, promise to improve the efficiency of hardware implementations of ANNs [1], [2]. In particular, analog implementations can achieve significant gains compared to fully-digital implementations [3]. In such systems, emerging Non-Volatile Memory (NVMs) devices [4], such as resistive random-access memories (RRAMs), magnetic RAMs (MRAMs), phase-change RAMs (PCRAMs) or ferroelectric RAMs (FeRAMs), can be used to encode synaptic weights of ANNs. The non-volatility of these devices allows them to retain the information even if the power supply is turned off, which is important in Internet of Things applications which wake up on demand. In addition, multi-level cell programming strategies allow more than one bit of information to be stored in a single NVM device, by encoding multiple non-volatile states in the memory. Therefore, multi-level NVMs allow energy-efficient and dense hardware implementations of ANNs [5], [6].

Nevertheless, emerging NVMs are prone to variability, inducing the occurence of errors, which can significantly degrade the accuracy of ANNs. This trend is especially exacerbated with dense multi-level approaches due to the reduced programming window for each level [7], [8]. Therefore, enhancing the robustness of neural networks to noisy multi-level weights is essential to achieve accurate and efficient hardware implementations of ANNs. In addition, variability in NVMs comes from different sources and results in different types of errors, which can have a different impact on the accuracy of ANNs [9], [10].

The robustness of ANNs to NVM non-idealities has been shown to depend on the topology of ANNs. For instance, wider and narrower neural networks are more robust than deep networks [11]. Besides, Spiking Neural Networks (SNNs) are thought to be particularly robust to noise in synaptic weights, due to the computations using accumulation over time [12]. Moreover, SNNs appear to be energy-efficient alternatives to ANNs, due to their brain-like computations and communications using sparse 1-bit spiking activations [13]. Besides, the robustness of ANNs and SNNs to noisy synaptic weights have been compared [12], [14]. However, authors in [12] do not consider a realistic hardware model, as weights are simulated with 32-bit floating point precision and only one type of error

This work has been partially supported by MIAI @ Grenoble Alpes, (ANR-19-P3IA-0003).

is considered. A more realistic hardware model of a RRAM crossbar implementation for evaluating the robustness of SNNs and ANNs is presented in an other work [14]. Nevertheless, none of these works [11], [12], [14] consider the benefits of injecting noise during training, which has proven very effective to enhance the fault tolerance of neural networks [15]. Strategies based on injecting noise during training have been demonstrated with NVM implementations [5], [6], [16], [17]. However, these works focus on a specific hardware implementation and do not distinguish the effect of the different types of faults on the neural network performance. In addition, to the best of our knowledge, there has been no attempt to evaluate and compare the robustness of convolutional neural networks (CNNs) and recurrent neural networks (RNNs) in the context of an implementation with non-ideal NVMs.

In this paper, we propose a methodology to evaluate and improve the robustness of neural networks to noisy multi-level NVM-based synapses. This general methodology is applicable to various types of neural networks and NVM technologies. Moreover, we show that considering the characteristics of NVMs during the training of ANNs is essential to optimize the overall system performance. This work makes the following contributions:

1) an abstract hardware fault model distinguishing two types of errors, namely static and dynamic, capturing the variability of NVMs;
2) a comparison of neural network topologies (CNNs and RNNs) and coding strategies (ANNs and SNNs), evaluated on a keyword spotting task;
3) a comparison of the performance of the various neural networks with a standard (error-agnostic) training and with an adapted (error-aware) training;
4) a deep understanding of the impact of the error-aware training on the parameters learned by the neural networks. These findings could be used to further improve the performance of neural networks by considering the specificity of multi-level NVM implementations.

II. Methods

A. Dataset and Pre-processing

A keyword spotting task from the Google Speech Command Dataset (GSCD) v2 [18] is used for the experiments (Fig. 1.A). This allows us to evaluate RNN and CNN topologies, which are both relevant for this task. GSCD v2 contains 35 different words of at most 1 second, sampled at 16 kHz. The keyword spotting task consists of a 12-class classification problem with 10 keywords (“yes”, “no”, “up”, “down”, “left”, “right”, “on”, “off”, “stop”, “go”), “silence” (background noise) and “unknown” (non-keyword words) classes. The dataset is provided with 84,843 training, 9,981 validation and 4,890 test samples. 40 log-Mel features are extracted from the audio signals, with frequencies ranging from 80Hz to 8kHz, a window size of 30 ms and a hop length of 15 ms. This results in an input data of size of 67x40 for *time x frequency* dimensions that are re-scaled such that each frequency channel has a unit variance across the time dimension. Data augmentation techniques (background noise, time shift, and time and frequency masking) are used to improve the accuracy.

B. Neural Networks

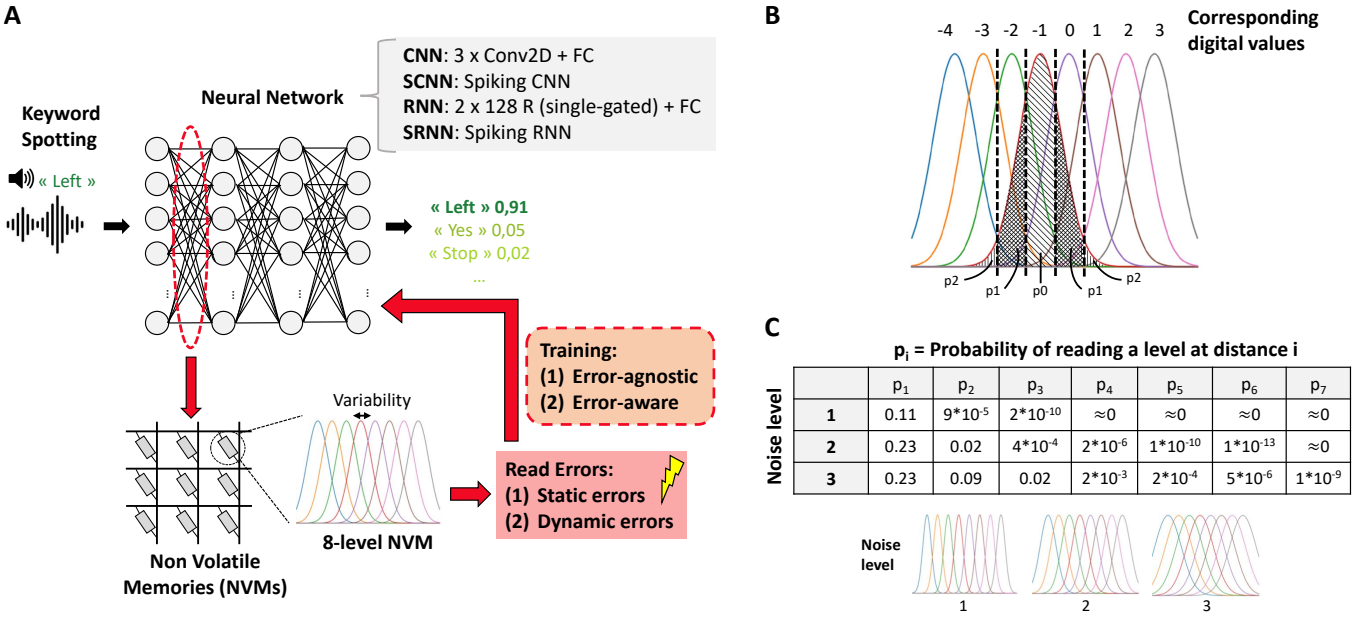
Four different models are used in the experiments: a CNN, a RNN, and their corresponding spiking versions (Fig. 1.A). The CNN and RNN topologies are chosen to have a similar number of parameters and operations per inference. The CNN has three 2D convolutional layers and a final Fully Connected (FC) layer. The convolutional layers have 16, 32, 64 output channels respectively, with a fixed kernel size of (6,4), a stride of (2,2) and a padding of (2,2), with (H,W) being the time and frequency dimensions respectively. The RNN is composed of 2 layers of 128 recurrent units with a single gate similar to the implementation proposed in [19], with a $tanh$ activation function. Indeed, it has been shown that single-gated RNNs can achieve the same accuracy as multi-gated RNN while having a reduced number of parameters and operations [19]. The RNN has a final FC layer where leaky neurons integrate the inputs over time, as in [20]. The CNN uses 2D convolutions, therefore the input data are processed as images of size *time x frequency* dimensions. For the RNN, the 40 frequency channels are fed to the network at each timestep.

Spiking versions of the RNN and CNN topologies are also implemented. Yet, the input data are not converted into spikes but kept in full precision, as for ANNs, which makes it possible to obtain sufficient accuracy and to have the same inputs for the ANNs and SNNs [21]. The Spiking CNN (SCNN) is composed of Leaky Integrate-and-Fire neurons. 10 timesteps are used to simulate the SNN temporal dynamics, thus the input data are fed to the SNN 10 times. The Spiking RNN (SRNN) is composed of SpikGRU units [20], which corresponds to the spiking version of the single-gated RNN. In this case, as the input data are already fed in the form of timesteps, no supplementary timesteps are added as in [20].

C. Multi-Level Non-Volatile Memories Model

Multi-level programming strategy in NVMs allow more than one bit of information to be stored in a single NVM device, by encoding multiple non-volatile states in the memory. For instance, in a resistive NVM (RRAM), the resistance of the device represents the value to be stored. Thus, multiple values can be encoded using multiple stable resistive states in the memory. Multi-level NVMs can be used in different ways to implement ANNs in hardware. The most mature and widely used approach consists in using the NVMs only to store the weights while performing the computation of the matrix vector multiplication digitally. In-memory computing (IMC) is another strategy that leverages the physical properties of the devices to directly perform the matrix vector multiplication in the memory array [5], [6], [22], [23]. In this study, the former approach will be modeled, however, the method can be extended to the latter, as will be discussed.

In this work, one NVM is used to encode each weight of the neural network. The NVMs are modelled with 8 levels [7],



Noise level	p_1	p_2	p_3	p_4	p_5	p_6	p_7
1	0.11	$9*10^{-5}$	$2*10^{-10}$	≈0	≈0	≈0	≈0
2	0.23	0.02	$4*10^{-4}$	$2*10^{-6}$	$1*10^{-10}$	$1*10^{-13}$	≈0
3	0.23	0.09	0.02	$2*10^{-3}$	$2*10^{-4}$	$5*10^{-6}$	$1*10^{-9}$

Fig. 1. **Methods**. **A.** Four types of neural networks using 8-level NVMs for weight implementation are simulated on a keyword spotting task. The effects of two training strategies and two types of errors are considered. **B.** Error model of the 8-level NVM. Digital values associated to each level in these experiments are indicated. p_0 is the probability to correctly read the level, p_i is the probability to read the level at distance i instead of the correct level. **C.** Probabilities of errors depending on the distance between two levels (cf B.). Three noise levels are considered (varying sigma in the gaussian distribution).

[8] associated with a corresponding digital value that will be used as the weight value (see Fig. 1.B). In these experiments, the digital values are signed integers ranging from -4 to +3. A gaussian distribution is used to model the variability in NVMs [24], both with a static and a time varying model. All levels are assumed to have the same variability and to be equally distanced inside the reading window. Therefore, the noise level is determined by the variance of the gaussian distribution. Note that, for some technologies, the variability depends on the programmed resistance, and hence is different between levels [24]. However, by programming the levels such that the overlap in the gaussian distribution of neighboring levels is similar (i.e. by adjusting the distance between levels based on their variability), it is assumed that the NVM would be equivalent to the one modeled in this study.

Three variances are used to obtain three noise levels (Fig. 1.C). In Fig. 1.B and C, p_i corresponds to the probability for a given level to be read as the level at distance i (i.e. it is equal to the area under the curve of a given level that is between the reading thresholds of a level at distance i). p_0 corresponds to the probability that a given level will be read correctly. Note that, although all levels have the same variability, they have different p_0, as they do not have the same neighborhood. For instance, the middle level corresponding to the digital value "-1" has a p_0 equals to 0.50 in the case of noise level 2. Indeed, it has a probability of 0.23 (p_1) to be read "0" and a probability of 0.23 to be read "-2" (neighbors at distance 1). Likewise, it has a probability of 0.02 (p_2) to be read "-3" and a probability of 0.02 to be read "1" (neighbors at distance 2), etc.

An abstract error model covering the variability of NVMs is defined with two categories of errors, namely static and dynamic (modeled with the same gaussian distribution). Static (respectively dynamic) errors are defined as fixed (respectively changing) during the inference time. When testing with static errors, the errors are sampled once for each input data and used for the whole inference. When testing with dynamic errors, an identical and independent sampling of the error distribution is performed each time the weight is read. In practice, these errors result from different physical effects. Depending on the technology, static errors can correspond to programming failures, or temporal fluctuations of the device with a timescale higher than the inference time, such as conductance relaxation for RRAMs [25], or drift for PCRAMs [26]. Dynamic errors can correspond to the inherent noisy behavior of analog devices [10], [17], [27]. In addition, read operations on FeRAMs are destructive [4] and hence errors are always dynamic as the device is re-programmed after each read. Static and dynamic errors have to be distinguished as they do not have the same impact on neural networks, for example in the case where weights are read several times during the inference. In this experiment, it is assumed that weights are read at each timestep for the RNN and at each incoming spike for the SNNs (assuming an event-based hardware implementation). On the contrary, for CNNs, it is assumed that the architecture only reads the weights once per inference due to data re-use techniques [28]. Note that static and dynamic errors can exist simultaneously. However, in this study, they are considered separately to understand their respective impact.

D. Training Procedure

1) General Procedure: The neural networks are implemented based on the NVM model described above. Therefore, the weights are uniformly quantized with signed integers from -4 to +3. To allow the network to adjust the range of input values, a scaling, which multiplies the activations, is defined for each layer as a learnable parameter. Moreover, a full-precision version of the weights (also called "hidden weights") is kept during the training to allow the network to learn despite the highly quantized weights, using a strategy similar to Binarized Neural Networks [29].

All neural networks are trained with backpropagation for 100 epochs and a batch size of 128, with Adam optimizer and a cosine annealing scheduler. The initial learning rate is set at 0.001, except for the weights, for which it is set at 0.01 (as they have a higher magnitude than the other parameters). Neural networks are defined with biases that are not quantized. Biases are initialized from a uniform distribution $U(-1/\sqrt{k}, 1/\sqrt{k})$, k being the input size of the layer. The scaling parameter per layer is initialized to $1/\sqrt{k}$. The "hidden weights" are initialized from a uniform distribution $U(-1, 1)$. For SNNs, the time constants are defined as learnable parameters (per neuron in SRNN and per channel in SCNN) and initialized at 0.8. The voltage threshold for the spiking activation function is set to 1. As the spiking activation function is not differentiable, a surrogate gradient is used for backpropagation as in [20]. In all the experiments, for each configuration, the models are trained five times and the mean accuracy with a 95% confidence interval is reported. Note that, as randomness is involved when testing with errors, models are tested ten times and the mean accuracy is used.

2) Error-aware Training: In the classic error-agnostic training condition, the models are trained from scratch with no knowledge of the errors (the forward pass is done without errors on the weights). In the error-aware training condition, errors on the weights are also applied during training, both in the forward pass and the backward pass. Yet, the weight update is performed on the error-free "hidden weights", which are then quantized to obtain the updated quantized weights, as in [17], [30]. With errors, training from scratch is slower to converge, especially with a high error rate. Therefore, the models in the error-aware condition are initialized with the weights of the models trained in the error-agnostic condition. The models are trained with static errors, so that only one set of weights is used per inference. Note that static errors are sampled for each input data so that the neural network does not learn which particular synapses are faulty, but rather that all synapses are potentially faulty. Hence, this training procedure targets a general robustness to errors rather than a robustness to a specific configuration of errors [5], [31]. Note that the accuracy can be further increased by re-training the neural network involving the hardware in the loop. For instance, authors in [6], [32] propose to fine-tune the neural network using the measured outputs of the fabricated circuit to account for its specific errors. However, this strategy is costly as it must be done after the chip fabrication and repeated for each hardware unit. Conversely, the only knowledge required in the proposed methodology is the expected overall noise level of the devices. Indeed, the noise level used for training must be similar to the one used for testing.

III. Results

A. Robustness to Errors

The figures of merit of the different neural networks are provided in Fig. 2. The number of operations per inference and the number of parameters are given in Fig. 2.B. All models have a similar number of parameters. On one hand, the ANN versions (CNN and RNN) have a similar number of operations per inference. On the other hand, the spiking versions have a lower number of operations due to the spike sparsity. Indeed, the SCNN (respectively SRNN) shows 2x (respectively 4x) reduction in operations compared to the CNN (respectively RNN).

The accuracy of neural networks on the different training conditions (error-agnostic and error-aware) is shown in Fig. 2.A. The two types of errors (static and dynamic) and the different noise levels (described in Section II.C, noise level 0 corresponding to the error-free case) are considered. In the error-agnostic training, the accuracy is largely degraded by errors, with up to 48% accuracy loss in the case of the highest noise level. For the CNN, the accuracy degradation due to static and dynamic errors is the same, as they are considered identical in this simulation, as explained in Section II.C. For SNNs and the RNN, the accuracy degradation is less significant in the case of dynamic errors than static errors. In addition, CNN topologies (SNN and ANN) seem inherently more robust to static errors than RNN topologies (SNN and ANN). Indeed, the CNN and the SCNN have only 30% and 28% accuracy loss (respectively) in the case of highest noise level, compared to 43% and 48% for the RNN and the SRNN (respectively).

Error-aware training is very efficient at increasing the robustness of neural networks to noise, even in the worst noise level scenario. However, the higher the noise level, the higher the accuracy gap with the error-free case. Indeed, the accuracy loss with respect to the error-free case is less than 1% for the lowest noise level, but up to 3% for the highest noise level. In addition, with error-aware training, there are no longer significant differences between CNN and RNN topologies in terms of robustness to static errors. Nevertheless, SNNs and the RNN are still significantly more robust than the CNN to dynamic errors for the highest noise level.

B. Impact of Error-aware Training

The impact of the training condition (error-aware vs. error-agnostic) on the weights and scaling parameters is shown in Fig. 3. Fig. 3.A illustrates the role of the weights and scaling parameters in the computation of the output activation of a neuron. Input activations are multiplied by the weights and a scaling factor, which are learned by the neural networks, before the activation function. Fig. 3.B shows the scaling learned

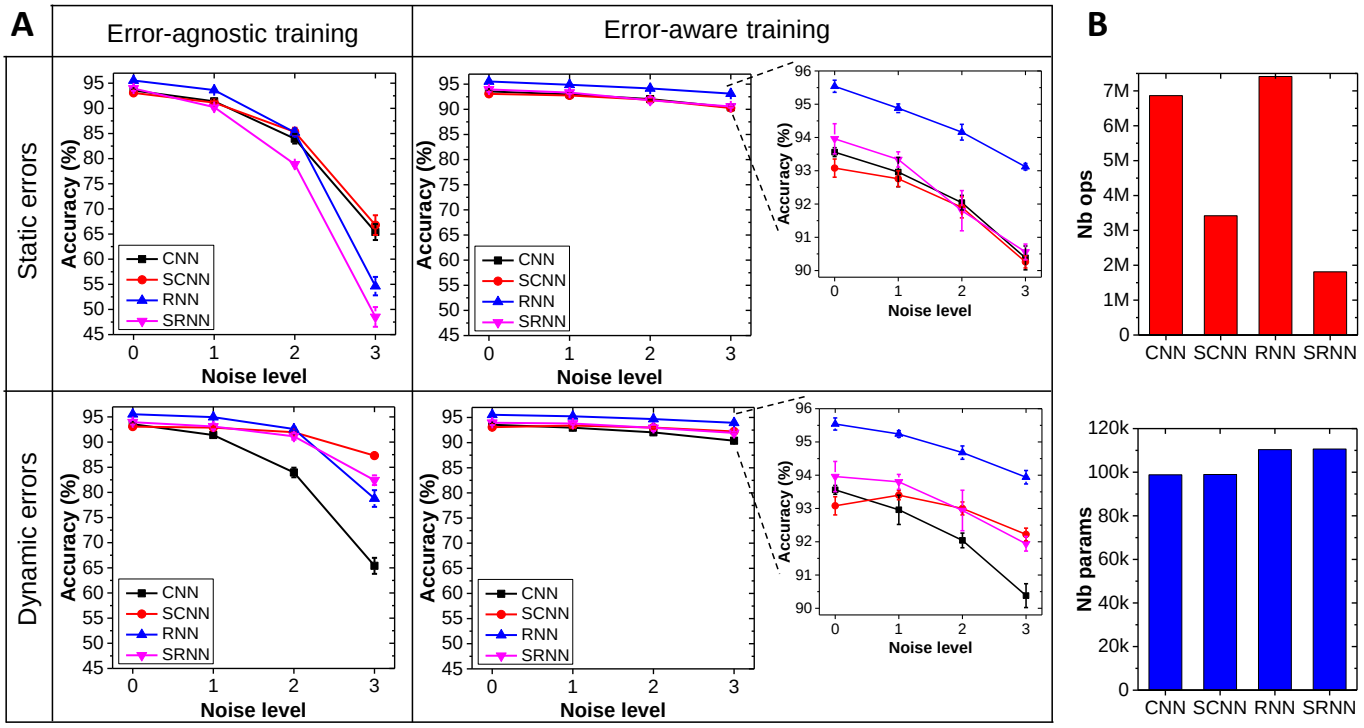


Fig. 2. **Results. A.** Test accuracy of the different neural networks (shown with a 95% confidence interval) on the keyword spotting task. Static (top) and dynamic (bottom) errors are considered with different noise levels (cf Fig. 2.C), noise level 0 corresponding to the error-free case. Error-agnostic training (left) and error-aware training (right) are compared. In error-aware training, models are trained to the same noise level as the one used for testing. **B.** Number of operations per inference (top) and number of parameters (bottom) of the models.

for the different layers of the neural networks depending on the noise level during training. For almost all layers, the scaling learned in the error-aware training is lower than in the error-agnostic training. Moreover, the higher the noise level during training, the lower the scaling learned. In addition, in the error-agnostic condition, the neural networks learn a gaussian-like weight distribution centered on 0. On the contrary, in the error-aware condition, the weights are distributed more equally among levels, except for the levels on the sides (corresponding to the highest weight magnitude), which are the most used. Moreover, the average magnitude of the weights learned with error-aware training is higher than with error-agnostic training. In addition, as for the scaling, the higher the noise level during training, the more this trend is exacerbated.

IV. Discussion

A. Robustness to Errors

RNN topologies (ANN and SNN) seem inherently (in error-agnostic training) less robust to static errors than CNN topologies (ANN and SNN). This could be explained by the high temporal depth of RNNs, which could lead to errors accumulation and hence accuracy degradation [11]. Indeed, although spatially shallow (they have a small number of layers), RNNs are very deep in time. In fact, the output from a recurrent layer is used as input at the next timestep. This means that, when the RNN is unrolled in time, it has an equivalent depth of 67 in the temporal dimension (in these experiments 67 timesteps are used). Nevertheless, the results demonstrate that RNN topologies can recover as much accuracy as CNN topologies with the error-aware training. Note that the SCNN also has a temporal depth due to the use of 10 timesteps to simulate the temporal dynamics of SNN. However the number of timesteps is small compared to the case of the RNN and the SRNN. This is in line with the results in [14] showing that a lower number of timesteps mitigates the accumulation of errors over time in SNNs.

On the contrary, SNNs and RNNs appear to have a better robustness to dynamic errors than CNNs, in both error-agnostic and error-aware training conditions. This is consistent with the results in [12], where SNNs are found to be more robust than ANNs to dynamic errors in the case of CNN and FC topologies. Indeed, as explained in [12], synapses in CNNs or FCs are used only once per inference. On the contrary, synapses in SNNs are used as many time as the number of spikes transmitted across the synapse during inference. In the case of dynamic errors, each weight read for a given synapse is sampled from the same gaussian distribution. Therefore, the more spikes transmitted across the synapse, the more the gaussian noise is minimized. Note that even if the average number of spikes per synapse is lower than 1, some neurons are activated more frequently, resulting in some synapses transmitting a large number of spikes. Nevertheless, this property does not apply to the case of static errors, where the same faulty weights are used for the whole inference. In addition,

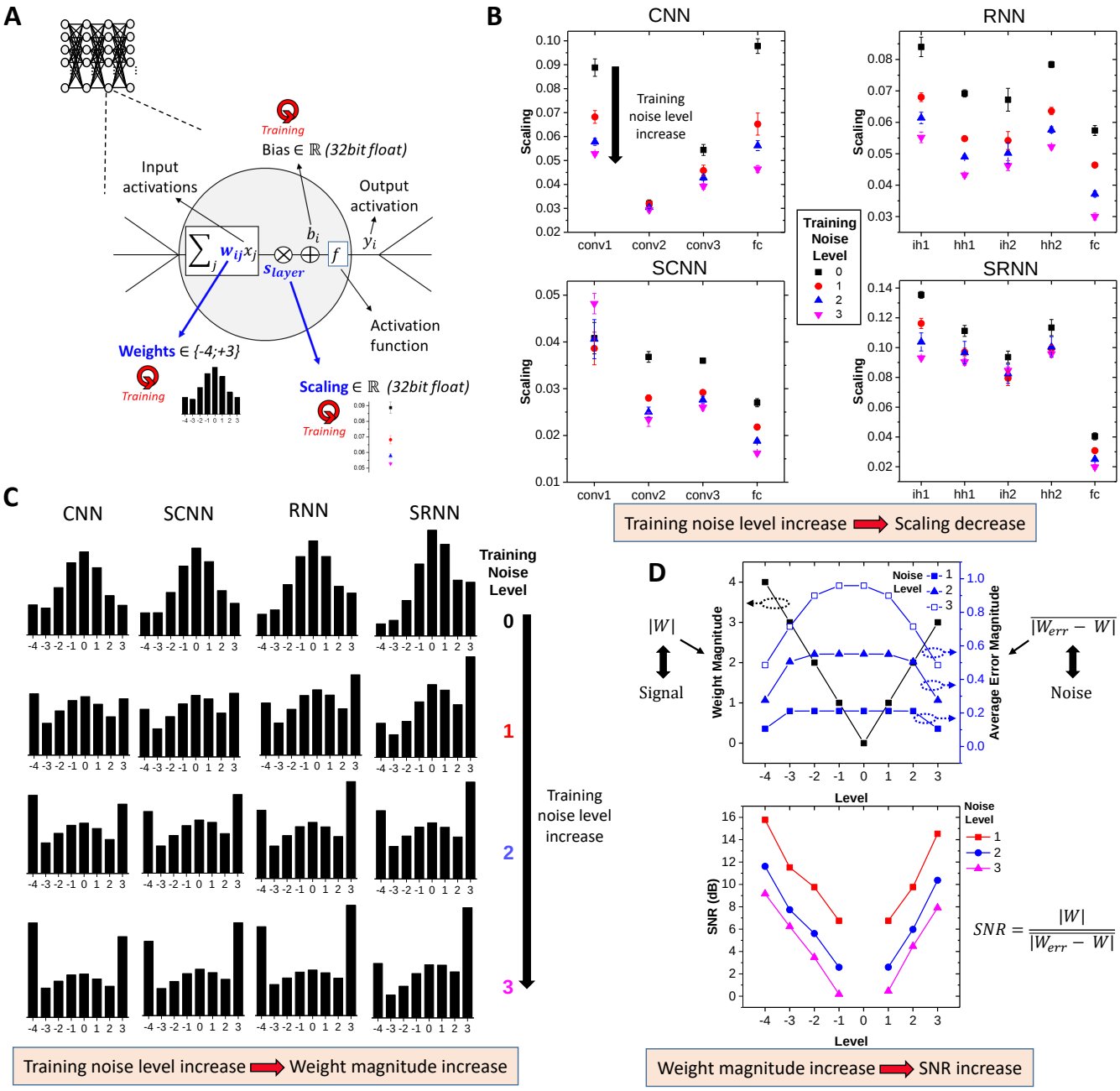


Fig. 3. **Analysis of error training. A.** Illustration of the operations for computing the output activation of a neuron. Input activations are multiplied with the quantized weights (8 levels) and by a scaling (full precision, 1 per layer) to adjust the range of the pre-activation. A bias (full precision, 1 per neuron for RNNs and 1 per channel for CNNs) is added before the activation function. Weights, scaling and bias are trained. **B.** Scaling of each layer for the different models after training, when the models are trained with different noise levels (0 corresponding to error-agnostic training). **C.** Weight distribution for the different models after training, when the models are trained with different noise levels (0 corresponding to error-agnostic training). **D.** (Top) Weight Magnitude and Average Error Magnitude for each of the 8 NVM levels, for different noise levels. (Bottom) Signal-to-Noise Ratio (SNR) for each of the 8 NVM levels, for different noise levels. As the level associated with the digital value "0" has a signal amplitude of "0", its SNR in dB is $-\infty$ and is not represented.

the results show that this property is also applicable to RNNs. Indeed, RNNs have the same behavior as SNNs in the sense that a synapse is re-used at each timestep of the inference if the input activation is non-zero. For comparison, the CNN was tested with the same implementation as SCNN (i.e. with the weights being read at each pixel). However, even in this case, the results obtained with dynamic errors are similar to the results obtained with static errors. This shows the importance of the accumulation over time in the same synapse for the dynamic errors to compensate. Therefore, SNNs seem to be a good choice for an energy-efficient hardware implementation as they benefit from an increased robustness to dynamic errors, as well as a significant reduction in operations per inference.

B. Impact of Error-aware Training

In these experiments, the weights and scaling parameters are highly impacted by the noise level during training. Indeed, the higher the noise level during training, the higher the magnitude of the weights and the lower the scaling learned. Two reasons can explain the change in weight distribution (and hence magnitude increase). First, in the error-aware training condition, the weights are distributed more uniformly compared to the error-agnostic training condition, with less weights at level "0". Therefore, the neural network may increase its robustness to errors by making more weights specialized, as proposed in [6]. Second, the NVM levels have a different Signal-to-Noise Ratio (SNR), as shown in Fig 3.D. Indeed, the outer levels have a high SNR, meaning that the amplitude of the noise is relatively small compared to the amplitude of the signal they carry, compared to the levels in the middle. On one hand, outer levels benefit from a higher weight magnitude (signal) and a lower error magnitude (noise). On the other hand, middle levels have a lower magnitude and a higher error magnitude. Indeed, as shown in Fig 1.B, levels in the middle have more neighbors compared to outer levels, and hence more overlap with other levels. Moreover, the error magnitude between two levels depends only on the distance between those level (it is equal to the difference between the two values encoded by these levels). For instance, mistaking a "-4" for a "-3" has likely less impact than mistaking a "0" for a "+1", while these two errors have the same magnitude. Therefore, by increasing the magnitude of the weights, the SNR of synapses is increased, and hence the accuracy of the neural network is improved. Finally, the scaling decrease may be a consequence of the weight magnitude increase, allowing to keep the same magnitude of pre-activations.

These findings can allow further improvement of the performance of neural networks in the context of such hardware implementations. In these experiments, both weights and scaling are important for the network to increase its robustness to errors. Nevertheless, this result depends on the quantization method, training procedure and error model. For instance, the quantization method, such as the choice of the digital values associated with the levels, has an impact on both the SNR of the levels and the learned weight distribution, and hence may be carefully considered. Moreover, having a NVM level associated with the value "0" may not be the optimal strategy under high noise level. Indeed, the value "0" is inherently very sensitive to noise due to its null magnitude. Finally, this shows the importance of including all possible hyperparameters in the optimization process, as some hyperparameters may have an unexpected impact on the robustness of neural networks to errors.

C. Limitations and Perspectives

This study considers the role of topology (CNN, RNN) and coding (SNN, ANN) on the robustness of neural networks to errors, while other factors may be important. For instance, deeper networks are inherently less robust to errors as errors accumulate through layers [11]. In these experiments, the effect of depth in time was considered (with RNNs). However, the case of deep networks in the spatial dimension (i.e. having more than a few layers) should be further investigated. Note that noise injection training strategies have shown high accuracy for CNNs with 20 [6] or 34 [5] layers, which suggests that this strategy is also effective for deeper networks in space. In addition, the role of the activation function could be further studied. For instance, a higher inherent robustness to errors (without specific training) was observed for the RNN when using a $tanh$ activation function rather than a $ReLU$ or a $linear$ activation function, which suggests that bounded activation functions increase robustness to errors (in line with [33]). Besides, the role of the binary activation function of SNNs in the robustness to errors was not investigated. In addition, robustness to errors may decrease with increasing task difficulty.

In this study, the NVM and error model is very general to be the most independent of the hardware implementation (such as choice of bit-cell implementation and NVM technology). Nevertheless, this model can be adapted for each specific case. For instance, the impact of the combination of the two types of errors could be studied. Moreover, only errors related to the memories have been considered, while, depending on the implementation, other sources of errors can be added to the model [9], [32], [34]. Finally, although specifically focused on the case of analog memories only used for weight storage, the methodology can be extended to the case of IMC. Note that the way of applying errors to the weights would be slightly different. Indeed, in these experiments, the errors are applied in a discretized way as levels are discretized (for instance, a weight can be read at level "0" or "1", but not at an intermediate value). On the contrary, in the case of IMC, the noisy analog values are directly used in the computation. In that case, sources of noise coming from the analog computation should be added to the model [9].

V. Conclusion

A general methodology to evaluate and enhance the performance of neural networks in the context of synaptic weights implemented with multi-level NVMs was presented. Error-aware training is demonstrated to be very effective to improve the robustness of neural networks to high error rates, making them perfectly suitable for multi-level NVM implementations. Moreover, two types of errors capturing the variability of NVMs, namely static and dynamic errors, have been distinguished and have shown a different impact on the accuracy of neural networks. In particular, SNNs and RNNs appear to be inherently more robust to dynamic than static errors, due to the nature of their computation using accumulation over time. In addition, they are found to be more robust to dynamic errors than CNNs. Moreover, error-training was observed to change the weight distribution learned by the neural networks. These findings are of interest to further improve the performance of neural networks in this context. The hardware fault model and the training strategy presented aim at providing tools in a hardware-software co-design perspective. In this context, this

methodology can pave the way for a broader use of multi-level NVM-based synapses, promoting highly efficient hardware implementations of neural networks.

REFERENCES

[1] C. Mead, "Neuromorphic electronic systems," *Proceedings of the IEEE*, vol. 78, no. 10, pp. 1629–1636, 1990.

[2] S. K. Bose, J. Acharya, and A. Basu, "Is my Neural Network Neuromorphic? Taxonomy, Recent Trends and Future Directions in Neuromorphic Engineering," in *2019 53rd Asilomar Conference on Signals, Systems, and Computers*, 2019, pp. 1522–1527.

[3] J.-M. Hung, Y.-H. Huang, S.-P. Huang, F.-C. Chang, T.-H. Wen, C.-I. Su, W.-S. Khwa, C.-C. Lo, R.-S. Liu, C.-C. Hsieh, K.-T. Tang, Y.-D. Chih, T.-Y. J. Chang, and M.-F. Chang, "An 8-Mb DC-Current-Free Binary-to-8b Precision ReRAM Nonvolatile Computing-in-Memory Macro using Time-Space-Readout with 1286.4-21.6TOPS/W for Edge-AI Devices," in *2022 IEEE International Solid- State Circuits Conference (ISSCC)*, vol. 65, 2022, pp. 1–3.

[4] D. Ielmini and S. Ambrogio, "Emerging neuromorphic devices," *Nanotechnology*, vol. 31, no. 9, p. 092001, Dec. 2019.

[5] V. Joshi, M. Le Gallo, S. Haefeli, I. Boybat, S. Nandakumar, C. Piveteau, M. Dazzi, B. Rajendran, A. Sebastian, and E. Eleftheriou, "Accurate deep neural network inference using computational phase-change memory," *Nat Commun*, vol. 11, no. 1, Dec. 2020.

[6] W. Wan, R. Kubendran, C. J. S. Schaefer, S. B. Eryilmaz, W. Zhang, D. Wu, S. R. Deiss, P. Raina, H. Qian, B. Gao, S. Joshi, H. Wu, H. P. Wong, and G. Cauwenberghs, "A compute-in-memory chip based on resistive random-access memory," *Nat.*, vol. 608, no. 7923, pp. 504–512, 2022.

[7] T. Nirschl, J. Philipp, T. Happ, G. Burr, B. Rajendran, M.-H. Lee, A. Schrott, M. Yang, M. Breitwisch, C.-F. Chen, E. Joseph, M. Lamorey, R. Cheek, S.-H. Chen, S. Zaidi, S. Raoux, Y. Chen, Y. Zhu, R. Bergmann, H.-L. Lung, and C. Lam, "Write Strategies for 2 and 4-bit Multi-Level Phase-Change Memory," in *2007 IEEE International Electron Devices Meeting*, 2007, pp. 461–464.

[8] S. Balatti, S. Larentis, D. C. Gilmer, and D. Ielmini, "Multiple Memory States in Resistive Switching Devices Through Controlled Size and Orientation of the Conductive Filament," *Advanced Materials*, vol. 25, no. 10, pp. 1474–1478, 2013.

[9] K. Higuchi, C. Matsui, and K. Takeuchi, "Investigation of Memory Non-Ideality Impacts on Non-Volatile Memory Based Computation-in-Memory AI Inference by Comprehensive Simulation Platform," in *2022 IEEE Silicon Nanoelectronics Workshop (SNW)*, 2022, pp. 1–2.

[10] Z. Yan, X. S. Hu, and Y. Shi, "On the Reliability of Computing-in-Memory Accelerators for Deep Neural Networks," in *System Dependability and Analytics: Approaching System Dependability from Data, System and Analytics Perspectives*, Cham, 2023, pp. 167–190.

[11] T.-J. Yang and V. Sze, "Design Considerations for Efficient Deep Neural Networks on Processing-in-Memory Accelerators," in *2019 IEEE International Electron Devices Meeting (IEDM)*, 2019, pp. 22.1.1–22.1.4.

[12] C. Li, R. Chen, C. Moutafis, and S. Furber, "Robustness to Noisy Synaptic Weights in Spiking Neural Networks," in *2020 International Joint Conference on Neural Networks (IJCNN)*, 2020, pp. 1–8.

[13] M. Dampfhoffer, T. Mesquida, A. Valentian, and L. Anghel, "Are SNNs Really More Energy-Efficient Than ANNs? An In-Depth Hardware-Aware Study," *IEEE Transactions on Emerging Topics in Computational Intelligence*, 2022.

[14] A. Bhattacharjee, Y. Kim, A. Moitra, and P. Panda, "Examining the Robustness of Spiking Neural Networks on Non-Ideal Memristive Crossbars," in *Proceedings of the ACM/IEEE International Symposium on Low Power Electronics and Design*, ser. ISLPED '22, 2022.

[15] A. Murray and P. Edwards, "Enhanced MLP performance and fault tolerance resulting from synaptic weight noise during training," *IEEE Transactions on Neural Networks*, vol. 5, no. 5, pp. 792–802, 1994.

[16] J. Doevenspeck, K. Garello, S. Rao, F. Yasin, S. Couet, G. Jayakumar, A. Mallik, S. Cosemans, P. Debacker, D. Verkest, R. Lauwereins, W. Dehaene, and G. Kar, "Multi-pillar SOT-MRAM for Accurate Analog in-Memory DNN Inference," in *2021 Symposium on VLSI Technology*, 2021, pp. 1–2.

[17] J. Minguet Lopez, Q. Rafhay, M. Dampfhoffer, L. Reganaz, N. Castellani, V. Meli, S. Martin, L. Grenouillet, G. Navarro, T. Magis, C. Carabasse, T. Hirtzlin, E. Vianello, D. Deleruyelle, J.-M. Portal, G. Molas, and F. Andrieu, "1S1R Optimization for High-Frequency Inference on Binarized Spiking Neural Networks," *Advanced Electronic Materials*, p. 2200323, jun 2022.

[18] P. Warden, "Speech commands: A dataset for limited-vocabulary speech recognition," 2018. [Online]. Available: https://arxiv.org/abs/1804.03209

[19] M. Ravanelli, P. Brakel, M. Omologo, and Y. Bengio, "Light Gated Recurrent Units for Speech Recognition," *IEEE Transactions on Emerging Topics in Computational Intelligence*, vol. 2, no. 2, pp. 92–102, 2018.

[20] M. Dampfhoffer, T. Mesquida, A. Valentian, and L. Anghel, "Investigating Current-Based and Gating Approaches for Accurate and Energy-Efficient Spiking Recurrent Neural Networks," in *Artificial Neural Networks and Machine Learning – ICANN 2022. Lecture Notes in Computer Science, vol 13531*, 2022.

[21] L. Deng, Y. Wu, X. Hu, L. Liang, Y. Ding, G. Li, G. Zhao, P. Li, and Y. Xie, "Rethinking the performance comparison between SNNs and ANNs," *Neural Networks*, vol. 121, pp. 294–307, Jan. 2020.

[22] H. Amrouch, N. Du, A. Gebregiorgis, S. Hamdioui, and I. Polian, "Towards Reliable In-Memory Computing:From Emerging Devices to Post-von-Neumann Architectures," in *2021 IFIP/IEEE 29th International Conference on Very Large Scale Integration (VLSI-SoC)*, 2021, pp. 1–6.

[23] S. Jung, H. Lee, S. Myung, H. Kim, S. Yoon, S.-W. Kwon, Y. Ju, M. Kim, W. Yi, S. Han, B. Kwon, B. Seo, K. Lee, G.-H. Koh, K. Lee, Y. Song, C. Choi, D. Ham, and S. Kim, "A crossbar array of magnetoresistive memory devices for in-memory computing," *Nature*, vol. 601, pp. 211–216, 01 2022.

[24] A. Grossi, E. Nowak, C. Zambelli, C. Pellissier, S. Bernasconi, G. Cibrario, K. El Hajjam, R. Crochemore, J. Nodin, P. Olivo, and L. Perniola, "Fundamental variability limits of filament-based RRAM," in *2016 IEEE International Electron Devices Meeting (IEDM)*, 2016, pp. 4.7.1–4.7.4.

[25] M. Zhao, H. Wu, B. Gao, Q. Zhang, W. Wu, S. Wang, Y. Xi, D. Wu, N. Deng, S. Yu, H.-Y. Chen, and H. Qian, "Investigation of statistical retention of filamentary analog RRAM for neuromophic computing," in *2017 IEEE International Electron Devices Meeting (IEDM)*, 2017, pp. 39.4.1–39.4.4.

[26] I. V. Karpov, M. Mitra, D. Kau, G. Spadini, Y. A. Kryukov, and V. G. Karpov, "Fundamental drift of parameters in chalcogenide phase change memory," *Journal of Applied Physics*, vol. 102, no. 12, p. 124503, 2007.

[27] L. Reganaz, D. Deleruyelle, Q. Rafhay, J. Minguet Lopez, N. Castellani, J. F. Nodin, A. Bricalli, G. Piccolboni, G. Molas, and F. Andrieu, "Investigation of resistance fluctuations in ReRAM: physical origin, temporal dependence and impact on memory reliability," in *accepted to 2023 IEEE International Reliability Physics Symposium (IRPS)*.

[28] V. Sze, Y.-H. Chen, T.-J. Yang, and J. S. Emer, "Efficient Processing of Deep Neural Networks," *Synthesis Lectures on Computer Architecture*, vol. 15, no. 2, pp. 1–341, Jun. 2020.

[29] M. Courbariaux, I. Hubara, D. Soudry, R. El-Yaniv, and Y. Bengio, "Binarized Neural Networks: Training Deep Neural Networks with Weights and Activations Constrained to +1 or -1," 2016.

[30] T. Hirtzlin, M. Bocquet, J.-O. Klein, E. Nowak, E. Vianello, J.-M. Portal, and D. Querlioz, "Outstanding Bit Error Tolerance of Resistive RAM-Based Binarized Neural Networks," in *2019 IEEE International Conference on Artificial Intelligence Circuits and Systems (AICAS)*, 2019, pp. 288–292.

[31] S. Burel, A. Evans, and L. Anghel, "MOZART+: Masking Outputs With Zeros for Improved Architectural Robustness and Testing of DNN Accelerators," *IEEE Transactions on Device and Materials Reliability*, vol. 22, no. 2, pp. 120–128, 2022.

[32] S. Moon, K. Shin, and D. Jeon, "Enhancing Reliability of Analog Neural Network Processors," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 27, no. 6, pp. 1455–1459, 2019.

[33] E. Malekzadeh, N. Rohbani, Z. Lu, and M. Ebrahimi, "The Impact of Faults on DNNs: A Case Study," in *2021 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT)*, 2021, pp. 1–6.

[34] E.-I. Vatajelu, G. Di Natale, and L. Anghel, "Special Session: Reliability of Hardware-Implemented Spiking Neural Networks (SNN)," in *2019 IEEE 37th VLSI Test Symposium (VTS)*, 2019, pp. 1–8.