Fast and Accurate Output Error Estimation for Memristor-Based Deep Neural Networks

Jonathan Kern, Sébastien Henwood, Gonçalo Mordido, Elsa Dupraz, *Member, IEEE,* Abdeldjalil Aïssa-El-Bey, *Senior Member, IEEE,* Yvon Savaria, *Fellow, IEEE,* and François Leduc-Primeau, *Member, IEEE*

***Abstract*—Memristors allow computing in memory, which may be leveraged by deep neural network (DNN) accelerators to reduce energy footprint. However, such gains in energy efficiency come at the cost of noise on the computation results due to the analog nature of memristors. In this work, we introduce a theoretical framework to estimate the mean squared error (MSE) of a memristor-based DNN. We propose an efficient software implementation of this framework which is shown to be orders of magnitude faster than using Monte-Carlo simulations. Additionally, we study two different techniques for mapping convolutional layers to memristors and compare their relative impact on the mean squared error and its computation time. The accuracy of the proposed analysis is first evaluated on a simple regression problem, and then on a more complex classification task with a network capable of achieving high accuracy on the CIFAR-10 dataset, which shows that our method is efficient over practical up-to-date DNNs. The proposed framework is then used to perform a meta-heuristic optimization of the memristor maximal conductance value so as to minimize the energy usage.**[1]

***Index Terms*—Memristors, neural networks, energy efficiency, in-memory computing**

I. Introduction

MEMORY accesses are one of the most energy-consuming parts of a computing system [1]. This is a critical issue for systems that have high memory requirements [2], frequent data access [3], or are prone to run on embedded devices [4]. For example, state-of-the-art deep neural networks (DNNs) on the ImageNet [5] classification task require millions of parameters and billions of FLOPs [6]–[8] when implemented with floating-point arithmetic. Therefore processing-in-memory (PIM) recently emerged as a very promising alternative compared to the conventional Von-Neumann architecture [9], [10]. Specifically, several PIM designs have been shown to achieve significantly lower energy footprints for DNN implementations [11]. Among them, memristors [12] are a form of low-power, non-volatile memory that allows computing matrix-vector multiplications (MVM) directly in memory, by programming the memristor conductance values to some specific levels [13], [14]. Hence, memristor-based DNN implementations are able to achieve a significant reduction in energy consumption compared to conventional DNN systems.

However, the programmed conductance levels in memristors may be affected by noise originating from several possible sources [15], [16]. For instance, in memristors, as in any resistors, there is a thermal noise that can affect computations. Additionally, device-to-device variability of the resistance is another source of unreliability that is likely to affect the result of the computation. Despite such non-idealities, DNNs have been shown to inherently tolerate some level of noise up to a certain extent, while still maintaining their performance [17]. Moreover, methods to further improve DNN noise tolerance during the training process have also been recently proposed [18]–[21]. Existing methods primarily rely on injecting noise during training, which has also been applied to the weights [20] and activations [22] of memristor-based DNNs.

Even though memristor-based DNN implementations have recently gained in popularity, their analyses, design, and optimization have been primarily empirical in nature [23]–[26]. In this work, we propose an alternative to the vast empirical work on memristors by using a theoretical framework instead. More specifically, our analysis can be used to estimate the performance of a trained DNN deployed on noisy PIM hardware. As highlighted previously, the development of memristor-enabled DNNs could be a critical asset for curbing the increasing power usage of state-of-the-art neural networks. A theoretical framework capable of estimating the influence of the unreliability of memristors on the computations of DNNs could be useful for helping this development by providing tools for the joint design of reliable DNN architectures and memristor implementations. As shown in this paper, such joint considerations between hardware and algorithm can be used to further increase the energy gains achievable by using a memristor architecture while preserving the accuracy of DNNs. However, to be of interest, this framework needs to have a high degree of accuracy and efficiency, in line with the existing methods used for estimating uncertainty on standard DNNs.

As related work, several advances have been made to study uncertainty in the outputs of neural networks without relying on Monte-Carlo simulations, which can be highly resource-intensive. In these approaches, weights are modeled as random variables, and the goal is to estimate the posterior distribution

J. Kern is with the Department of Electrical Engineering, Polytechnique Montreal, QC, Canada , and also with IMT Atlantique, Lab-STICC, UMR CNRS 6285, 29238 Brest, France (email: jonathan.kern@imt-atlantique.fr). S. Henwood, G. Mordido, Y. Savaria and F. Leduc-Primeau are with the Department of Electrical Engineering, Polytechnique Montreal, QC, Canada (emails: {sebastien.henwood,yvon.savaria,francois.leduc-primeau}@polymtl.ca). E. Dupraz and A. Aïssa-El-Bey are with IMT Atlantique, Lab-STICC, UMR CNRS 6285, 29238 Brest, France (emails: {elsa.dupraz,abdeldjalil.aissaelbey}@imt-atlantique.fr). G. Mordido is also with Mila - Quebec AI Institute, Montreal, QC, Canada (email:goncalomordido@gmail.com).

This work was supported by an IVADO grant (PRF-2019-4784991664) and by the Samuel-de-Champlain program.

[1]The code for the framework presented in this work is available at https://github.com/sebastienwood/MemSE

of network outputs over the network parameters. Most recent methods used variational inferences. A first example is [27] where the *Bayes by Backprop* approach is introduced. This method considers a Gaussian distribution over the weights of the network. Another example of the use of variational inference can be found in [28], which proposes to use Bernoulli variational distributions to interpret dropout in a network. These works aimed to model uncertainty in each layer of DNNs but did not propagate the variational distribution through the layers and preferred to sample randomly from the distribution and pass it to the respective next layers. This approach was motivated by the difficulty of propagating moments through the non-linear operations of DNNs. However [29] presented an extended variational inference framework, capable of propagating the mean and covariance of the DNN at the output of each layer. This paper, and later [30], [31], clearly identify the non-linearities of each layer as a challenge for moment propagation and proposed two methods to address this issue: one using a first-order Taylor series approximation and another using an unscented transformation.

While all the previously mentioned works focused on standard implementations suffering from noise or adversarial attacks, moment propagation was also applied to memristor-based implementations. This was first reported in [32] which presented a theoretical analysis for studying the influence of conductance variations on a DNN at inference time. However, [32] considers a memristor crossbars model for MVM computations based on passive summing circuits, which seems less widely utilized in current experimental implementations compared to the memristor model used in our work. Then our preliminary work [33], which uses the same model as in this work, also proposed a theoretical framework for estimating the mean squared error (MSE) of a memristor-based DNN. Here, unlike [31]–[33], we investigate the case where convolutional layers are directly implemented on memristors, rather than converted to fully-connected layers. This has an important impact on the theoretical analysis based on moment propagation. Moreover, we propose a novel and more accurate method for moment estimation after the activation functions.

In this work, we propose an improved method, in terms of adaptability and precision, for theoretically estimating the influence of noise on the computation of a memristor-based DNN. Particularly, we develop a framework for estimating the MSE between an unreliable neural network implemented using memristors and its reliable counterpart. This framework allows the prediction of the first and second moments of the outputs of a variety of DNN layers. We also present equations for evaluating the power consumption of the memristors used by the network. Finally, we propose an optimization method that allows minimizing the power consumption of a memristor-based DNN to meet a desired MSE. Furthermore, we introduce a runtime-efficient implementation of our theoretical framework and empirically demonstrate a speedup of two orders of magnitude compared to Monte-Carlo simulations. We showcase the correctness of the proposed theoretical analysis when estimating the MSE of a small DNN model applied to a regression task and a larger convolutional neural network (CNN) capable of achieving an accuracy superior to 90% on the CIFAR-10 classification dataset [34]. Additionally, we show that using the theoretical analysis in the previously mentioned optimization problem, it is possible to reach the baseline accuracy of a network with 6% less power consumption than with its non-optimized counterpart. In the end, our results show that our theoretical framework can accurately and efficiently predict the performance of a memristor-based DNN as a function of device characteristics.

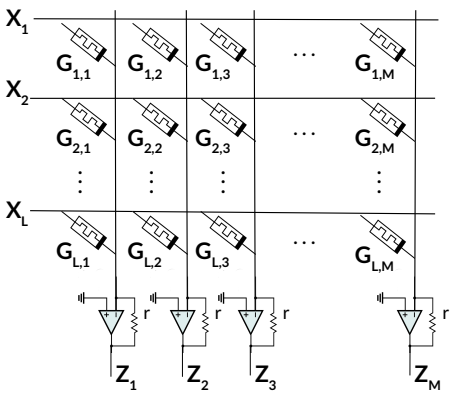


Fig. 1: Memristor crossbar architecture for MVM.

The main contributions of this paper can be summarized as follows:

- We provide theoretical equations capable of efficiently estimating the MSE and power usage of memristor-enabled DNNs with respect to their reliable digital counterpart through a moment propagation method.
- We propose a framework to compute these theoretical metrics which can be several orders of magnitude faster than using Monte-Carlo simulations to achieve the same estimation error.
- We make use of this framework to efficiently optimize the hardware parameters of memristor crossbars to minimize their energy consumption under performance constraints.

The remainder of the paper is organized as follows: Section II introduces the memristors and DNN models, Section III details our theoretical analysis of the memristor-based DNN performance and provides an overview of the computational challenges derived from efficiently implementing our analytical framework, Sections V and VI describe the proposed optimization method and corresponding results, respectively, and, finally, Section VII provides a summary of our main contributions and future directions.

II. System models

Before presenting our theoretical analysis, we first need to describe the memristor crossbar model we consider and its associated noise model, as well as how this model is used in the context of MVM on memristors. Then, we introduce the characteristics and notation of the DNNs that will be considered in this paper.

A. Memristor model

Figure 1 illustrates the architecture of the considered memristor crossbar. In accordance with Ohm's Law and Kirchoff's Law,

the current in each branch is given by the conductance at each node multiplied by the input voltage of the row. These products are then summed along the column. Finally, a transimpedance amplifier (TIA) converts the current into a voltage at the end of each column [35]. The output z_j of the j-th column is thus given by

$$z_j = r \sum_{i=1}^{L} g_{i,j} x_i \,, \tag{1}$$

where x_i is the voltage at the input of row i, $g_{i,j}$ is the conductance of the memristor at row i and column j, and r is the feedback resistance of the TIA.

Unfortunately, several practical issues may cause the actual computation to differ from the ideal case presented in equation (1). Specifically, conductance values may be affected by fabrication variations and noise during programming [14], [16], [20], [36]. We consider that the memristors conductance ranges from $\mathrm{g_{min}}$ to $\mathrm{g_{max}}$, which respectively represent the minimum and maximum physical conductance values programmable on a memristor. To take variability into account, we represent the programmed conductance values as random variables $G_{i,j}$, which we model as

$$G_{i,j} = g_{i,j} + \epsilon^v_{i,j} \,, \tag{2}$$

where $g_{i,j}$ is the desired value and $\epsilon^v_{i,j}$ is the noise due to variability in conductance programming. Commonly, this noise is assumed to follow a normal distribution with 0 mean [20]. We use σ_v^2 to denote the variance of ϵ^v. In practice, σ_v^2 may vary with the conductance value $g_{i,j}$ [16], but for simplicity, we assume here that it is constant. Nevertheless, the analysis proposed in this paper can be easily extended to the case where σ_v^2 depends on the conductance value.

B. Mapping dot-product computation into a memristor crossbar

Since memristors can only store positive values, we actually use two memristor crossbars for storing one matrix: the $g^{(+)}_{i,j}$ are the positive values of the matrix and the $g^{(-)}_{i,j}$ are the opposite of the negative values. As in [25], the MVM can then be realized by

$$Z_j = \sum_{i=1}^{L} r G^{(+)}_{i,j} X_i - \sum_{i=1}^{L} r G^{(-)}_{i,j} X_i \,, \tag{3}$$

where Z_j, $G^{(+)}_{i,j}$, $G^{(-)}_{i,j}$, and X_i are random variables. Moreover, X_i and Z_j can be seen as noisy versions of the input x_i and output z_j presented in (1), respectively.

Now, an MVM required by a neural network can be formulated as $y = wx$, where w is a weight matrix and x is the input vector. To map this operation into a memristor crossbar, we need to convert the original weight matrix w to two conductance arrays $g^{(+)}$ and $g^{(-)}$. Because of the range $[\mathrm{g_{min}}, \mathrm{g_{max}}]$ of possible conductance values, the weights $w_{i,j}$ are first scaled by a factor $\lambda_{i,j}$, such that

$$w^s_{i,j} = \lambda_{i,j} w_{i,j} \,, \tag{4}$$

where $\lambda_{i,j} = \frac{\mathrm{g_u} - \mathrm{g_{min}}}{w^u}$, with w^u representing the maximum absolute weight value. In addition, $\mathrm{g_u}$ is chosen depending on the desired trade-off between accuracy and power consumption, and it only needs to be smaller than the maximum physical conductance value $\mathrm{g_{max}}$. Note that the scaling factor $\lambda_{i,j}$ may vary for different parts of the neural network, and in Section V, we will study cases where $\lambda_{i,j}$ can vary from layer to layer, or from column to column.

After computing the scaled weights, we compute the positive and negative memristors values as

$$g^{(+)}_{i,j} = {w^s_{i,j}}^{(+)} + \mathrm{g_{min}} \,, \tag{5}$$

$$g^{(-)}_{i,j} = {w^s_{i,j}}^{(-)} + \mathrm{g_{min}} \,, \tag{6}$$

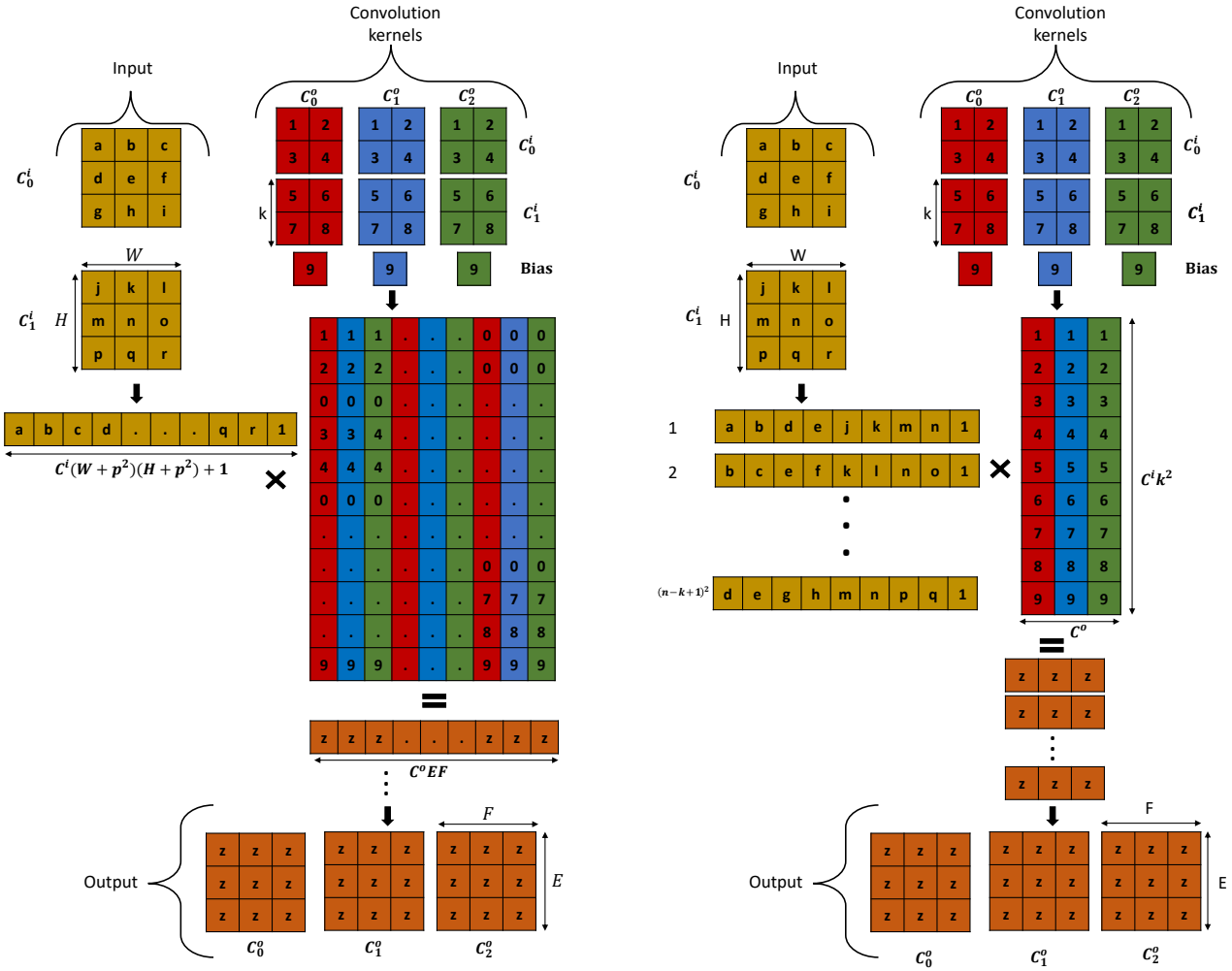
where ${w^s_{i,j}}^{(+)} = \lfloor \frac{\mathrm{sgn}(w^s_{i,j})+1}{2} \rfloor w^s_{i,j}$ and ${w^s_{i,j}}^{(-)} = \lfloor \frac{\mathrm{sgn}(w^s_{i,j})-1}{2} \rfloor w^s_{i,j}$. For simplicity, we define $g_{i,j}$ as $g_{i,j} = g^{(+)}_{i,j} - g^{(-)}_{i,j}$. We note that the random variables $G^{(+)}_{i,j}$ and $G^{(-)}_{i,j}$, which appear in equation (3), are the noisy versions of $g^{(+)}_{i,j}$ and $g^{(-)}_{i,j}$, respectively.

C. Computation models for DNNs and CNNs

We consider a DNN as a sequence of layers, where each layer corresponds to either a MVM or to other types of linear or non-linear transformations. Here, we present the different types of DNN layers that we consider in our analysis, and we explicitly describe their computation models for memristor crossbars. Note that the parts of a DNN that heavily rely on MVMs are fully-connected layers and convolutional layers. Hence, we consider that only these layers are implemented using memristors; other functions in the network such as pooling and non-linear activation functions are assumed to be processed by digital circuits that are not affected by noise, as in [37], [38]. In the following discussions, we consider that the input and output of each layer is a three-dimensional tensor. The input, denoted by X, has size $H \times W \times C^i$, while the output, denoted by Z, has size $E \times F \times C^o$, where C^i and C^o correspond to the number of input and output channels, respectively. The outputs shapes E and F can be expressed from the input size and the layer's parameters such as the padding and stride. They are used here to provide a common notation between the different types of layers.

1) Fully-connected layer: A fully-connected layer is represented by its weight matrix of size $HWC^i \times EFC^o$ as well as by a bias vector b of size EFC^o. To implement the corresponding dot-product operation using memristors, we construct two memristor crossbars of the same size as the weight matrix plus one additional row for the bias b. The conductance values $g^{(+)}_{i,j}$ and $g^{(-)}_{i,j}$ are computed following (4) and mapped onto the memristors. Then, the memristor crossbar outputs $\tilde{Z}_j^{(+)} = \sum_{i=1}^{L} r G^{(+)}_{i,j} X_i + B^{(+)}_j$ and $\tilde{Z}_j^{(-)} = \sum_{i=1}^{L} r G^{(-)}_{i,j} X_i + B^{(-)}_j$ are computed. We assume that the difference $\tilde{Z}_j = \tilde{Z}_j^{(+)} - \tilde{Z}_j^{(-)}$, as well as the necessary rescaling, $Z_j = \frac{\tilde{Z}_j}{\lambda_j}$, are computed by digital circuits outside of the memristor crossbars.

2) Convolutional layer: We consider a convolutional layer with C^i input channels and C^o output channels, a kernel size k, and a padding size p. A bias vector b may exist as well.



(a) *Unrolled-linear* mapping (b) *Unfold-repeat* mapping

Fig. 2: Mapping designs of the convolution operation onto memristor arrays.

We consider that a convolutional layer may be mapped to a memristor device using one of the following two approaches.

The first approach, which we call *unrolled-linear* (UL), consists in converting the convolutional layer into a fully connected layer. This is illustrated in Figure 2a, where the weight matrices of each kernel are unfolded and repeated into a large matrix. Then, the input is flattened into a vector which can be directly multiplied with the unrolled new weight matrix stored using memristors. Note that the weight matrix is therefore of size $C^i(H+2p)(W+2p) \times C^o EF$. This approach has the advantage that the computation of each layer can be done in one pass, meaning that only one MVM is performed. On the other hand, it requires storing a large, although sparse, matrix on memristor crossbars. Moreover, the larger the height of the matrix that is stored on a memristor crossbar, the more noise will be added to the result of the MVM carried out.

In the second approach, which we refer to as *unfold-repeat* (UR), a weight matrix is constructed for all the kernels [20], [39]. Figure 2b shows how the mapping of the operation translates to a memristor crossbar. The constructed matrix has size $k^2C^i \times C^o$ with each row containing all the flattened kernels for one output channel. This matrix is then stored on a memristor crossbar. To implement the convolution operation, the input is then unrolled into patches of size k^2C^i, and each patch is multiplied with the memristor crossbar. With this approach, a much smaller memristor array is needed, but the number of MVMs required is now $(H-k+1)(W-k+1)$. In addition, as with the fully-connected layer, the existence of the bias b implies an extra row on the weight matrix, as displayed in Figures 2.

In terms of actual hardware implementation, the second convolution mapping is more practical. This is due to the required size of the memristor crossbar being much smaller, which reduces the amount of noise in the final computation. In the following, we consider that the equations proposed for the fully-connected layers also apply to the UL convolutional layers.

3) Other linear operations: We now underline that some non-linear operations such as batch normalization may be implemented by resorting to the previous UL and UR approaches described for convolutional layers. In this case, an equivalent UL or UR can be constructed. Alternatively, a preprocessing

fusion step may also be applied, as discussed in Sec. IV-A. In our theoretical discussions and experimental results, we use average pooling for the pooling operation due to its linear nature (compared to the often-used max-pooling) which facilitates the theoretical analysis.

4) Activation function: As previously mentioned, we consider the DNN non-linear operations to be implemented in digital circuits. Nonetheless, the first and second-order moments must be propagated through the activation function. For most activation functions, evaluating such moments may require an approximation of some operations, which can be computationally expensive and hence reduce the practicality of using a theoretical analysis compared to a Monte-Carlo evaluation of the MSE. Instead, we impose the use of the ReLU activation function and propose an efficient analytical formula to compute the first and second-order moments (see Section III-D).

III. Theoretical analysis

To evaluate the robustness to noise of a memristor-based DNN, we aim to compare the difference between the output of the unreliable DNN with the output of a DNN of the same architecture but using a standard digital hardware devoid of noise and perturbations, which we refer to as a reliable DNN. To achieve this, we compute the MSE between the output of the memristor-based DNN and the output of the reliable DNN. Denoting by Z the output of the reliable network and by $\tilde{Z}$ the output of the noisy variant, we can express the MSE as

$$\mathrm{MSE}[\tilde{Z}] = \mathbb{V}[\tilde{Z}] + (\mathbb{E}[\tilde{Z}] - Z)^2\,, \tag{7}$$

where $\mathbb{E}$ and $\mathbb{V}$ are the mean and variance operator, respectively. As we see from equation (7), the computation of the MSE requires two terms for each operation performed on the memristor-enabled computing device: the first and the second-order moments. Therefore, the goal of our theoretical analysis is to compute these moments at the output of each DNN layer.

Since we focus on fully-connected and convolutional layers that are implemented on memristor crossbars, they are affected by noise, and as such have a significant effect on the successive moments. On the other hand, some operations, namely batch normalization, average pooling, and the activation function, are performed on reliable digital circuits. However, they propagate the moments from earlier layers.

In the following discussions, we denote by μ and γ the mean and covariance matrix of the input X of a given layer, respectively, and present the detailed equations for calculating the mean and covariance of the output of the different layer types considered in this work.

A. Moment propagation for fully-connected layers

As mentioned in Section II, the noise introduced in each memristor weight is independent from the noise introduced in the other memristors. In other words, the only correlation between outputs is due to the computations performed by previous layers.

The random variable $G_{i,j} = G^{(+)}_{i,j} - G^{(-)}_{i,j}$, has mean $\mathbb{E}[G_{i,j}] = \lambda w_{i,j}$. If $G^{(+)}_{i,j}$ and $G^{(-)}_{i,j}$ have noise variance σ^2, then $G_{i,j}$ has variance $2\sigma^2$. Hence, the fully-connected layer computations presented in (3) followed by rescaling, can be rewritten as

$$Z_j = \frac{r}{\lambda}\left(\sum_{i=1}^{L} G_{i,j}X_i + B_j\right)\,, \tag{8}$$

where B_j is the scaled noisy bias. This leads to the following expressions for the first and second moments $\mathbb{E}[Z_j]$, $\mathbb{V}[Z_j]$ and $\mathrm{Cov}[Z_j, Z_{j'}]$ at the output of fully-connected layers.

Proposition 1 (First and second moment propagation for fully-connected layers)**.**

$$\mathbb{E}[Z_j] = r\left(\sum_{i=1}^{L} w_{i,j}\mu_i + b_j\right)\,, \tag{9}$$

$$\begin{aligned}\mathbb{V}[Z_j] = r^2\Bigg(\frac{2\sigma^2}{\lambda^2} + \sum_{i=1}^{L}\left(\frac{\sigma^2\mu_i^2}{\lambda^2} + \gamma_i^2 w_{i,j}^2 + \frac{\gamma_i^2\sigma^2}{\lambda^2}\right) \\ + \sum_{i=1}^{L}\sum_{i'=1,i'\neq i}^{L} w_{i,j}w_{i'j}\gamma_{i,i'}\Bigg)\,,\end{aligned} \tag{10}$$

and

$$\mathrm{Cov}[Z_j, Z_{j'}] = r^2\sum_{i=1}^{L}\sum_{i'=1}^{L} w_{i,j}w_{i'j'}\gamma_{i,i'}\,. \tag{11}$$

B. Moment propagation for convolutional layers

Unlike the fully-connected layer case presented above, all outputs from the same output channel in the UR convolution mapping pass through the same memristors. Hence, the same noise realization is present at different computation stages. To take this into account, we introduce an additional term in the computation of the covariance between two outputs from the same output channel.

We rewrite the computation at the convolution layer followed by the rescaling as

$$Z_{c_o,i_o,j_o} = \frac{r}{\lambda_{c_o}}\sum_{c}\sum_{\substack{i=-k/2\\ j=-k/2}}^{k/2} G_{c_o,c,i,j}X_{c,i_o+i,j_o+j} + B_{c_o}\,. \tag{12}$$

Using this expression, we propose analytical formulas for the mean $\mathbb{E}[Z_{c_o,i_o,j_o}]$, variance $\mathbb{V}[Z_{c_o,i_o,j_o}]$, and covariance $\mathrm{Cov}[Z_{c_o,i_o,j_o}, Z_{c'_o,i'_o,j'_o}]$, of Z_{c_o,i_o,j_o}. For the covariance calculation, since several inputs will encounter the same noise realization through the same memristor, we analyze two cases covering the computation of the covariance : in the first case, two outputs are from the same output channel, and in the second case, they are from different output channels.

Proposition 2 (First and second moment propagation for convolutional layers)**.**

$$\mathbb{E}[Z_{c_o,i_o,j_o}] = r\sum_{c}\sum_{\substack{i=-k/2\\ j=-k/2}}^{k/2} \bar{w}_{c_o,c,i,j}\mu_{c,i_o+i,j_o+j} + b_{c_o}\,, \tag{13}$$

$$\mathbb{V}[Z_{c_o,i_o,j_o}] = r^2\Bigg(\sum_{c,c'}\sum_{\substack{i=-\frac{k}{2}\\ j=-\frac{k}{2}}}^{\frac{k}{2}}\sum_{\substack{i'=-\frac{k}{2}\\ j'=-\frac{k}{2}}}^{\frac{k}{2}} \varpi^{c,c',i,j,i',j'}_{c_o,c_o,i_o,j_o,i_o,j_o} + \frac{2\sigma^2}{\lambda^2_{c_o}}\Big(1+\sum_{c}\sum_{\substack{i=-\frac{k}{2}\\ j=-\frac{k}{2}}}^{\frac{k}{2}}\big(\gamma^2_{c,i_o+i,j_o+j}+\mu^2_{c,i_o+i,j_o+j}\big)\Big)\Bigg). \quad (14)$$

If two outputs are from the same output channel, *i.e.* if $c_o = c'_o$:

$$\mathrm{Cov}[Z_{c_o,i_o,j_o}, Z_{c_o,i'_o,j'_o}] = r^2\Big(\sum_{c,c'}\sum_{\substack{i=-\frac{k}{2}\\ j=-\frac{k}{2}}}^{\frac{k}{2}}\sum_{\substack{i'=-\frac{k}{2}\\ j'=-\frac{k}{2}}}^{\frac{k}{2}} \varpi^{c,c',i,j,i',j'}_{c_o,c_o,i_o,j_o,i'_o,j'_o} + \frac{2\sigma^2}{\lambda^2_{c_o}}\Big(1+\sum_{c}\sum_{\substack{i=-\frac{k}{2}\\ j=-\frac{k}{2}}}^{\frac{k}{2}}\gamma_{c,i_o+i,j_o+j,c,i'_o+i,j'_o+j} + \mu_{c,i_o+i,j_o+j}\mu_{c,i'_o+i,j'_o+j}\Big)\Big), \quad (15)$$

and otherwise, if $c_o \neq c'_o$:

$$\mathrm{Cov}[Z_{c_o,i_o,j_o}, Z_{c'_o,i'_o,j'_o}] = r^2\sum_{c,c'}\sum_{\substack{i=-\frac{k}{2}\\ j=-\frac{k}{2}}}^{\frac{k}{2}}\sum_{\substack{i'=-\frac{k}{2}\\ j'=-\frac{k}{2}}}^{\frac{k}{2}} \varpi^{c,c',i,j,i',j'}_{c_o,c'_o,i_o,j_o,i'_o,j'_o}, \quad (16)$$

with

$$\varpi^{c,c',i,j,i',j'}_{c_o,c'_o,i_o,j_o,i'_o,j'_o} = \bar{w}_{c_o,c,i,j}\bar{w}_{c'_o,c',i',j'}\gamma_{c,i_o+i,j_o+j,c',i'_o+i',j'_o+j'}.$$

C. Moment propagation for average pooling layers

We consider a 2D average pooling layer with stride s. The operation done at this layer is

$$Z_{c,i,j} = \frac{1}{s^2}\sum_{k=i}^{i+s}\sum_{l=j}^{j+s} X_{c,k,l}\,. \quad (17)$$

Since the average pooling layer is a linear operation, its moments $\mathbb{E}[Z_{c,i,j}]$, $\mathbb{V}[Z_{c,i,j}]$ and $\mathrm{Cov}[Z_{c,i,j}, Z_{c',i',j'}]$ can be computed exactly.

Proposition 3 (First and second moment propagation for average pooling layers)**.**

$$\mathbb{E}[Z_{c,i,j}] = \frac{1}{s^2}\sum_{k=i}^{i+s}\sum_{l=j}^{j+s}\mu_{c,k,l}\,, \quad (18)$$

$$\mathbb{V}[Z_{c,i,j}] = \frac{1}{s^4}\sum_{k=i}^{i+s}\sum_{l=j}^{j+s}\sum_{m=i}^{i+s}\sum_{n=j}^{j+s}\gamma_{c,k,l,c,m,n}\,, \quad (19)$$

and (20)

$$\mathrm{Cov}[Z_{c,i,j}, Z_{c',i',j'}] = \frac{1}{s^4}\sum_{k=i}^{i+s}\sum_{l=j}^{j+s}\sum_{m=i'}^{i'+s}\sum_{n=j'}^{j'+s}\gamma_{c,k,l,c',m,n}\,. \quad (21)$$

D. Moment propagation for the activation functions

Activation functions introduce non-linearities in the network. Letting f denote the activation function, the operation done at this layer can be written as $Z_{i,j} = f(X_{i,j})$. Because of the non-linear nature of f, it can be complex to compute exactly the propagation of the first and second-order moments through the activation functions. This step is thus the only one in the proposed framework where some approximations are used. In order to compute the mean and variance of the moments after the activation function, we need to know the probability distribution of the moments at the input of the function. Since we cannot know the exact distribution, we assume the output of fully-connected and convolutional layers to follow a normal distribution. This assumption is supported by the fact that in the memristor computations, only the inputs are dependent, but not the noise added on each memristor. Therefore, we can apply the central limit theorem. The validity of this hypothesis was also empirically verified through simulations presented in Section VI.

In the case of the ReLU activation function considered in this paper, if the input follows a normal distribution, the output follows a one-sided truncated normal distribution. Using this assumption we can directly compute the closed-form expressions for the moments $\mathbb{E}[Z_{i,j}]$, $\mathbb{V}[Z_{i,j}]$ as follows.

Proposition 4 (First and second moment propagation for ReLU functions)**.**

$$\mathbb{E}[Z_{i,j}] = \frac{\gamma_{i,j}}{\sqrt{2\pi}}e^{-\frac{\mu^2_{i,j}}{2\gamma^2_{i,j}}} + \frac{\mu_{i,j}}{2}\left(1-\mathrm{erf}\left(\frac{-\mu_{i,j}}{\gamma_{i,j}\sqrt{2}}\right)\right) \quad (22)$$

and

$$\mathbb{V}[Z_{i,j}] = \left(\frac{\mu^2_{i,j}}{2}+\frac{\gamma^2_{i,j}}{2}\right)\left(1-\mathrm{erf}\left(\frac{-\mu_{i,j}}{\gamma_{i,j}\sqrt{2}}\right)\right) + \frac{\gamma_{i,j}\mu_{i,j}}{\sqrt{2\pi}}e^{-\frac{\mu^2_{i,j}}{2\gamma^2_{i,j}}} - (\mathbb{E}[Z_{i,j}])^2\,, \quad (23)$$

where erf is the Gauss error function.

Proof. We can express the mean and variance of the output of the activation function f when the input follows a normal distribution as

$$E[Z_{i,j}] = \int_{-\infty}^{\infty} f(x)\frac{1}{\gamma_{i,j}\sqrt{2\pi}}\exp\left(-\frac{1}{2}\left(\frac{x-\mu_{i,j}}{\gamma_{i,j}}\right)^2\right)dx \quad (24)$$

and

$$\mathbb{V}[Z_{i,j}] = E[Z^2_{i,j}] - E[Z_{i,j}]^2 = \int_{-\infty}^{\infty} f(x)^2\frac{1}{\gamma_{i,j}\sqrt{2\pi}}\exp\left(-\frac{1}{2}\left(\frac{x-\mu_{i,j}}{\gamma_{i,j}}\right)^2\right)dx - E[Z_{i,j}]^2\,. \quad (25)$$

We then replace f by the ReLU function and therefore we simply need to compute the following integrals:

$$\Psi_1 = \int_0^{\infty} x\frac{1}{\gamma_{i,j}\sqrt{2\pi}}\exp\left(-\frac{1}{2}\left(\frac{x-\mu_{i,j}}{\gamma_{i,j}}\right)^2\right)dx\,, \quad (26)$$

and

$$\Psi_2 = \int_0^\infty x^2 \frac{1}{\gamma_{i,j}\sqrt{2\pi}} \exp\left(-\frac{1}{2}\left(\frac{x-\mu_{i,j}}{\gamma_{i,j}}\right)^2\right) dx\,. \quad (27)$$

The solutions to Ψ_1 and Ψ_2 can be computed using the general forms of these normal integrals, as shown in [40]. □

Following [32], the covariance can also be computed through approximation using a Taylor expansion:

$$\mathrm{Cov}[Z_{i,j}, Z_{i',j'}] \approx f'(\mu_{i,j})\, f'(\mu_{i',j'})\, \gamma_{i,j,i',j'}\,. \quad (28)$$

E. Power consumption

We now use the previous expressions of the moments to compute the mean power usage of each memristor crossbar for the inference of one input. We separate the power usage of the memristor crossbars into two parts: the power usage of the memristors and the power usage of the TIAs.

1) Fully-connected layer and UL convolution: We first derive an estimation of the power consumption of the memristor computations $\mathbb{E}[P_{i,j}^{(\text{mem})}]$ and $\mathbb{E}[P_{\text{bias},j}^{(\text{mem})}]$ when using a fully-connected layer, which also directly extends to our UL convolution mapping.

Proposition 5 (Mean power usage of each memristor in a fully connected layer)**.**

$$\mathbb{E}[P_{i,j}^{(\text{mem})}] = (\lambda\,|w_{i,j}| + 2\,\mathrm{g}_{\min})(\gamma_i^2 + x_i^2) \quad (29)$$

$$\mathbb{E}[P_{\text{bias},j}^{(\text{mem})}] = \lambda\,|w_{\text{bias},j}| + 2\,\mathrm{g}_{\min}\,. \quad (30)$$

Proof. For the power consumption of memristors in crossbars, it is possible to express directly the power usage for a pair of memristors storing the positive and negative weight of the same position in the original weight matrix. Defining $P_{i,j}^{(\text{mem})} = P_{i,j}^{(\text{mem})(+)} + P_{i,j}^{(\text{mem})(-)}$, the power consumption of each pair of memristor can be written as $P_{i,j}^{(\text{mem})} = |G_{i,j}|\,X_i^2$. We then compute the mean $\mathbb{E}[G_{i,j}X_i^2]$. □

Moreover, the mean power consumption of each TIA, $\mathbb{E}[P_j^{(\text{TIA})(+)}]$ and $\mathbb{E}[P_j^{(\text{TIA})(-)}]$, is evaluated for both the positive and negative memristor arrays by the following expressions.

Proposition 6 (Mean power usage of each TIA in a fully connected layer)**.**

$$\mathbb{E}[P_j^{(\text{TIA})(+)}] = \lambda^2 \frac{\rho_i^{(+)^2} + \mu_i^{(+)^2}}{r}\,, \quad (31)$$

$$\mathbb{E}[P_j^{(\text{TIA})(-)}] = \lambda^2 \frac{\rho_i^{(-)^2} + \mu_i^{(-)^2}}{r}\,. \quad (32)$$

Proof. We can express the power consumption of each TIA as

$$P_j^{(\text{TIA})(+)} = r\left(\sum_{i=1}^{L} G_{i,j}^{(+)} X_i\right)^2 = \frac{\tilde{Z}_j^{(+)^2}}{r}\,, \quad (33)$$

$$P_j^{(\text{TIA})(-)} = r\left(\sum_{i=1}^{L} G_{i,j}^{(-)} X_i\right)^2 = \frac{\tilde{Z}_j^{(-)^2}}{r}\,. \quad (34)$$

We then simply express the mean of the squared outputs of the memristor computations as computed in Section III-A. □

Hence, the mean power consumption $\mathbb{E}[P_{\text{tot}}^{(\text{FC})}]$ of a fully-connected layer or of a convolutional layer with UL mapping can be computed based on Propositions 5 and 6.

Corollary 1 (Total power consumption of memristors of a fully-connected layer)**.**

$$\mathbb{E}[P_{\text{tot}}^{(\text{FC})}] = \sum_{j=1}^{L}\left(\sum_{i=1}^{L}\mathbb{E}[P_{i,j}^{(\text{mem})}] + \mathbb{E}[P_j^{(\text{TIA})(+)}] + \mathbb{E}[P_j^{(\text{TIA})(-)}]\right). \quad (35)$$

2) UR convolution: We consider a convolutional layer with kernel size k, C^i input channels and C^o output channels and $H \times W$ the height and width of each feature map. We can compute the mean power for each weight stored on a memristor $\mathbb{E}[P_{c_o,c_i,i,j}^{(\text{mem})}]$ and $\mathbb{E}[P_{\text{bias},c_o}^{(\text{mem})}]$ as follows.

Proposition 7 (Mean power usage of each memristor in a UR convolutional layer)**.**

$$\mathbb{E}[P_{c_o,c_i,i,j}^{(\text{mem})}] = \sum_{m=k/2}^{M-k/2}\sum_{n=k/2}^{M-k/2}(\lambda_{c_o}\,|w_{c_o,c_i,i,j}| + 2\,\mathrm{g}_{\min})(\gamma_{c_i,m,n}^2 + x_{c_i,m,n}^2) \quad (36)$$

and

$$\mathbb{E}[P_{\text{bias},c_o}^{(\text{mem})}] = (M-k+1)^2(\lambda_{c_o}\,|w_{\text{bias},c_o}| + 2\,\mathrm{g}_{\min}) \quad (37)$$

for each bias stored on a memristor.

Proof. Similarly to Proposition 5, we can express the power consumption of each memristor during inference except that, in this case, we must also count the number of times the MVM product is repeated to complete the computation for one input map:

$$P_{c_o,c_i,i,j}^{(\text{mem})} = \sum_{m=k/2}^{M-k/2}\sum_{n=k/2}^{M-k/2}|G_{c_o,c_i,i,j}|\,X_{c_i,m,n}^2\,. \quad (38)$$

We can then take the mean of this expression:

$$\mathbb{E}[P_{c_o,c_i,i,j}^{(\text{mem})}] = \sum_{m=k/2}^{M-k/2}\sum_{n=k/2}^{M-k/2}\mathbb{E}[|G_{c_o,c_i,i,j}|\,X_{c_i,m,n}^2]\,. \quad (39)$$

Similarly we can express the mean of each memristor bias as

$$\mathbb{E}[P_{\text{bias},c_o}^{(\text{mem})}] = \sum_{m=k/2}^{M-k/2}\sum_{n=k/2}^{M-k/2}\lambda_{c_o}\,|w_{\text{bias},c_o}|\,. \quad (40)$$

□

We also propose expressions for evaluating the mean power consumption, $\mathbb{E}[P_{c_o}^{\text{TIA}(+)}]$ and $\mathbb{E}[P_{c_o}^{\text{TIA}(-)}]$, of each TIA in a similar way.

Proposition 8 (Mean power usage of each TIA in a convolutional layer)**.**

$$\mathbb{E}[P_{c_o}^{\text{TIA}(+)}] = \sum_{m=k/2}^{M-k/2}\sum_{n=k/2}^{M-k/2}\frac{\mathbb{V}[Z_{c_o,m,n}^{(+)}] + \mathbb{E}[Z_{c_o,m,n}^{(+)}]^2}{r}\,, \quad (41)$$

and

$$\mathbb{E}[P_{c_o}^{\text{TIA}^{(-)}}] = \sum_{m=k/2}^{M-k/2} \sum_{n=k/2}^{M-k/2} \frac{\mathbb{V}[Z_{c_o,m,n}^{(-)}] + \mathbb{E}[Z_{c_o,m,n}^{(-)}]^2}{r}. \tag{42}$$

Proof. If $Z_{c_o,m,n}$ is the result of the convolution, we have

$$P_{c_o}^{\text{TIA(+)}} = \sum_{m=k/2}^{M-k/2} \sum_{n=k/2}^{M-k/2} \frac{Z_{c_o,m,n}^{(+)^2}}{r}, \tag{43}$$

and similarly for $P_{c_o}^{\text{TIA}^{(-)}}$. We then take the expectation. □

We can then express the mean of the total power consumption of one UR convolutional layer $\mathbb{E}[P_{\text{tot}}^{(\text{Conv})}]$ as a corollary of Propositions 7 and 8.

Corollary 2 (Total power consumption of memristors of a convolutional layer)**.**

$$\mathbb{E}[P_{\text{tot}}^{(\text{Conv})}] = \sum_{c_o=1}^{C^o} \left(\mathbb{E}[P_{c_o}^{(\text{TIA})^{(+)}}] + \mathbb{E}[P_{c_o}^{(\text{TIA})^{(-)}}] + \mathbb{E}[P_{\text{bias},c_o}^{(\text{mem})}] + \sum_{i=1}^{k} \sum_{j=1}^{k} \sum_{c_i=1}^{d} \mathbb{E}[P_{c_o,c_i,i,j}^{(\text{mem})}] \right). \tag{44}$$

IV. Implementation details

A runtime-efficient implementation of our theoretical analysis is essential to ensure the practicality of our approach compared to Monte-Carlo simulations. Algorithm 1 describes the operations of our theoretical analysis and serves as the blueprint for its implementation. Note that computing the theoretical equations for a given DNN and input tensor X is similar to computing its outputs, i.e., presenting an input to the DNN and forward propagating the output of each layer with its weights w to the next; with the exception that the inputs and outputs consists of the triplet μ, γ and P. In

Algorithm 1 DNN Memristor Squared-Error Estimator

$\mu_{b,c,i,j} \leftarrow x_{b,c,i,j}$
$\gamma_{b,c,i,j,k,l,m} \leftarrow 0$ ▷ Cov init. to 0
$P_{tot} \leftarrow 0$ ▷ Power init. to 0
for Layer l^i in the DNN **do**
  $P_{tot} \leftarrow P_{tot} + P(\mu, \gamma, l^i, w_i)$ ▷ Update the power
  $\mu \leftarrow \Phi_{\mathbb{E}}(\mu, l^i, w_i)$ ▷ Update μ
  $\gamma \leftarrow \Phi_{\text{Cov}}(\mu, \gamma, l^i, w_i)$ ▷ Update γ
  $\gamma_{b,c,i,j,c,i,j} \leftarrow \Phi_{\mathbb{V}}(\mu, \gamma, l^i, w_i)$ ▷ Diagonal of γ
end for

Algorithm 1, $\Phi_{\mathbb{E}}$, Φ_{Cov} and $\Phi_{\mathbb{V}}$ correspond to the theoretical equations for computing the mean, covariance and variance of a layer, respectively. There are a few efficiency issues with this algorithm if it is implemented naively. Namely, the covariance tensor γ tends to be very large for the first layers of the DNN since it grows quadratically in the number of output activations of the layer. This may result in out-of-memory errors early in the computations. However, note that μ shrinks with each successive layer in popular DNN architectures. Another source of improvement is to leverage the same data access to perform slightly different operations resulting in the mean $\Phi_{\mathbb{E}}(\cdot)$, the covariance $\Phi_{\text{Cov}}(\cdot)$, and the power $P(\cdot)$.

The aforementioned issues lead to a heavily memory-bound implementation, thus limiting the size of input batches and preventing the use of larger DNNs, datasets, and optimization techniques for the memristor parameters. We aim to alleviate such issues by carefully analyzing the operations in a given layer, as discussed next.

A. Fusing convolutional and batch normalization operations

Operator fusion is a well known optimization technique for DNN deployment by losslessly merging a set of sequential operations into one [41], [42]. A popular instance of such merging is by taking a sequence of batch-norm and convolution operations and modifying the convolution weights to incorporate the linear transformation that would have been performed by batch normalization. Note that, on top of promoting computational efficiency, there is also a reduction of the memory footprint. Hence, we perform convolutional and batch normalization operator fusion prior to the theoretical analysis in our experiments.

B. Lazy instantiation of the covariance tensor

There are two elements involved in the successive computation of the covariance γ. First, there is a tensor of large size $B \times C^i \times H \times W \times C^i \times H \times W$ initialized with zero values. Second, as we iterate through Algorithm 1, the dimensions of γ follow the size of the output of intermediate layers. In popular DNN architectures, the output size of the intermediate layers shrinks, lessening the memory burden of γ early on in the analysis implementation. Therefore, we alleviate such memory bottleneck by representing γ with its size, i.e. storing only the values of (C^i, H, W, C^i, H, W), and lazily instantiate it when needed. This delays the large tensor memory allocation to at least the second layer, which as noted might be of reduced size. This approach further simplifies some computations in the first layer.

V. Optimization

The maximal programming conductance value $\mathrm{g_u}$ has a direct impact on both the power consumption of the device and the MSE at the output of the DNN. Consequently, when designing a memristor-based implementation of a DNN, the choice of $\mathrm{g_u}$ values becomes a critical consideration. Our proposed theoretical framework allows exploring possible trade offs. Depending on whether the primary goal of the implementation is to minimize energy consumption or maximize accuracy, our framework can be used as an efficient tool to determine the most suitable $\mathrm{g_u}$ value to align with these objectives. To this end, we consider a DNN represented by a function $f(.)$, with a target input X and a set of parameters W that includes the weights and biases of the network. We consider that $\mathrm{g_u}$ belongs to the interval $[\mathrm{g_{min}}, \mathrm{g_{max}}]$, where possible $\mathrm{g_u}$ values correspond to different trade-offs between power consumption and MSE. While lower values lead to smaller power requirements but

increased MSE, higher values have the opposite effect. Based on this observation, we formulate the following optimization problem:

$$\begin{aligned} \min_{\mathrm{g_u}} \quad & \mathbb{E}(P_{tot}) \\ \text{s.t.} \quad & \mathbb{MSE}(f(X, W, \mathrm{g_u})) \leq \nu\,, \\ & \mathrm{g_{min}} < \mathrm{g_u} \leq \mathrm{g_{max}} \end{aligned} \tag{45}$$

where ν represents the target MSE to be achieved by the network. This optimization problem represents finding the best values for $\mathrm{g_u}$ which will minimize the power usage of the memristors for a desired MSE constraint.

Although the optimization problem (45) aims to satisfy a constraint on the MSE, this metric is not always the end goal in machine learning problems. For example, in the case of a classification problem, it is often the accuracy of the network that needs to be maximized. However, as will be shown in Section VI, the MSE can be considered as a good proxy for these other metrics, in particular for the accuracy in a classification task.

In what follows, we address the optimization problem (45) for three cases, called designs, of increasing complexity with respect to $\mathrm{g_u}$. Each design increases the degrees of freedom of the optimization compared to the previous one, so as to ideally achieve larger power gains:

1) In the *scalar design*, we consider only one value of $\mathrm{g_u}$ for the whole DNN. To do so, we compute w^u as the maximum absolute weight value of all the network weights. Therefore, we compute and apply the same scaling factor for all the weight matrices of the network.
2) In the *layer-wise design*, a distinct $\mathrm{g_u}$ is used for each fully connected or convolution layer in the DNN. In this case, we compute one w^u for each layer as the maximum absolute weight of a given layer. Hence, a different scaling factor is computed for each layer depending on w^u and the chosen value of $\mathrm{g_u}$.
3) Finally, in the *column-wise design*, we have a vector of $\mathrm{g_u}$ for each fully connected or convolution layer, which have the size of the number of outputs of the respective layers. For this, we consider a different value of w^u for each column of each weight matrix. A different scaling factor is then applied to each column of each memristor crossbar to give more flexibility for reducing the power usage of the memristors.

Note that each design is also associated with a different methodology to compute $\mathrm{w_{max}}$.

Due to the forms of the equations for computing the MSE and the power usage, no analytical way of solving this optimization problem could be proposed. Indeed, the problem is non convex, and also the results of a layer affect the following layers so it is not possible to compute them separately. With these issues in mind, to solve problem (45) related to each design, we propose to use a heuristic optimizing search, based on a genetic algorithm [43]. Genetic algorithm have the advantages of more easily avoiding local optima and they can be used with a fitness function of any form.

Algorithm 2 describes the genetic optimization process used. In a first step, we initialize the starting population composed of N_{pop} $\mathrm{g_u}$ vectors. To generate our initial population, we add random gaussian noise to our initial best $\mathrm{g_u}$ estimation noted $\mathrm{g_u}^{(\text{init})}$. Then, at each iteration until we reach the maximum number of generations desired N_{gen}, using our implementation of the theoretical framework, which we refer to as the operator $\Phi(\mathrm{g_u})$, we compute the MSE and power usage of each $\mathrm{g_u}$ in the population. We keep track of the MSE and power values achieved by each configuration $\mathrm{g_u}$ as dictionaries $\mathbf{MSE}$ and $\mathbf{P_{tot}}$. The $\mathrm{g_u}$ parameters that allows to have the least power usage while either satisfying the MSE constraint ν or being the closest to the constraint if no $\mathrm{g_u}$ can satisfy it are then selected using a tournament selection process [44]. A new population for the next generation is then created during the "reproduction" step. In this step, the selected individuals are recombined and have random perturbations added to promote the evolution of better solutions over time. The $\mathrm{g_u}$ with the lowest power usage in the final generation is considered the best solution $\mathrm{g_u}^*$ to our optimization problem.

Algorithm 2 Optimizing for g_u

Input: $\mathrm{g_u}^{(\text{init})}$, $\sigma^{(\text{init})}$, N_{pop}, ν
Output: $\mathrm{g_u}^*$
  $\xi \sim \mathcal{N}(0, \sigma^{(\text{init})})$ ▷ Sample Gaussian noise
  $\mathrm{g_u}^{\text{pop}} \leftarrow \mathrm{g_u}^{(\text{init})} + \xi$ ▷ Initialisation of population
  for $i \in \{1, 2, ..., N_{\text{gen}}\}$ **do**
    for each $\mathrm{g_u} \in \mathrm{g_u}^{\text{pop}}$ **do**
      $(\mathbf{MSE}[\mathrm{g_u}], \mathbf{P_{tot}}[\mathrm{g_u}]) \leftarrow \Phi(\mathrm{g_u})$ ▷ Evaluate solution
    end for
    $\mathrm{g_u}^{\text{pop}} \leftarrow$ Selection($\mathrm{g_u}^{\text{pop}}$, $\mathbf{MSE}$, $\mathbf{P_{tot}}$)
    $\mathrm{g_u}^{\text{pop}} \leftarrow$ Reproduction($\mathrm{g_u}^{\text{pop}}$)
  end for
  $\mathrm{g_u}^* \leftarrow \arg\min_{\mathrm{g_u}} \mathbf{P_{tot}}[\mathrm{g_u}]$
  return $\mathrm{g_u}^*$

Note that depending on the considered design for $\mathrm{g_u}$, the number of parameters to optimize may vary, which directly influences the number of iterations of the genetic algorithm required to converge toward a good solution. To reduce the optimization time, we use the solution found in the case of a scalar $\mathrm{g_u}$ to initialize the optimization problem in the case of a layer-wise $\mathrm{g_u}$. Similarly, we use the solution of the layer-wise $\mathrm{g_u}$ to initialize the optimization of the column-wise $\mathrm{g_u}$.

VI. Experimental results

This section presents numerical results comparing our theoretical analysis to Monte-Carlo simulations for several types of DNNs. It also provides optimization results for the three considered designs. In addition, it provides a study of the time efficiency of our implementation of the theoretical framework described in this paper.

A. Experimental setups

To assess the accuracy of our theoretical analysis and to evaluate our optimization process, we used two different neural networks trained on different tasks: we trained a small DNN on a regression problem, and a larger DNN on a classification

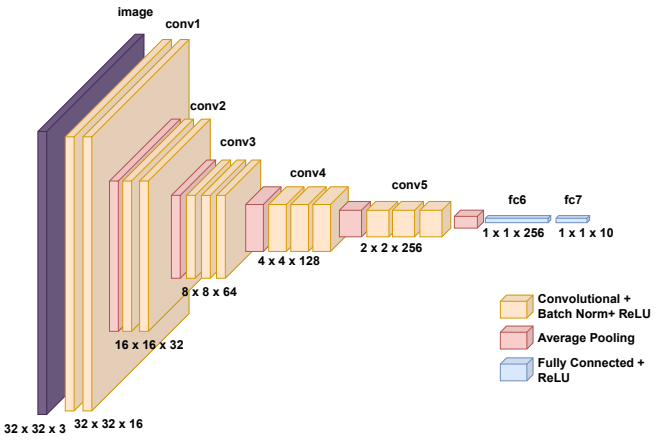


Fig. 3: Architecture of the classification network used in our experiments.

problem. Our goal is to showcase the ability of our analytical framework to efficiently scale to larger DNNs while still maintaining its accuracy.

For the regression problem, we used the naval dataset [45] of 11,934 datapoints made of 16 input features of a simulated naval vessel with the task to predict the decay coefficients of its gas turbine. We trained on this dataset a simple feed-forward network with a hidden layer of 50 neurons and a ReLU activation function for 200 epochs using the Adam optimizer [46] with a learning rate of 0.01. For the classification problem, we trained a convolutional neural network on the CIFAR-10 dataset [34]. The DNN architecture is presented in Figure 3 and consists of five blocks of convolutional layers with a kernel size of 3, a unit stride, and a varying number of filters: 16, 32, 64, 128, and 256 filters were considered. An average pooling layer follows each convolution block and the final part of the network is made of a classifier module composed of two fully-connected layers. We use the ReLU activation function, and dropout is applied after each average pooling during training. The network was trained for 200 epochs using stochastic gradient descent (SGD) with a momentum of 0.9, a weight decay of 5×10^{-4}, and an initial learning rate of 0.1 (reduced by a factor of 10 every 50 epochs).

In the following, we refer to a reliable DNN as a DNN implemented on a standard noiseless digital system, without any error on the computations and without the need for any weight scaling, contrary to the memristor-based DNN. Inference on the memristor-based DNN is simulated based on experimental memristor models [25], [26] as described in Section II. This signifies that for each Monte-Carlo simulation, a new realization of random noise following a normal distribution is computed and applied to the simulated memristor crossbars which are converted from the weights as detailed in Section II-B and that the DNN computations are then simulated as described in Section II-C.

B. Accuracy of the MSE estimation

1) Regression problem: Figure 4(a) shows the MSE between the output of the network implemented using memristors and the output of the same network implemented on a reliable system. The MSE is shown as a function of the conductance noise variance σ, while each curve corresponds to different values of $\mathrm{g_u}$. The results were computed using the theoretical analysis as well as using Monte-Carlo simulations. Similarly, Figure 4(b) shows the MSE between the memristor network and the ground truth of the data, which is a standard method for measuring the performance of a neural network for a regression problem. In both cases, we observe that the simulations and the analytical solution match almost perfectly. Moreover, we see that as the variance of the memristor noise decreases, the MSE also decreases and converges toward the minimum MSE achievable with a reliable network. This confirms this intuition that as $\mathrm{g_u}$ increases, the network is able to tolerate a higher level of noise for the same performance.

2) Classification problem: We next studied if the analytical formulas match the simulations for the considered classification DNN. Figure 5 shows the MSE between the output of the memristor network and a reliable network as a function of the noise variance σ. We plot the results for two implementations, one using the unfolded convolution mapping (UR), and one using the unrolled convolution mapping (UL). In all cases, $\mathrm{g_u}$ is chosen such that the scaling factor λ equals 1. We observe that the simulation almost completely matches with the theoretical results. On the right y-axis, we also show the accuracy of the noisy classification CNN depending on σ. We observe the ability of the network to tolerate some error in its computation in the low-noise regime while still maintaining its original accuracy. As mentioned in Section II-C2, with the UR convolution the amount of noise on the computation is lower than with the UL convolution when considering the same memristor noise variance σ. Hence, the MSE is lower for the UR convolution, which transposes to a better capacity for maintaining a high accuracy. Note that the link between MSE and classification accuracy is discussed further in Section VI-D2.

C. Run time comparison with Monte-Carlo simulations

Among other features, the theoretical analysis presented in Section III allows for evaluating the MSE of an unreliable network faster than using Monte-Carlo simulations. To quantify the execution time reduction compared to Monte-Carlo simulations, we first estimate the number of Monte-Carlo iterations needed using the following equation [47]:

$$n = \left[\frac{z_{\alpha/2} \overline{s}}{p \overline{x}} \right]^2 . \tag{46}$$

In this expression, $z_{\alpha/2}$ is a parameter related to the desired confidence interval, and it can be retrieved from tables of the cumulative distribution function for a normally distributed random variable [47], p is the percentage by which we allow the result to differ from the true value, and $\overline{s}$ and $\overline{x}$ are the sample standard deviation and sample mean, respectively. The values of $\overline{s}$ and $\overline{x}$ may be computed using the Monte-Carlo simulations through a substantial number of iterations to grasp an accurate estimate. Equation (46) allows us to compute the minimum number of iterations needed to compute an MSE

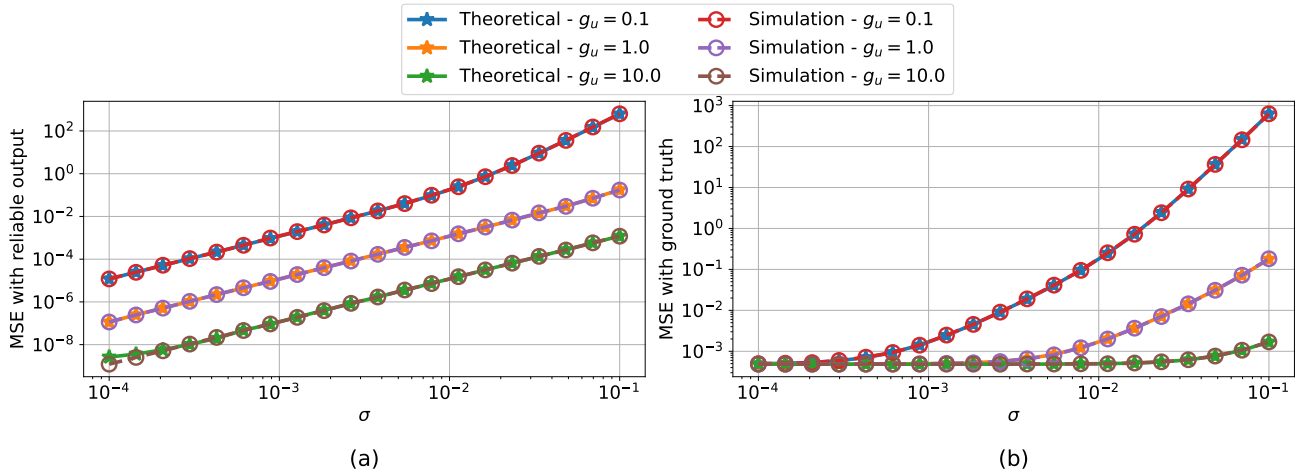


Fig. 4: Theoretical and Monte-Carlo computations of the MSE depending on the noise variance σ for different values of g_u.

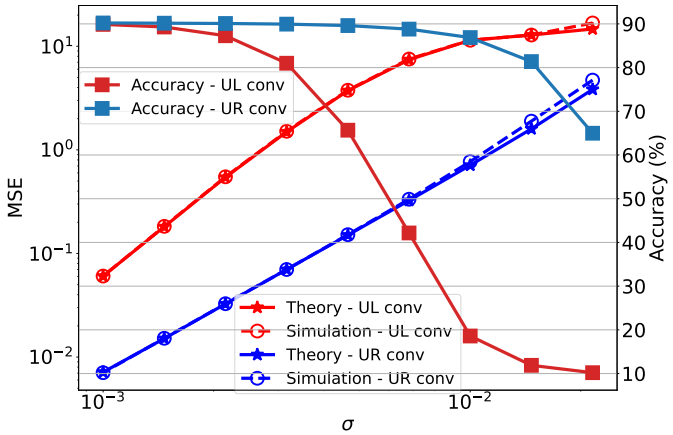


Fig. 5: MSE and accuracy comparison between the noisy and original DNNs for a classification problem depending on σ for the *Unrolled-Linear* convolution mapping (UL conv) and the *Unfold-Repeat* convolution mapping (UR conv).

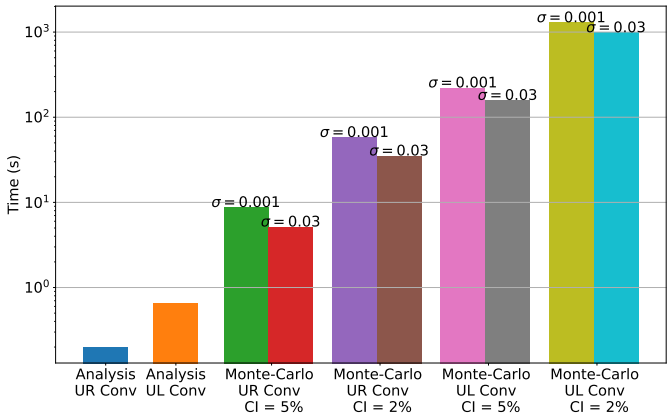


Fig. 6: Run time comparison between the theoretical analysis implementation and Monte-Carlo simulations.

which will fall within a specific interval from the true value for a certain confidence level. For example, using $z_{\alpha/2} = 1.96$ and $p = 0.01$ translates to a 95% confidence that the computed MSE will not be above or below a 1% difference from the true MSE.

Figure 6 shows a comparison between the run time needed for computing the MSE of an input from the theoretical analysis and from Monte-Carlo simulations on the classification network. The run time is measured for both types of convolution mappings. For the simulations, we measure the Monte-Carlo run time for two possible confidence intervals (which directly influence the number of Monte-Carlo iterations) and two σ values. The computations were done using one A100 Nvidia GPU and an AMD Milan 7413 CPU. The run time measurements for each data point were repeated 10 times and the results were then averaged. We observe that, for the unfold-repeat convolution mapping, the theoretical analysis is between 26 to 290 times faster than the Monte-Carlo simulations. The same trend is observed for the unrolled-linear mapping, where the acceleration factor is between 243 to 1982 depending on the desired confidence interval.

D. Optimization

1) Regression problem: We investigate the three designs introduced in Section V. In each case, we want to minimize the power usage under a specified constraint of the MSE between the output and the ground truth. The optimization is done using a genetic algorithm [43] with a population of size 100, a batch of 128 samples, and noise variance $\sigma^2 = 0.01$. The results are presented in Figure 7. We observe that layer-wise optimization achieves the same performance as the scalar case for less power usage. However, going from the layer-wise design to the column-wise does not seem to yield very significant gains in power consumption.

2) Classification problem: For our classification network, we first investigate if minimizing the MSE of the network is a good proxy for maximizing accuracy. To this end, we randomly generated different sets of g_u in the columns-wise case and, for each set, we compute both the MSE and accuracy. Each g_u set is represented by a point in Figure 8. We observe a

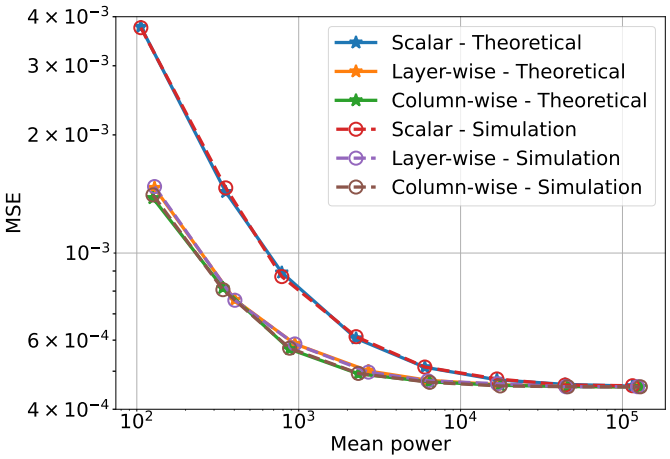


Fig. 7: Minimized MSE for different optimized networks subject to various power constraints for the regression network.

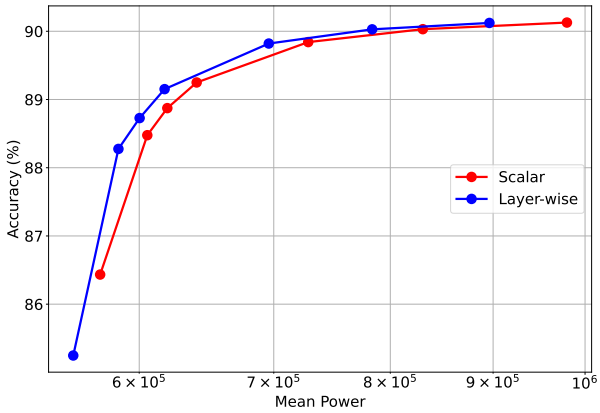


Fig. 9: Accuracy after minimizing the MSE for different cases of optimization and MSE constraints for the classification network.

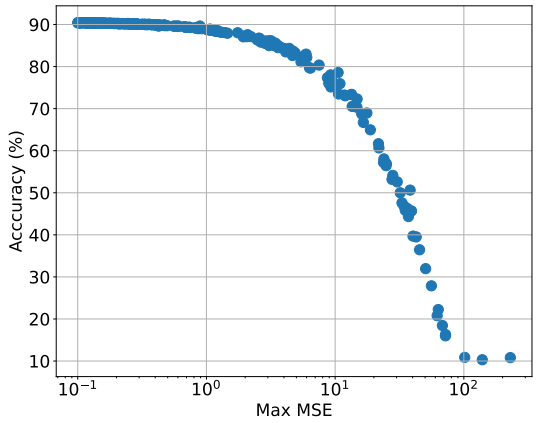


Fig. 8: Accuracy as a function of the maximum MSE for different sets of random g_u.

general trend where as the maximum of the MSE decreases, the accuracy of the network increases. This suggests that reducing the MSE of the network is a good way to also improve its accuracy.

We now try to solve the optimization problem for our classification network. As in the regression problem, we select a subset of the CIFAR-10 data set of 100 samples (10 of each class) and use the genetic algorithm to find the best g_u that minimizes the power usage for a given MSE constraint. Once the algorithm has converged to a solution for g_u, we use it to compute the accuracy of the network and its mean power usage over the whole test set. The results using different MSE constraints for the scalar design and the layer-wise design are presented in Figure 9. Just like for the regression network, preliminary results showed no differences between the layer-wise design and column-wise design. Due to the high computational cost of optimizing for the column-wise design in the case of a larger network, the data points for all different MSE constraints were not evaluated. As a result, the column-wise design is not depicted in the figure. We observe that having more degrees of freedom for the optimization by using the layer-wise design for g_u permits to achieve lower power usage for the same network performance. For instance, there is a 6% difference in power usage between the layer-wise and single g_u case when achieving an accuracy of 90%.

VII. Conclusion

In this work, we studied the mechanisms of error propagation in a memristor-based DNN using probabilistic analysis. This led us to a theoretical framework capable of accurately estimating the MSE of a high-performing, noisy network comprising a variety of layer types. We observed that such MSE estimation is significantly faster to obtain than with a Monte-Carlo analysis. We further proposed two mappings for convolutional layers to memristor crossbars and compared their impact on the error. The accuracy of the method was validated on multiple tasks and networks and its speed was demonstrated for different confidence intervals on the MSE estimate as well as well as several levels of noise. We have also shown how to use this approach to optimize the memristor hardware parameters with varying degrees of freedom, achieving significant power gains in both the regression and classification tasks.

The proposed framework is versatile and it can easily be adapted to other types of layers or error models. Albeit fast, the efficiency of our theoretical analysis implementation could still be improved by promoting better memory management and implementing dedicated software kernels, paving the way for applying it to larger networks and tasks.

References

[1] A. Pedram, S. Richardson, M. Horowitz, S. Galal, and S. Kvatinsky, "Dark memory and accelerator-rich system optimization in the dark silicon era," *IEEE Des. Test*, vol. 34, no. 2, pp. 39–50, Apr. 2017.

[2] G. Mordido, M. Van Keirsbilck, and A. Keller, "Instant quantization of neural networks using Monte Carlo methods," in *Workshop on Energy Efficient Mach. Learn. and Cogn. Comput. - NeurIPS Ed.*, 2019.

[3] ——, "Compressing 1D time-channel separable convolutions using sparse random ternary matrices," *arXiv preprint arXiv:2103.17142*, 2021.

[4] Y. Zhang, Y. Savaria, S. Zhao, G. Mordido, M. Sawan, and F. Leduc-Primeau, "Tiny CNN for seizure prediction in wearable biomedical devices," in *Annu. Int. Conf. IEEE Eng. in Medicine & Biology Soc.*, 2022.

[5] J. Deng, W. Dong, R. Socher, L.-J. Li, Kai Li, and Li Fei-Fei, "ImageNet: A large-scale hierarchical image database," in *IEEE Conf. on Comput. Vision and Pattern Recognit.*, 2009.

[6] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *IEEE Conf. on Comput. Vision and Pattern Recognit.*, 2016.

[7] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[8] A. Brock, S. De, S. L. Smith, and K. Simonyan, "High-performance large-scale image recognition without normalization," in *Int. Conf. Mach. Learn.*, 2021.

[9] S. Yin, Z. Jiang, M. Kim, T. Gupta, M. Seok, and J.-S. Seo, "Vesti: Energy-efficient in-memory computing accelerator for deep neural networks," *IEEE Trans. on Very Large Scale Integr. (VLSI) Syst.*, vol. 28, no. 1, pp. 48–61, 2020.

[10] C. Sakr and N. R. Shanbhag, "Signal processing methods to enhance the energy efficiency of in-memory computing architectures," *IEEE Transactions on Signal Processing*, vol. 69, pp. 6462–6472, 2021.

[11] A. Sebastian, M. Le Gallo, R. Khaddam-Aljameh, and E. Eleftheriou, "Memory devices and applications for in-memory computing," *Nature Nanotechnol.*, vol. 15, no. 7, pp. 529–544, Mar. 2020.

[12] D. B. Strukov, G. S. Snider, D. R. Stewart, and R. S. Williams, "The missing memristor found," *Nature*, vol. 453, no. 7191, pp. 80–83, May 2008.

[13] G. Pedretti and D. Ielmini, "In-memory computing with resistive memory circuits: Status and outlook," *Electronics*, 2021.

[14] S. Liu, Y. Wang, M. Fardad, and P. K. Varshney, "A memristor-based optimization framework for artificial intelligence applications," *IEEE Circuits and Syst. Mag.*, 2018.

[15] J. Wen, M. Ulbricht, E. Perez, X. Fan, and M. Krstic, "Behavioral model of dot-product engine implemented with 1T1R memristor crossbar including assessment," in *Int. Symp. on Des. and Diagnostics of Electron. Circuits Syst.*, 2021.

[16] A. J. Pérez-Ávila, G. González-Cordero, E. Pérez, E. P.-B. Quesada, M. Kalishettyhalli Mahadevaiah, C. Wenger, J. B. Roldán, and F. Jiménez-Molinos, "Behavioral modeling of multilevel HfO2-based memristors for neuromorphic circuit simulation," in *Conf. on Des. of Circuits and Integr. Syst.*, 2020.

[17] J.-C. Vialatte and F. Leduc-Primeau, "A study of deep learning robustness against computation failures," in *Int. Conf. on Adv. Cogn. Technol. and Appl.*, 2017.

[18] S. Henwood, F. Leduc-Primeau, and Y. Savaria, "Layerwise noise maximisation to train low-energy deep neural networks," in *IEEE Int. Conf. on Artif. Intell. Circuits and Syst.*, 2020.

[19] G. B. Hacene, F. Leduc-Primeau, A. B. Soussia, V. Gripon, and F. Gagnon, "Training modern deep neural networks for memory-fault robustness," in *IEEE Int. Symp. on Circuits and Syst.*, 2019.

[20] V. Joshi, M. Le Gallo, S. Haefeli, I. Boybat, S. R. Nandakumar, C. Piveteau, M. Dazzi, B. Rajendran, A. Sebastian, and E. Eleftheriou, "Accurate deep neural network inference using computational phase-change memory," *Nature Commun.*, 2020.

[21] G. Mordido, S. Chandar, and F. Leduc-Primeau, "Sharpness-aware training for accurate inference on noisy DNN accelerators," *arXiv preprint arXiv:2211.11561*, 2022.

[22] S. Moon, K. Shin, and D. Jeon, "Enhancing reliability of analog neural network processors," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, 2019.

[23] A. Shafiee, A. Nag, N. Muralimanohar, R. Balasubramonian, J. P. Strachan, M. Hu, R. S. Williams, and V. Srikumar, "ISAAC: A convolutional neural network accelerator with in-situ analog arithmetic in crossbars," in *ACM/IEEE Int. Symp. on Comput. Architecture*, 2016.

[24] C. Li, D. Belkin, Y. Li, P. Yan, M. Hu, N. Ge, H. Jiang, E. Montgomery, P. Lin, Z. Wang, W. Song, J. P. Strachan, M. Barnell, Q. Wu, R. S. Williams, J. J. Yang, and Q. Xia, "Efficient and self-adaptive in-situ learning in multilayer memristor neural networks," *Nature Commun.*, vol. 9, no. 1, Jun. 2018.

[25] C. Li, M. Hu, Y. Li, H. Jiang, N. Ge, E. Montgomery, J. Zhang, W. Song, N. Dávila, C. E. Graves, Z. Li, J. P. Strachan, P. Lin, Z. Wang, M. Barnell, Q. Wu, R. S. Williams, J. J. Yang, and Q. Xia, "Analogue signal and image processing with large memristor crossbars," *Nature Electron.*, vol. 1, no. 1, pp. 52–59, Dec. 2017.

[26] M. Hu, C. E. Graves, C. Li, Y. Li, N. Ge, E. Montgomery, N. Davila, H. Jiang, R. S. Williams, J. J. Yang, Q. Xia, and J. P. Strachan, "Memristor-based analog computation and neural network classification with a dot product engine," *Adv. Mater.*, vol. 30, no. 9, p. 1705914, 2018.

[27] C. Blundell, J. Cornebise, K. Kavukcuoglu, and D. Wierstra, "Weight uncertainty in neural networks," in *Proc. of the 32nd Int. Conf. on Int. Conf. on Machine Learning - Volume 37*, ser. ICML'15, 2015, p. 1613–1622.

[28] Y. Gal and Z. B. Ghahramani, "Bayesian convolutional neural networks with bernoulli approximate variational inference," in *Proc. of 4th Int. Conf. on Learning Representations, workshop track*, 2016.

[29] D. Dera, G. Rasool, and N. Bouaynaya, "Extended variational inference for propagating uncertainty in convolutional neural networks," in *IEEE 29th Int. Workshop on Machine Learning for Signal Processing (MLSP)*, 2019, pp. 1–6.

[30] D. Dera, G. Rasool, N. C. Bouaynaya, A. Eichen, S. Shanko, J. Cammerata, and S. Arnold, "Bayes-SAR net: Robust SAR image classification with uncertainty estimation using bayesian convolutional neural network," in *IEEE Int. Radar Conf. (RADAR)*, 2020, pp. 362–367.

[31] D. Dera, N. C. Bouaynaya, G. Rasool, R. Shterenberg, and H. M. Fathallah-Shaykh, "PremiUm-CNN: Propagating uncertainty towards robust convolutional neural networks," *IEEE Transactions on Signal Processing*, vol. 69, pp. 4669–4684, 2021.

[32] E. Dupraz, L. R. Varshney, and F. Leduc-Primeau, "Power-efficient deep neural networks with noisy memristor implementation," in *IEEE Inf. Theory Workshop*, 2021.

[33] J. Kern, S. Henwood, G. Mordido, E. Dupraz, A. Aissa-El-Bey, Y. Savaria, and F. Leduc-Primeau, "MemSE: Fast MSE prediction for noisy memristor-based DNN accelerators," in *IEEE Int. Conf. on Artif. Intell. Circuits and Syst.*, 2022.

[34] A. Krizhevsky and G. Hinton, "Learning multiple layers of features from tiny images," University of Toronto, Tech. Rep., 2009.

[35] M. Hu, J. P. Strachan, Z. Li, E. M. Grafals, N. Davila, C. Graves, S. Lam, N. Ge, J. J. Yang, and R. S. Williams, "Dot-product engine for neuromorphic computing: Programming 1T1M crossbar to accelerate matrix-vector multiplication," in *2016 ACM/EDAC/IEEE Des. Automat. Conf.*, 2016.

[36] V. Milo, C. Zambelli, P. Olivo, E. Pérez, M. K. Mahadevaiah, O. G. Ossorio, C. Wenger, and D. Ielmini, "Multilevel HfO2-based RRAM devices for low-power neuromorphic networks," *APL Materials*, vol. 7, no. 8, p. 081120, Aug. 2019.

[37] Z. Wang, C. Li, P. Lin, M. Rao, Y. Nie, W. Song, Q. Qiu, Y. Li, P. Yan, J. P. Strachan, N. Ge, N. McDonald, Q. Wu, M. Hu, H. Wu, R. S. Williams, Q. Xia, and J. J. Yang, "In situ training of feed-forward and recurrent convolutional memristor networks," *Nature Mach. Intell.*, vol. 1, no. 9, pp. 434–442, Sep. 2019.

[38] P. Yao, H. Wu, B. Gao, J. Tang, Q. Zhang, W. Zhang, J. J. Yang, and H. Qian, "Fully hardware-implemented memristor convolutional neural network," *Nature*, vol. 577, no. 7792, pp. 641–646, Jan. 2020.

[39] T. Gokmen, M. Onen, and W. Haensch, "Training deep convolutional neural networks with resistive cross-point devices," *Frontiers in Neuroscience*, vol. 11, Oct. 2017.

[40] D. B. Owen, "A table of normal integrals," *Commun. in Statist. - Simul. and Comput.*, vol. 9, no. 4, pp. 389–419, 1980.

[41] T. Chen, T. Moreau, Z. Jiang, L. Zheng, E. Yan, M. Cowan, H. Shen, L. Wang, Y. Hu, L. Ceze, C. Guestrin, and A. Krishnamurthy, "TVM: An automated end-to-end optimizing compiler for deep learning," in *Proc. 13th USENIX Conf. Operating Syst. Des. and Implementation*, 2018.

[42] W. Niu, J. Guan, Y. Wang, G. Agrawal, and B. Ren, "DNNFusion: Accelerating deep neural networks execution with advanced operator fusion," in *Proc. 42nd ACM SIGPLAN Int. Conf. Program. Lang. Des. and Implementation*, 2021.

[43] J. H. Holland, *Adaptation in natural and artificial systems: An introductory analysis with applications to biology, control, and artificial intelligence*. MIT Press, 1992.

[44] B. L. Miller and D. E. Goldberg, "Genetic algorithms, tournament selection, and the effects of noise," *Complex Syst.*, vol. 9, 1995.

[45] A. Coraddu, L. Oneto, A. Ghio, S. Savio, D. Anguita, and M. Figari, "Machine learning approaches for improving condition-based maintenance of naval propulsion plants," *J. Eng. for the Maritime Environ.*, vol. 230, no. 1, pp. 136–153, 2016.

[46] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[47] M. R. Driels and Y. S. Shin, "Determining the number of iterations for Monte Carlo simulations of weapon effectiveness," Naval Postgraduate School, Tech. Rep., 2004.