electronics

Article

Ternary Neural Networks Based on on/off Memristors: Set-Up and Training

Antoni Morell [1,*], Elvis Díaz Machado [1], Enrique Miranda [2], Guillem Boquet [3] and Jose Lopez Vicario [1]

1 Departament de Telecomunicació i Enginyeria de Sistemes, Universitat Autònoma de Barcelona (UAB), 08193 Bellaterra, Spain; elvis.diaz@uab.cat (E.D.M.); jose.vicario@uab.cat (J.L.V.)
2 Departament d'Enginyeria Electrònica, Universitat Autònoma de Barcelona (UAB), 08193 Bellaterra, Spain; enrique.miranda@uab.cat
3 Wireless Networks (WiNe) Research Laboratory, Internet Interdisciplinary Institute (IN3), Universitat Oberta de Catalunya (UOC), 08860 Castelldefels, Spain; gboquet@uoc.edu
* Correspondence: antoni.morell@uab.cat

Abstract: Neuromorphic systems based on hardware neural networks (HNNs) are expected to be an energy and time-efficient computing architecture for solving complex tasks. In this paper, we consider the implementation of deep neural networks (DNNs) using crossbar arrays of memristors. More specifically, we considered the case where such devices can be configured in just two states: the low-resistance state (LRS) and the high-resistance state (HRS). HNNs suffer from several non-idealities that need to be solved when mapping our software-based models. A clear example in memristor-based neural networks is conductance variability, which is inherent to resistive switching devices, so achieving good performance in an HNN largely depends on the development of reliable weight storage or, alternatively, mitigation techniques against weight uncertainty. In this manuscript, we provide guidelines for a system-level designer where we take into account several issues related to the set-up of the HNN, such as what the appropriate conductance value in the LRS is or the adaptive conversion of current outputs at one stage to input voltages for the next stage. A second contribution is the training of the system, which is performed via offline learning, and considering the hardware imperfections, which in this case are conductance fluctuations. Finally, the resulting inference system is tested in two well-known databases from MNIST, showing that is competitive in terms of classification performance against the software-based counterpart. Additional advice and insights on system tuning and expected performance are given throughout the paper.

Keywords: hardware neural networks; ternary networks; on/off memristors

check for updates

Citation: Morell, A.; Machado, E.D.; Miranda, E.; Boquet, G.; Vicario, J.L. Ternary Neural Networks Based on on/off Memristors: Set-Up and Training. *Electronics* **2022**, *11*, 1526. https://doi.org/10.3390/electronics11101526

Academic Editor: Kris Campbell

Received: 25 March 2022
Accepted: 3 May 2022
Published: 10 May 2022

1. Introduction

Neuromorphic computing, which imitates the principle behind biological synapses with a high degree of parallelism, has recently emerged as a very promising candidate for novel and sustainable computing technologies [1]. Among these technologies, neuromorphic systems based on hardware neural networks (HNNs) implemented with memristive devices have emerged as a promising solution for building energy-efficient computing frameworks for solving most of the tasks carried out in machine learning [2–4]. This is because memristors (1) behave as a resistor with memory that is electrically programmable and matches the functionality of the connections in a software neural network and (2) are efficiently integrated thanks to the crossbar array structure (i.e., aggressive size scaling is possible) [1,5,6].

Focusing on specific implementations that use memristors based on the crossbar array structure, it is worth noting first that this is one common approach found in the literature [7]. By using this structure, vector-matrix multiplications, which are a fundamental building block in all types of neural networks, are efficiently implemented by following an analog approach (i.e., by adding current flows). The efficiency of the operation is both in terms of (1) power consumption, because the involved currents are small, and (2) computational time [8], because the whole operation is performed by reading the outputs of the array. Note that a vector-matrix operation in software has a computational time that is $\mathcal{O}(m \cdot n)$ and that individual memristors in the crossbar array play the role of the matrix coefficients or, in terms of neural networks, the weights. The interested reader can find in [9] a specific sound localization application based on memristor arrays. Energy consumption is reduced a factor of 184 with regard to the existing Application-Specific Integrated Circuit (ASIC) design.

Memristor-based networks can be trained by offline (or ex situ) or online (or in situ)-learning methods. In the first case, which is the focus of this manuscript, the weights are calculated on a precursor software-based network and then imported sequentially into the crossbar circuit. In the second case, training is implemented in situ in hardware and only for small neural networks [10], so the weights are adjusted in parallel, which is significantly more demanding [5]. In both cases, a high precision weight import is required to implement complex networks and achieve the expected performance when the network is operating. However, various properties of memristors are known to negatively affect the performance of neuromorphic systems [1]. Specifically, the conductance response of any real nonvolatile memory (NVM) device exhibits non-idealities that can surface in the form of unreliable performance of the network. Those imperfections include non-linearity, stochasticity, varying maxima, asymmetry between increasing and decreasing responses, and unresponsive devices at low or high conductance [11–14]. For example, most memristive devices exhibit a nonlinear weight update, where the conductance gradually saturates [1]. In addition, related to HNNs from the perspective of high-performance computing, recent trends show a growing interest in hardware that is capable of accelerating both training and inference in neural networks, especially when dealing with deep learning schemes. That is the case, for example, with many Field-Programmable Gate Array (FPGA) implementations [15], which emphasize the idea of quantized neural network designs due to the nature of FPGA devices. In particular, binary [16] and also ternary [17] implementations have been raised as very interesting options. The main motivation of this alternative approach is the reduction of both power consumption and the FPGA specs (required area). Memristor-based neural networks can also benefit from the power and area. However, the operational principles of memristors are completely different to those found in FPGAs, and at the end of the day, all HNN solutions require solving very specific challenges, as far as a straightforward conversion from the ideal (or software-based) model does not exist. As commented above, one of the challenges in memristor-based neural networks, which work from an analog perspective, is the development of reliable weight implementation due to the variability that is common to all nano-electronic devices but is significantly important in memristors [18].

In that direction, the authors of [4] stated that many issues still need to be resolved at the material, device, and system levels to simultaneously achieve high accuracy, low variability, high speed, energy efficiency, a small area, low cost, and good reliability. Thus, the first step is to obtain memristor-based networks that are competitive in comparison to software-based networks. In order to achieve that, we need to cope with the hardware. This can be accomplished at the hardware level with more advanced mitigation techniques or at the analgorithmic level by taking into account the non-idealities. In that sense, the authors of [10] presented a mask technique to capture the sneak path problem, stating that any kind of training incorporating the knowledge of the crossbar array behavior will likely improve the accuracy of memristor-based networks significantly. This idea was explored by several recent works following different strategies. In [19], for instance, a tailored training method was proposed to address the voltage drop due to the interconnected wire resistance. Basically, the voltage drop is estimated to recompute the weights at the forward propagation stage during the training procedure. In [20], the authors considered the mapping of neural network weights by analyzing the parasitic resistance effects at the different areas of the crossbar array. By identifying those hardware cells providing higher accuracy as “safe zones”, adaptive weight allocation was performed to properly map the weights to the hardware. In [21], the authors mathematically modeled the sensitivity of the output of the neural network with respect to hardware impairments. Then, the cost function of the training algorithm was adapted to consider this sensitivity as an additional term (i.e., the weights were calculated to minimize the impact of hardware impairments as well).

The aim of our work is also to consider hardware impairments during the design and training of the memristor-based neural network. To do so, we depart from software models that emulate the behavior of the memristor-based neural network. More specifically, this work is an extension of the work in [22], and we consider building ternary networks using crossbar arrays. The goal is to achieve performances close to the software models, even when we consider a simple configuration of the memristors operating like ON/OFF switches. It is worth noting that we adopt ternary weights because they have stronger expressive abilities than their binary counterpart [17]. As shown in Section 2, a ternary option does not modify the proposed crossbar array architecture, and the hardware remains the same (i.e., two conductance levels at the memristor weights).

The main contributions of this work are as follows:

- The behavior of a ternary memristor-based HNN adopting crossbar arrays is emulated;
- Practical configuration strategies to tune the crossbar array structure from a system-level designer point of view are proposed;
- An offline (ex situ) training mechanism is derived to optimize the neural network's weights by minimizing the impact of conductance imperfections in the memristors' hardware.

In what follows, Section 2 defines the problem under study, including the crossbar array architecture that we are considering to emulate ternary networks. Section 3 the encompasses configuration issues as well as the algorithm considered to fix the memristors to either the ON or the OFF status. Finally, Section 4 provides the experimental results, and Section 5 concludes the paper.

2. Scenario Description and Assumptions

Let us consider a generic feedforward neural network (FFNN) that is dedicated to a classification task, as depicted in Figure 1. The network has inputs $\boldsymbol{x} = [1, x_1, \ldots, x_N]^T$, where $(\cdot)^T$ stands for the matrix transpose and the first input is manually set to 1 in order to accommodate the bias term. The FFNN operates as described next. First, the inputs are linearly combined by means of a matrix multiplication with $\boldsymbol{W}^1$, thus generating the values $\boldsymbol{z}^1 = [z_1^1, \ldots, z_{N_1}^1]$, i.e., $\boldsymbol{z}^1 = \boldsymbol{W}^1\boldsymbol{x}$ (see Figure 1). Superindex 1 here stands for the first layer of the network. The values in $\boldsymbol{z}^1$ go through a nonlinear function f (typically the sigmoid, the hyperbolic tangent or the rectified linear unit) to generate the activations at the first layer (i.e., $\boldsymbol{a}^1 = [a_1^1, \ldots, a_{N_1}^1]$ with $a_i^1 = f(z_i^1)$). This process is repeated at the subsequent layers of the FFNN, such as the output at the second layer being computed from $\boldsymbol{a}^1$ by first computing $\boldsymbol{z}^2 = \boldsymbol{W}^2\boldsymbol{a}^1$ and then transforming the values in $\boldsymbol{z}^2$ by using the non-linear function f again. Finally, at the last layer, also called the output layer, f is replaced by the softmax function. In this case, the output is normalized (i.e., $\sum_{i=1}^{N_J} a_i^J = 1$), and the value of a_i^J indicates our confidence level in that $\boldsymbol{x}$ corresponds to the ith class. Therefore, the network takes the output with largest value as the resulting classification.

All the operations described above are computed in floating-point arithmetic. We will refer to it as the software implementation. In this work, we employ the crossbar array to compute the vector-matrix multiplications at the neural network layers (i.e., $\boldsymbol{z}^j = \boldsymbol{W}^j\boldsymbol{a}^{j-1}$). Figure 2 depicts the operational principle of a crossbar array. Let us first consider the memristor in its linear zone, where it can be modeled simply as a resistor of conductance value g (adjustable) so that the memristor current is $i = g \cdot v$ when the voltage v is applied. By scaling this to a crossbar of the size $M \times N$ and arranging the conductance values in the matrix $\boldsymbol{G}$, we have $\boldsymbol{i} = \boldsymbol{G}\boldsymbol{v}$ with $\boldsymbol{i} = [i_1, \ldots, i_M]^T$ and $\boldsymbol{v} = [v_1, \ldots, v_N]^T$ (see Figure 2). In other words, the collected currents at the output of the crossbar array are in fact a vector-matrix multiplication between the input voltages $\boldsymbol{v}$ and the memristor conductances in $\boldsymbol{G}$.

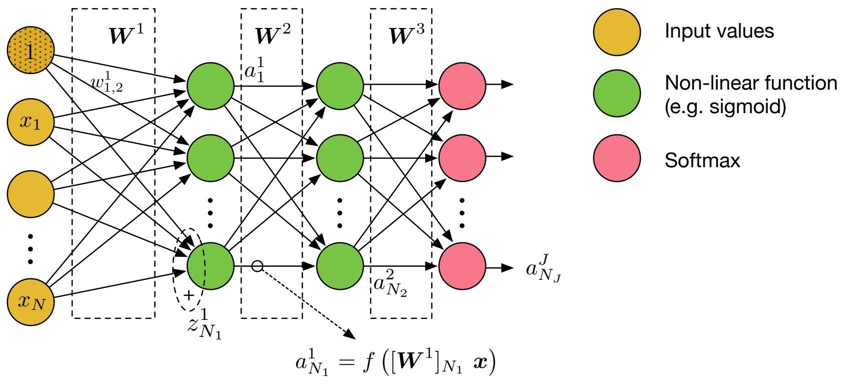

Figure 1. Feedforward neural network. $[\boldsymbol{W}^1]_{N_1}\ \boldsymbol{x}$ represents the matrix multiplication of the last row in $\boldsymbol{W}^1$ with the column vector $\boldsymbol{x}$, which includes all the input values plus the '1' to accommodate the bias term.

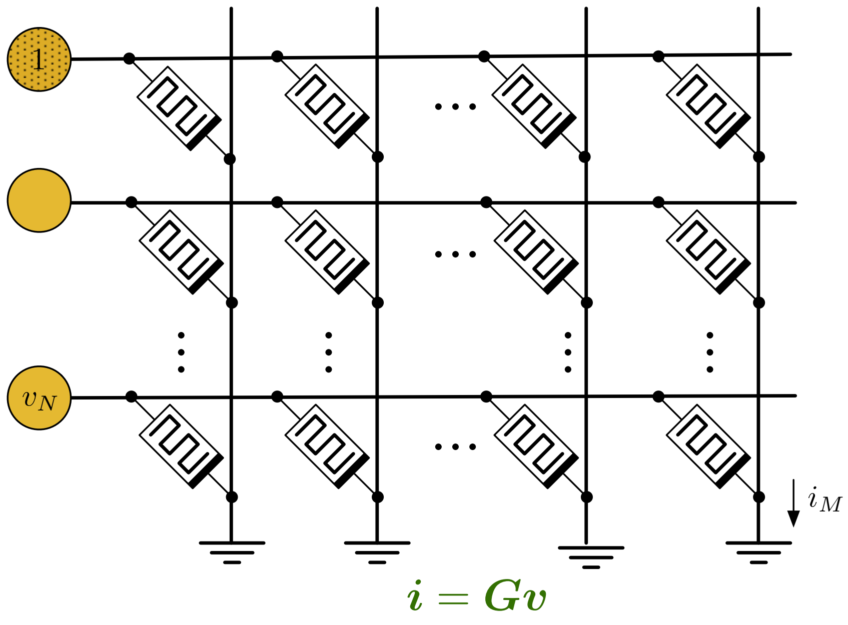

Figure 2. Crossbar array as a matrix multiplication.

Let us briefly comment on the linearity of the memristor we are considering. According to the memdiode model [23], the I–V characteristic of a memristor reads as

$$I = I_0(\lambda)\sinh[\alpha(V - IR)] \tag{1}$$

where I_0 is an increasing function of the parameter $0 \leq \lambda \leq 1$ (the memory state), R is the series resistance, and α is a fitting parameter. Notice that Equation (1) is an implicit equation for the current I. Let us consider two extreme cases. The first is the high-resistance state (HRS) regime (with $\lambda = 0$). In this case, for low voltages, we have $\sinh(x) \approx x$, and the potential drop across the series resistance can also be neglected such that

$$I = I_0(0)\alpha V \tag{2}$$

Second, for the low-resistance state (LRS) (with $\lambda = 1$), the difference is that the potential drop across R cannot be disregarded, and so

$$I = I_0(1)\alpha(V - IR) \tag{3}$$

which can be solved as

$$I = \frac{I_0(1)\alpha}{1 + I_0(1)\alpha R}V \tag{4}$$

The linear regime of the memristor corresponds to a case in between these two extreme situations so that the corresponding conductance reads as

$$G(\lambda) = I_0(\lambda)\alpha(1-\lambda) + \frac{I_0(\lambda)\alpha}{1 + I_0(\lambda)\alpha R}\lambda \tag{5}$$

which is independent of the voltage (i.e., it behaves as a simple resistor). This is the regime we are considering in our paper.

From an FFNN application point of view, the weights in $\boldsymbol{W}$ in the software model are equivalent to the conductances in $\boldsymbol{G}$. Usually, both positive and negative weights are represented, even when we consider only two possible values as in binary neural networks [16]. We may add a third possible value, a zero, as in ternary networks so that a particular input or activation does not affect the net outputs. As shown below, this ternary option does not modify the proposed hardware architecture (based on two crossbar arrays), as the zero weight is built naturally by combining the same conductance levels with opposite polarization. We are also exploiting the advantage of having a higher granularity when compared with its binary counterpart, as proven in [17]. Note, however, that some differences between the software model (i.e., complementary metal–oxide–semiconductor (CMOS)-based) and the memristor-based model arise. We next list the considerations in this paper:

- We need to transform the output currents at the crossbar array to voltages by means of I-to-V converters. The scale factor of the I-to-V converters is defined as a_{I2V}.
- The input voltages to the different layers shall be in the linear zone of the memristor (i.e., in the range $[0, V_{max}]$). Therefore, we need to scale both the inputs and the activations, because these are the inputs to the next network layers.
- We use the sigmoid as the nonlinear function f, which ranges from 0 to 1. Therefore, a scale factor of an amplitude equal to $V_{\max}$ is required.
- Memristors are set to either LRS, where the conductance is set to g_H, or HRS, where the conductance is set to g_L (ON/OFF).
- Since the conductance values are strictly positive, a single crossbar array cannot emulate both positive and negative weights, as we have in the software model. To overcome this, we need a second crossbar array that considers the negative weights as depicted in Figure 3. Equivalently, the value of each weight in the software model $w^j_{m,n}$ is emulated by the combination $g^{+,j}_{m,n} - g^{-,j}_{m,n}$, where the superindexes + and $-$ distinguish the first and second crossbar arrays at the jth layer, respectively.
- The memristors are programmed ex situ; that is, we first compute in the software the weights of the memristor-based neural network (considering non-idealities), and once obtained, we fix the conductances in the memristors. From that moment on, the crossbar arrays remain unchanged.
- The memristors are programmed to either g_L or g_H, but the conductance values actually written add a random additive component. In particular, $g^{j,\pm}_{m,n} \in \{g_L + n^{j,\pm}_{g_L,m,n}, g_H + n^{j,\pm}_{g_H,m,n}\}$, where $n^{j,\pm}_{g_L,m,n} \sim \mathcal{N}(0, \sigma^2_{g_L})$ and $n^{j,\pm}_{g_H,m,n} \sim \mathcal{N}(0, \sigma^2_{g_H})\ \forall\, m, n$. $\sigma^2_{g_L}$ and $\sigma^2_{g_H}$ represent the variances of the conductance in the HRS and LRS, respectively. We assume there are uncorrelated random additive components among the memristors.
- We consider $g_H - g_L$ to emulate the positive weight, say $+1$, $g_L - g_H$ to emulate the negative weight, say -1, and $g_L - g_L$ to emulate the null weight. Table 1 shows the set-up of the memristors in the positive and negative crossbar arrays and the corresponding weights. Alternatively, the null weight can be $g_H - g_H$, too. Note that our first option reduces the current and thus the power consumption.

Table 1. Set-up of memristors in $\boldsymbol{G}^{j,+}$ and $\boldsymbol{G}^{j,-}$ and corresponding ternary weights.

$g_{m,n}^{j,+}$	$g_{m,n}^{j,-}$	**Ternary Weight**
g_H	g_L	$+1$
g_L	g_L	0
g_L	g_H	-1

The goals in this work are the following:

- To adjust the conductance values in $\boldsymbol{G}^{j,+}$ and $\boldsymbol{G}^{j,-}$ (i.e., to decide which memristors are set to g_H and which are set to g_L);
- To adjust the value of g_H, taking into account that memristors can be configured in the range $g_H \in [g_H^{min}, g_H^{max}]$. Note that all the memristors are programmed to the same g_H value;
- To adjust the value of a_{I2V};
- To consider conductance randomness in the training process.

Note that we consider devices operating in the linear regime (i.e., in the low-voltage region), and thus nonlinearities in the I–V characteristic can be disregarded [23]. Beyond this point, the conductance of the devices may change as we move to the programming region, which is out of the scope of this work. Aside from that, line resistance, which does not affect the linearity of the devices, may also be considered, and the synaptic weights probably need to be recalculated because of the parasitic potential drops. If the devices operate in the low-voltage regime and the array is not too large, these voltage drops can be disregarded as well. This ultimately depends on the integration technology.

The next section describes the algorithm developed for the ex situ training of the memristor-based FFNN.

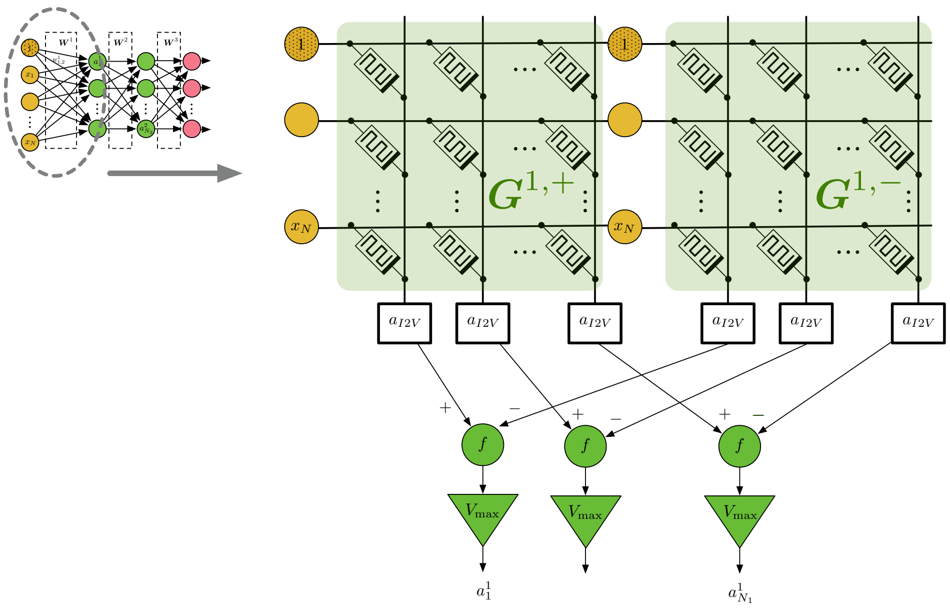

Figure 3. Implementation of an FFNN using crossbar arrays.

3. Proposed Algorithm

In this section, we consider the equivalent software model in Figure 1 in order to train and configure our memristor-based FFNN, depicted in Figure 3.

3.1. Training of Quantized Neural Networks

Training of the resulting quantized neural network is accomplished using the so-called backpropagation algorithm as described in [16]. The idea is simple: the forward pass in the backpropagation applies the quantization, whereas the backward pass computes the gradients as usual in order to update the weights. Stochastic and efficient optimization is accomplished by randomly shuffling the data and by training consecutively on small subsets of the data, respectively. Algorithm 1 shows the steps of the training process.

Algorithm 1 Algorithm for training a quantized network.

```
Input: Batch of training examples and labels
Output: $\widehat{W}^1, \dots, \widehat{W}^J$
    Initialization:
 1: Randomly initialize the weights $W_0^1, \dots, W_0^J$ at the $J$ layers in the FFNN
    LOOP Process
 2: for $k = 1$ to $N_{\text{epochs}}$ do
 3:   for $t = 1$ to $N_{\text{mini-batch}}$ do
 4:     $\widehat{W}_t^j = q\left(W_t^j\right) \forall j$   ($q$ is defined in Equation (6) below)
 5:     Forward propagation: compute network activations and outputs using $\widehat{W}_t^j$
 6:     Backward propagation: use $W_t^j$ to compute gradients
 7:     $W_{t+1}^j \longleftarrow W_t^j + \alpha \nabla W_t^j$
 8:   end for
 9: end for
10: return $\widehat{W}_t^j$
```

3.2. Ternarization

We considered the following quantization function $q(x)$, which is defined as

$$q(x) = \begin{cases} -g_H + g_L & x < -\Delta \\ 0 & -\Delta \le x \le \Delta \\ g_H - g_L & x > \Delta \end{cases} \tag{6}$$

We considered two options to fix Δ. The first one was to set it to a fixed value. The second one was to try to optimize the value of Δ according to the current weights at time t in W_t^j so that Δ was updated at each iteration of the algorithm. We followed the work in [17] to adjust the value of Δ as

$$\Delta_t = \frac{0.7}{N_{\text{total}}} \sum_{j=1}^{J} \mathbf{1}^T W_t^j \mathbf{1} \tag{7}$$

where $\mathbf{1}$ is the all-ones column vector and N_{total} is the total number of weights in the FFNN. The aim was to adapt the threshold to the current distribution of the weights. Note that Δ_t is the same for all network layers in our work, although different thresholds per layer could also be considered.

3.3. Adaptation of g_H

The proposed crossbar structure has two additional parameters to configure. Recall that we assumed an ON/OFF memristor model and that the conductances for the LRS and HRS were common to all memristors in the array. Notwithstanding, memristors can be programmed to different conductance values in the LRS. In this subsection, we develop the tuning of the conductance in g_H. Recall that in Algorithm 1, we configured the memristors in our network to either the LRS or the HRS, relying on backpropagation. In particular, note that the unquantized weights that are written in the memristor network, as the LRS will generally differ from g_H. In other words, usually we have

$$|w_{m,n,t}^j| \neq g_H - g_L \;\; \forall m,n,j \;\; s.t. \;\; |\hat{w}_{m,n,t}^j| = g_H - g_L \tag{8}$$

Therefore, we can use the values in $w^t_{m,n,t}$ to also update g_H and reach a consensus value g^*_H. Consider the following update rule:

$$\begin{aligned} g_H^{t+1} &= (1-\alpha_{g_H}) \cdot g_H^t \\ &+ \alpha_{g_H} \left[g_L + \frac{1}{N^{t+1}_{\neq 0}} \sum_{j=1}^{J} \mathbf{1}^T \mathcal{M}\left(\boldsymbol{W}^j_{t+1}\right) \mathbf{1} \right] \end{aligned} \tag{9}$$

where α_{g_H} is the forgetting factor and $\mathcal{M}$ is a masking function that operates element-wise in order to consider only the weights that have influence in g_H (i.e., not the null weights). When $\mathcal{M}$ is applied to a scalar in $\boldsymbol{W}^j_{t+1}$, say $w^j_{m,n,t+1}$, it produces the following output:

$$\mathcal{M}(w^j_{m,n,t+1}) = \begin{cases} -w^j_{m,n,t+1}, & \hat{w}^j_{m,n,t+1} = -g_H + g_L \\ 0, & \hat{w}^j_{m,n,t+1} = 0 \\ w^j_{m,n,t+1}, & \hat{w}^j_{m,n,t+1} = g_H - g_L \end{cases} \tag{10}$$

Additionally, $N^{t+1}_{\neq 0}$ is the total number of weights whose quantization is different from zero at iteration $t+1$.

However, note that a single weight in the neural network, say $w^j_{m,n,t}$, once quantized, requires three elements in our hardware model to be represented: two memristors (one in $\boldsymbol{G}^{j,+}_t$ and one in $\boldsymbol{G}^{j,-}_t$) and an I-to-V converter. In other words, $\hat{w}^j_{m,n,t}$ is represented in our physical model as $(g^{j,+}_{m,n,t} - g^{j,-}_{m,n,t}) \cdot a_{I2V}$. Furthermore, we set $a_{I2V} = 1/g_H$ (assuming $1/(g_H - gL) \approx 1/g_H$ when $g_H \gg g_L$) in order to map the weights $\{-1, 0, 1\}$, as is the case in software-based ternary networks [17]. However, the conductance variance $\sigma^2_{g_H}$, which does not depend on the particular value of g_H, now plays an important role, and the best choice is to set g_H to the largest value allowed. Note that after division by g_H in the I-to-V converter, the resulting conductance variance is also downsized.

In short, the discussion above is to point out that the best strategy is to set g_H as large as possible and then fine-tune our memristor-based neural network by adjusting the gain in the I-to-V converter, as we show next.

3.4. Adaptation of a_{I2V}

Let us consider that each output at the crossbar array could be adjusted separately (i.e., we have $a^j_{I2V,n}$). In this case, it is not complicated to compute the gradients for these parameters. It is similar to the weight gradients in backpropagation. For example, consider the scores $\boldsymbol{z}^J = \boldsymbol{a}^J_{I2V} \odot \boldsymbol{W}^J \boldsymbol{a}^{J-1}$, where $\odot$ stands for the Hadamard product at the output layer of the neural network. If we train it using cross-entropy (assuming a classification task) (i.e., $L = -\sum_{i=1}^{N_J} t_i \log y_i$, where N_J is the number of classes, t_i (0 or 1) are the targets and y_i are the network outputs), the gradients are found as follows:

$$\frac{\partial L}{\partial a^J_{I2V,i}} = \frac{\partial L}{\partial z^J_i} \cdot \frac{\partial z^J_i}{\partial a^J_{I2V,i}} = (y_i - t_i) \cdot \left(\left[\boldsymbol{W}^J \right]_i \right)^T \boldsymbol{a}^{J-1} \tag{11}$$

where $[\boldsymbol{X}]_i$ selects the ith row of matrix $\boldsymbol{X}$.

Let us analyze the effect of this gradient in the network, depicted in Figure 4. Assume that $t_i = 1$ (so the current example belongs to the ith class). Unless we get a perfect classification, $y_i < 1$ and the first term $(y_i - t_i)$ of the gradient will be negative. Therefore, if $\left(\left[\boldsymbol{W}^J \right]_i \right)^T \boldsymbol{a}^{J-1}$ is positive (i.e., we are at the positive side of the sigmoid or softmax), the gradient is negative, and $a^J_{I2V,i}$ should be increased according to Equation (11). The effect is to shrink the sigmoid or softmax in order to increase the value of y_i. If $\left(\left[\boldsymbol{W}^J \right]_i \right)^T \boldsymbol{a}^{J-1}$ is negative, we are on the negative x-axis of the sigmoid or softmax, and the update of $a^J_{I2V,i}$

stretches the curve. This increases the value of y_i and therefore diminishes the classification error. The reader can refer to Figure 4 for a graphical visualization of the discussion above. The analysis for $t_i = 0$ is similar and not included here for the sake of brevity.

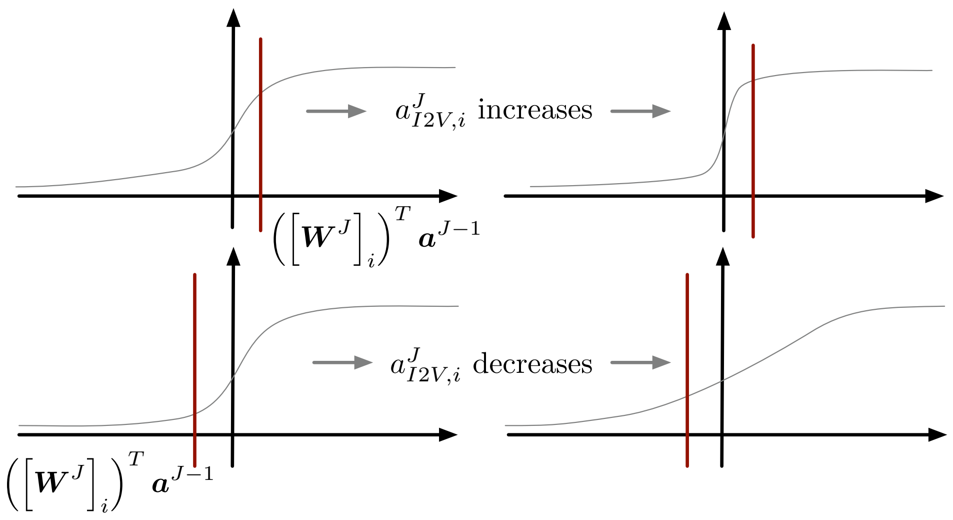
$a^J_{I2V,i}$ increases
$\left(\left[\boldsymbol{W}^J\right]_i\right)^T \boldsymbol{a}^{J-1}$
$a^J_{I2V,i}$ decreases
$\left(\left[\boldsymbol{W}^J\right]_i\right)^T \boldsymbol{a}^{J-1}$

Figure 4. Effect of adapting $a^J_{I2V,i}$ on the sigmoid.

Having separate conversion gains at all crossbar outputs that are individually adapted is a real possibility. However, since we assume a common converter value, we must build a consensus gradient from all individual gradients such that

$$\frac{\partial L}{\partial a_{I2V}} = \frac{1}{N_{\text{out}}} \sum_{j=1}^{J} \mathbf{1}^T \frac{\partial L}{\partial\, \boldsymbol{a}^j_{I2V}} \tag{12}$$

where $N_{\text{out}} = \sum_{j=1}^{J} N_j$ (i.e., the total number of outputs in the J layers of the FFNN). We can now apply gradient-descent-based solutions to optimize a_{I2V}.

Another option is to simply consider a_{I2V} as a hyperparameter of the neural network (it is a scalar value) and optimize.

3.5. Including Robustness in Perturbed Conductances

The last issue we consider is the perturbation of the conductances that are written to the memristors; that is, we want to set the memristor to a conductance level g_L or g_H, but the level we actually achieve differs by a Gaussian perturbation term (zero-mean and variances $\sigma^2_{g_L}$ and $\sigma^2_{g_H}$, respectively).

In order to cope with this physical impairment, we adopted an approach that resembles the training of quantized networks. Specifically, in the backward pass of backpropagation, we added a Gaussian term to the weights. The variance of that random contribution was set to $\sigma^2_{g_P}$, which is a hyperparameter of the network. In other words, we considered the following approach (in algorithmic style). This step substitutes step 7 in Algorithm 1.

```
$\boldsymbol{W}^j_{t+1} \leftarrow \boldsymbol{W}^j_t + \alpha\, \nabla \boldsymbol{W}^j_t + \mathcal{N}(0, \sigma^2_{g_P})$
```

The approach has a well-established foundation that connects to the regularization methods in neural networks. Primarily used in the context of recurrent neural networks, as described in [24] (Ch. 7.5), noise injection (i.e., adding random values to the weights) adds robustness to the network in the sense that the model learned is somehow insensitive to small variations in the weights. In other words, our approach can be interpreted as a form of regularization.

4. Experimental Results

In this section, we experimented with the proposed ternary network in order to evaluate the effects of the different adaptation mechanisms (conductance at the LRS and the conversion factor at the I-to-V stage), as well as the effect of quantizing the weights and the incorporation of weight variability during training. We considered two different datasets widely employed as benchmark datasets in machine learning: the Modified National Institute of Standards and Technology(MNIST) dataset [25] and the fashion MNIST dataset [26]. Both datasets consist of grayscale images of 28 × 28 pixels. The former contains images of handwritten numbers (from 0 to 9), whereas the latter also contains also different types (or classes) of images all related to clothes (e.g., t-shirts, pullovers or sandals, among others). In both cases, an 80–20 random split for training and testing was conducted.

In terms of neural network architecture, we considered an FFNN with two hidden layers of a size 1000 units/neurons. Taking into account 784 (=28 × 28) values at the input layer and 10 output classes, the whole architecture was 784–1000–1000–10, with a total of 7.85 G weights/parameters to be trained (including bias terms) in a full-software implementation and 15.7 G memristors to be set at either the LRS or HRS in the memristor-based neural network implementation. Training and evaluation were performed by means of Python programming using the Tensorflow library for deep neural networks [27]. The memristors were modelled in Python and Tensorflow according to the assumptions in Section 2.

Table 2 summarizes the electrical parameters considered in our experiments. We considered values that were in agreement with the state of the art of the memristive technology [10,28], but we also took into account larger conductance deviations in order to accommodate other fabrication technologies. Our goal here was to test the practical importance of synthesizing reliable devices in terms of conductance fluctuations. The conductance values are always relative to G_0, the quantum conductance.

Table 2. Electrical parameters in the memristor-based neural network.

Parameter	Value or Range
V_{max}	0.2 V
g_L	1
g_H	[14, 140]
σ_{g_L}	1
σ_{g_H}	1–10

Finally, ternarization was applied with the adaptive threshold in Equation (7), and the performance metric employed was classification accuracy, which measured the number of examples (images in this case) correctly classified with respect to the total number of images (i.e., it was the percentage of images correctly classified).

4.1. Adaptation of g_H

Our first experiment dealt with the adjustment of the conductance value for the LRS (i.e., g_H). In Figures 5 and 6, we considered violin plots that showed the distribution of accuracies obtained after 1000 realizations. Remember that memristor conductances incorporate a random term (i.e., $g^{+,j}_{m,n} = g_H + n^{+,j}_{g_H,m,n}$ and $g^{-,j}_{m,n} = g_L + n^{-,j}_{g_L,m,n}$). Although the weights computed during training remained unchanged, their mapping to memristor conductances changed from one realization to another due to the random term. That aside, we considered here $a_{I2V=1/g_H}$, the threshold used for the ternarization of the weights set as in Equation (7), and $\sigma_{g_P} = 0$. Figure 5 considers $\sigma_{g_H} = 1 \cdot G_0$, and Figure 6 considers $\sigma_{g_H} = 10 \cdot G_0$.

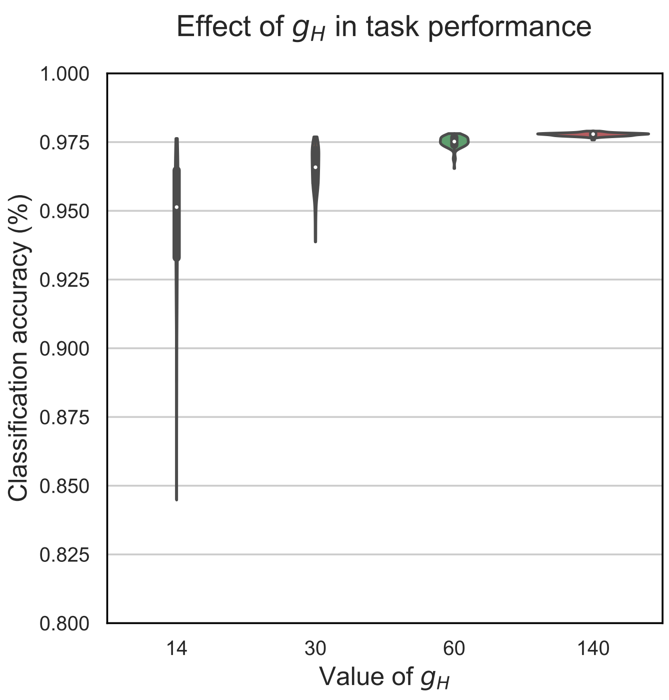

Figure 5. MNIST handwritten digit classification accuracy as a function of the value of g_H. Conductance fluctuations are $\sigma_{g_H} = 1 \cdot G_0$.

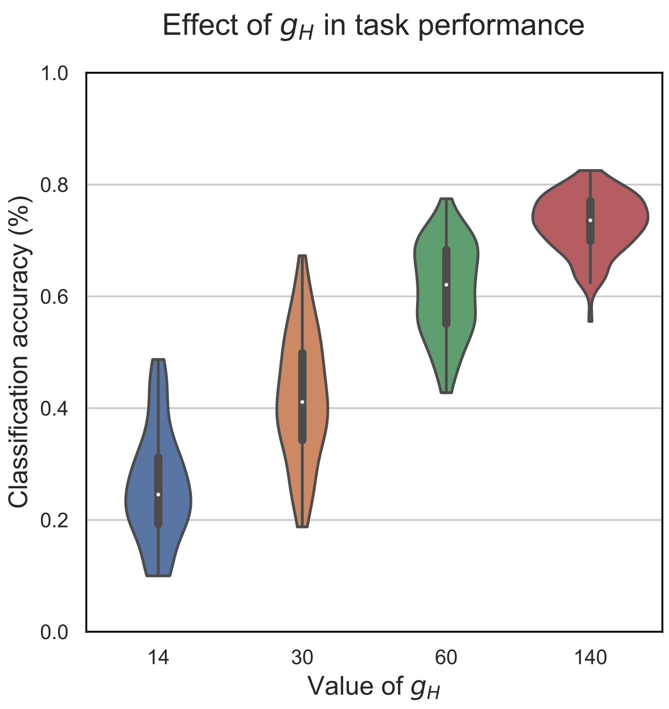

Figure 6. MNIST fashion classification accuracy as a function of the value of g_H. Conductance fluctuations are $\sigma_{g_H} = 10 \cdot G_0$.

As we can appreciate in Figure 5, where the conductance perturbations were moderate, the average accuracy was above 95% with all the tested adjustments of g_H. However, the distribution of accuracy values was spread out significantly more for the case $g_H = 14$ (ranging from 0.845 to 0.976), whereas the dispersion diminished for the case of $g_H = 30$ (ranging from 0.939 to 0.977) and practically vanished for $g_H = 60$ (ranging from 0.965 to 0.978) and more notably for $g_H = 140$ (ranging from 0.976 to 0.979).

Figure 6 involves experiments with a severe conductance perturbation. In this case, the classification task was more complex, and with equal network configuration, the performance dropped. We appreciated the dispersion in the accuracy distributions for all cases of g_H, although the dispersion tended to reduce as g_H increased. In this best case, the accuracy ranged from 0.551 to 0.825, so the difference between the max and min value was 0.274. In this application, it is important to note the mean values for the accuracy. For the two lowest conductance values, the mean accuracy was 25.8% for $g_H = 14 \cdot G_0$ and 41.9% for $g_H = 30 \cdot G_0$. This value grew up to 61.1% for $g_H = 60$ and to 73% for $g_H = 140$.

In conclusion, both experiments confirmed that g_H should be adjusted to the highest possible value (depending on the available technology) in order to achieve the best possible performance.

4.2. Adaptation of a_{I2V} and Robustness to Perturbed Conductance Values

In Figure 7, we tested how sensitive the classification accuracy was to adjustment of the value in a_{I2V} and to the weight perturbance introduced during training (i.e., σ_{g_P}), assuming $g_H = 140 \cdot G_0$. We considered both datasets under study (handwritten digits and fashion MNIST), and we plotted the classification accuracy as a function of $g_h \cdot a_{I2V}$, testing different combinations of σ_{g_H} and σ_{g_H}, particularly $\sigma_{g_H} \in \{1 \cdot G_0, 5 \cdot G_0, 10 \cdot G_0\}$ and $\sigma_{g_P} \in \{0, 1 \cdot G_0, 5 \cdot G_0, 10 \cdot G_0, 20 \cdot G_0, 50 \cdot G_0\}$. As we can appreciate in the figure, setting $a_{I2V} = 1/g_H$ (i.e., $a_{I2V} \cdot g_H = 1$) gave us a particularly good initial adjustment. In the applications tested, the plots show that the performance could be just slightly increased by choosing the optimal value of a_{I2V}, as long as the sensitivity around the initial adjustment was low. Note that the perturbance introduced during training (i.e., the value in σ_{g_P}) had a larger influence on the classification accuracy (i.e., the different accuracy curves became more separated), especially for the fashion MNIST dataset when $\sigma_{g_H} = 5 \cdot G_0$ and $\sigma_{g_H} = 10 \cdot G_0$. Note also that in general, the higher σ_{g_H} was, the more variation in performance we observed. As a rule of thumb, setting σ_{g_P} to a value in the range $[\sigma_{g_H}, 5 \cdot \sigma_{g_H}]$ provided a proper adjustment.

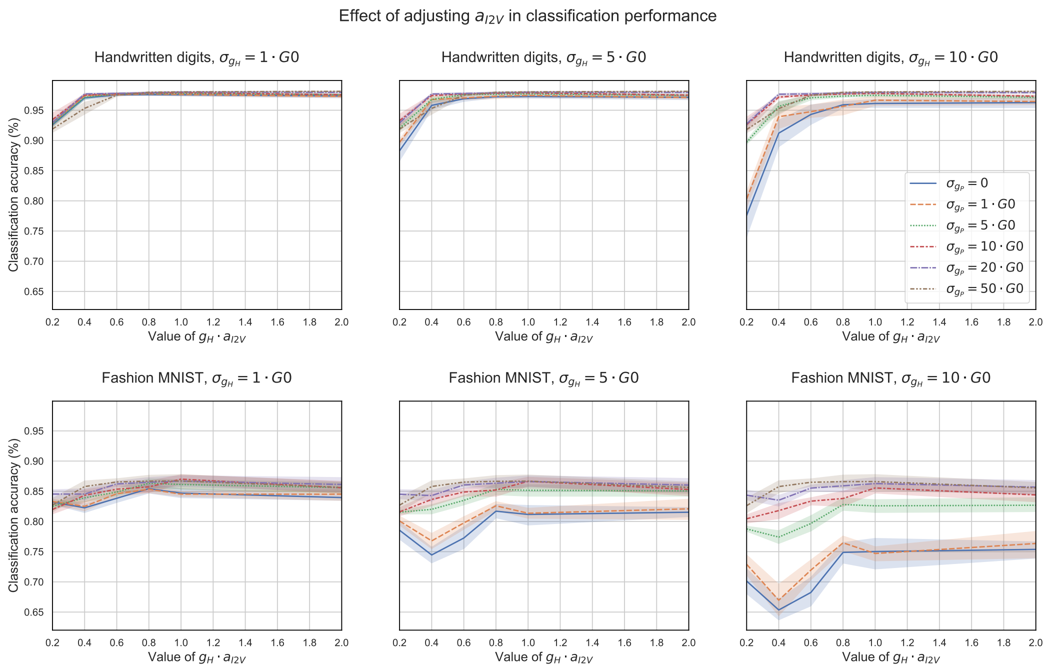

Figure 7. Classification accuracy as a function of the value of a_{I2V} and σ_{g_P} for $g_H = 140 \cdot G0$.

4.3. Comparison with the Software-Based Neural Network

We then tested how a memristor-based neural network (MBNN) compared to a software-based neural network (SBNN). For this occasion, we took into account ternarization of the weights as well as mitigation of the conductance perturbations and optimization of a_{I2V}. Aside from the reference network architecture (i.e., 784–1000–1000–10), we also considered 784–500–500–10 and 784–100–100–10 for the MBNN and SBNN models. The results reported the empirical cumulative density functions (ecdf). Note that the SBNN suffered no perturbation once the weights were fixed, but the performance slightly varied due to the random initialization of weights in training, too. In order to reflect this issue, we then considered 1000 realizations in total for each model containing 10 different training processes. In other words, the same set of weights was used to perform 100 inferences. Note that in the SBNN, all inferences that used the same set of weights produced the same results, whereas in the MBNN, this was not the case due to conductance fluctuations. We considered here the classification of the fashion MNIST dataset, which is a more complex task than handwritten digit classification, assuming $g_H = 140 \cdot G_0$ and $\sigma_{g_H} = 10 \cdot G_0$.

In Figure 8, we next compare the following methods: (1) SBNNs with 1000, 500 and 100 units in the two hidden layers; (2) an MBNN (1000 units in the hidden layers) with $\sigma_{g_P} = 0$, $a_{I2V} = 1/g_H$ (i.e., we did not consider conductance fluctuations in training), an MBNN with $\sigma_{g_P} = 10 \cdot G_0$, $a_{I2V} = 1/g_H$ (i.e., a default set-up assuming fluctuations in training) and a fine-tuned MBNN (in this case requires increasing σ_{g_P} to $50 \cdot G_0$); and (3) the fine-tuned MBNN version with 500 and 100 units in the hidden layers.

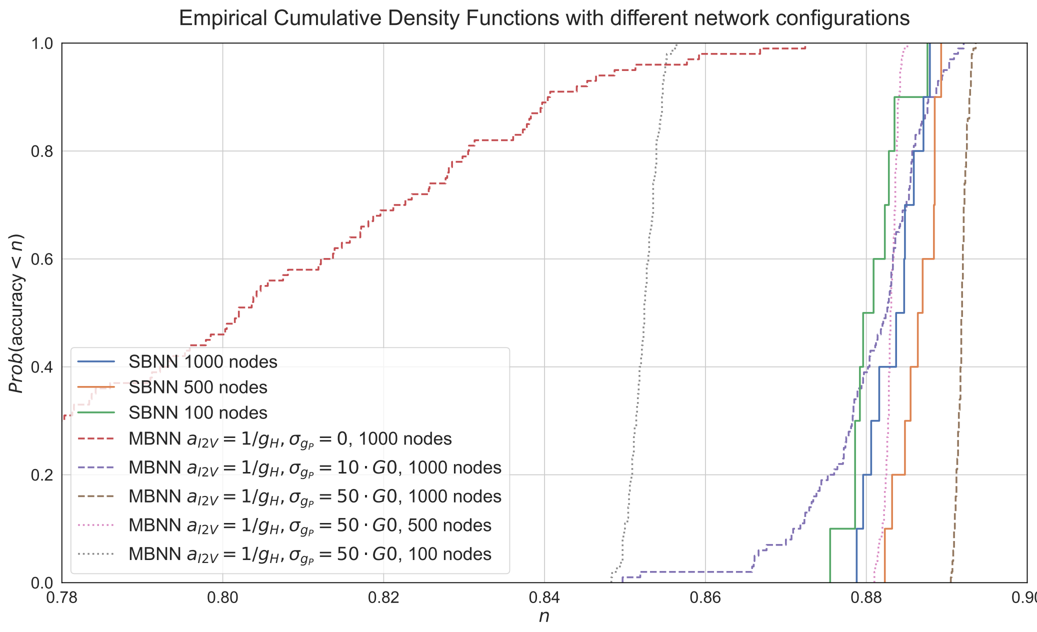

Figure 8. Performance of SBNN and MBNN with different configurations in the classification of fashion MNIST data.

The results essentially show two issues when we considered 1000 units at each hidden layer. First of all, there was the importance of considering memristor fluctuations during training. Note the spreading in the ecdf for $\sigma_{g_P} = 0$, which had a maximum value of 0.8724 and a minimum value of 0.6274 (i.e., the gap was 0.245). This gap significantly reduced to 0.042 in the case of the default set-up and practically vanished in the tuned MBNN and also the SBNN. Second, a properly tuned MBNN achieved a performance that was similar to the SBNN counterpart. If we look at the worst performance in all the set-ups, the SBNN archived an accuracy of 0.8788, the MBNN with $\sigma_{g_P} = 0$ yielded 0.6274 (a 28.6% reduction with respect to the SBNN), and the MBNN with $\sigma_{g_P} = 50 \cdot G_0$ obtained 0.8905 (a 1.3% increase with respect to the SBNN). This slightly better performance might have been due to the regularization effect produced when we included robustness to perturbed conductance values by means of σ_{g_P}. For the cases of 500 and 100 hidden units per layer, the MBNN performed close the SBNN for 500 units and suffered a reduction of about 3% in accuracy for the case with 100 hidden units. Therefore, weight quantization affected the performance more as the complexity of the model was further constrained.

4.4. Summary and Extension of Results

To summarize the results so far, we saw that the performance in general depended on the task complexity, the network configuration as well as on the memristor quality, where the larger the g_H and the lower the σ_{g_H}, the better. Regarding task complexity, Figure 7 plots the results of the exact same models applied to two different tasks. In the top row, the less complex task showed less variability among models such that a proper adjustment of σ_{g_P} was less critical. In the bottom row, the more complex task showed more variability and required a good adjustment of σ_{g_P}. In order to complete our analysis, now with a lower quality memristor,we could reproduce in Figure 9 the same experiments while considering $g_H = 14 \cdot G_0$ instead of $g_H = 140 \cdot G_0$. As we can appreciate in the figure, the classification accuracy in the different models was far more sensitive to the proper adjustment of σ_{g_P}. The worst performing models in the classification of handwritten digits (top row) then achieved accuracies around or below 20%, whereas the worst accuracy in Figure 7 was above 75%. Something similar occurred in the classification of clothes; the worst performing models achieved values around 20% whereas in Figure 7, the worst performance was above 65%.

Finally, we tested our memristor-based solution as part of a convolutional neural network (CNN) implementation applied to the classification of the fashion MNIST dataset. The input in this case was grayscale images of 28 × 28 pixels (i.e., 2D data). The configuration of our CNN was as follows:

- 2D convolutional layer with 32 filters, 3 × 3 kernels and rectified linear unit (ReLU) activation;
- 2D max pooling 2 × 2 layer;
- 2D convolutional layer with 64 filters, 3 × 3 kernels and ReLU activation;
- 2D max pooling 2 × 2 layer;
- 2D convolutional layer with 128 filters, 3 × 3 kernels and ReLU activation;
- Flatten layer (1152 values at output);
- Fully connected layer (1000 values at output);
- Fully connected layer (1000 values at output);
- Fully connected layer (10 values at the output to identify each of the 10 classes in the dataset).

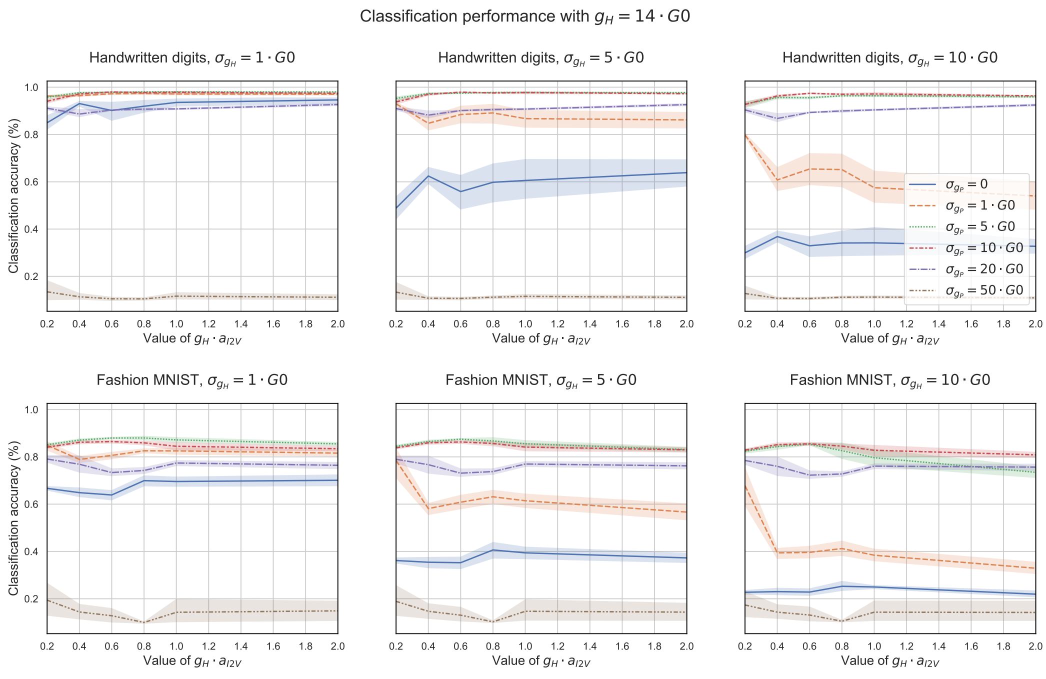

Figure 9. Performance in the classification of handwritten digits and fashion MNIST data using memristors with $g_H = 14 \cdot G_0$.

Convolutional, max pooling and flatten layers were implemented in the software. This first block transformed each image into 1152 positive values at the output of the flatten layer (1D). The second block embraced the fully connected layers and was identical to the FFNN tested so far, except for the number of input values (768 before vs. 1152 now). This second block was implemented both in the software and using the proposed memristor-based neural network. In the latter case, the outputs at the flatten layer were scaled to fit the range $[0, V_{max}]$. Note that the memristors could also be considered in the first block, but this introduces additional complexity in so far as more peripheral circuitry is required. This point is beyond the scope of the paper. In Figure 10, we reproduced the results in Figure 8 using the described CNN approach.

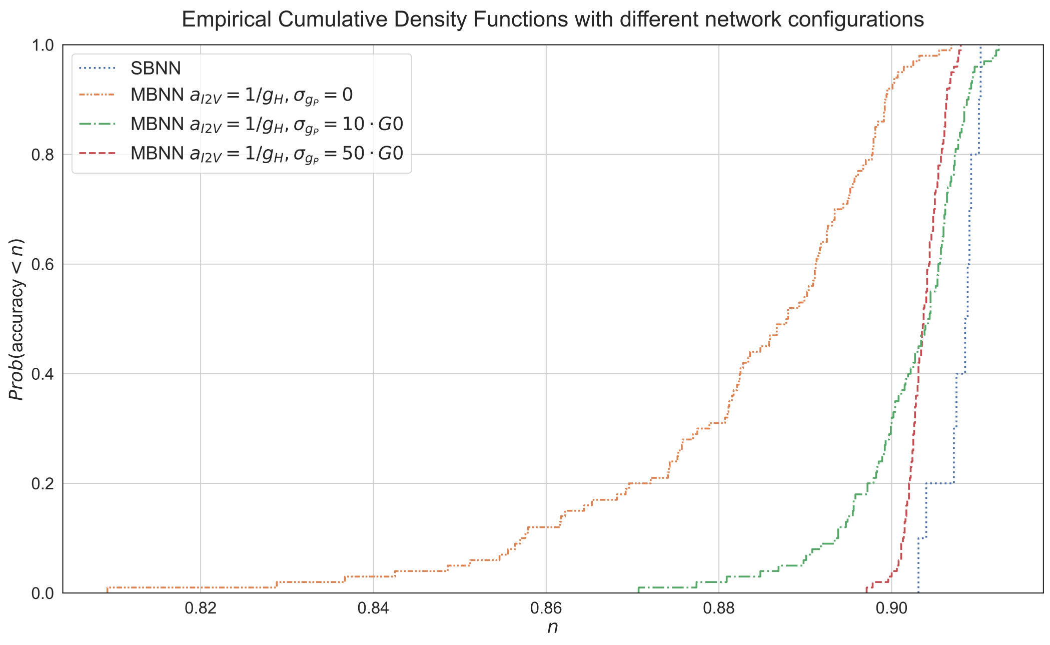

Figure 10. Performance of the SBNN and MBNN with different configurations in the classification of fashion MNIST, using the CNN approach.

The results show that the performance, in terms of prediction accuracy, increased for all the configurations tested with respect to the FFNN approach. This was due to the high-level features processed at the convolutional and max pooling layers. The second observation is that increasing the value of σ_{g_P} gave robustness to the system (i.e., the accuracy values were less spread out). This result is coherent with the results obtained so far. Finally, we observed that the MBNN with a proper configuration was close in performance to the SBNN. These preliminary results encourage us to explore the application of memristors to more complex neural network architectures.

5. Conclusions

In this paper, we analyzed the implementation of deep neural networks using crossbar arrays of memristors, and more specifically, we considered the case where these devices can be configured in only two different states: a low-resistance state (LRS) and a high-resistance state (HRS). The natural usage of crossbar arrays in the context of neural networks is in performing vector-matrix multiplications in an analog fashion (i.e., by adding currents), thus reducing the power consumption and computational time. Our approach aims at emulating ternary neural networks, which sets the weights in the neural network to a value in the range of $\{-1, 0, 1\}$. In order to achieve this behavior, we need to implement two crossbar arrays for each feedforward layer in the network (i.e., one to represent the positive weights and the other one to represent the negative weights). Additionally, some other adaptation issues in relation to software-based neural networks arise: (1) the currents at the output of the crossbar arrays have to be converted to voltages for the next stage, resulting in a conversion factor that can be potentially tuned to boost network performance and (2) memristor device experiment conductance fluctuations that also impinge on performance. Taking these issues into account, we designed an algorithm to train the weights in the network and later map these weights to the network, where memristors are programmed to either the LRS or the HRS.

The results show that the proposed system design and offline training method represent a real alternative to the traditional software-based (i.e., CMOS-based) neural networks. The lessons learned in this work are as follows: (1) the higher the conductance of the memristor in the LRS, the better performance we can achieve; (2) the conversion factor that maps the output currents at one layer to input voltages at the next layer can be fine-tuned, but it is not a sensitive parameter; and (3) it is very important to consider mitigation of the conductance variability, as performance is very sensitive to this. In our experiments, we achieved accuracies that were similar to the software-based counterpart, but without considering conductance variability during training, we observed large gaps in terms of classification accuracy between the worst realizations. This gap could be above 50% in the 10-class classification tasks (handwritten digits and fashion MNIST data) we tested.

Future work could consider additional hardware issues such as nonlinearity, stochasticity, varying maxima, asymmetry between increasing and decreasing responses, nonresponsive devices at low or high conductance, mixed time-varying delays or the sneak-path problem in crossbar arrays [10–14]. We also need to evaluate the performance using more complex and widely used neural network models, such as convolutional or recurrent networks. The preliminary results have been presented for a CNN here, showing the potential of memristor-based approaches.

Author Contributions: Conceptualization, A.M. and J.L.V.; methodology, A.M., J.L.V., E.D.M. and G.B.; software, E.D.M. and G.B.; validation, A.M., J.L.V. and E.M.; formal analysis, A.M. and E.M.; investigation, A.M., G.B., E.D.M. and J.L.V.; resources, A.M., J.L.V. and E.M.; data curation, E.D.M.; writing—original draft preparation, A.M. and J.L.V.; writing—review and editing, A.M., J.L.V. and E.M.; visualization, A.M. and E.D.M; supervision, A.M., J.L.V. and E.M.; project administration, A.M. and J.L.V.; funding acquisition, A.M. and E.M. All authors have read and agreed to the published version of the manuscript.

Funding: This work is supported by the Spanish Government under Project TEC2017-84321-C4-4-R co-funded with European Union ERDF funds and also by the Catalan Government under Project 2017 SGR 1670.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: The MNIST and Fashion MNIST datasets used in this work are publicly available and can be obtained from http://yann.lecun.com/exdb/mnist/ (accessed on 2 May 2022) and https://github.com/zalandoresearch/fashion-mnist (accessed on 2 May 2022), respectively.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Choi, S.; Ham, S.; Wang, G. Memristor synapses for neuromorphic computing. In *Memristors-Circuits and Applications of Memristor Devices*; IntechOpen: London, UK, 2020.
2. Thomas, A. Memristor-based neural networks. *J. Phys. Appl. Phys.* **2013**, *46*, 93001. [CrossRef]
3. Li, C.; Belkin, D.; Li, Y.; Yan, P.; Hu, M.; Ge, N.; Jiang, H.; Montgomery, E.; Lin, P.; Wang, Z.; et al. Efficient and self-adaptive in-situ learning in multilayer memristor neural networks. *Nat. Commun.* **2018**, *9*, 1–8. [CrossRef] [PubMed]
4. Miranda, E.; Suñé, J. Memristors for Neuromorphic Circuits and Artificial Intelligence Applications. *Materials* **2020**, *13*, 938. [CrossRef] [PubMed]
5. Alibart, F.; Zamanidoost, E.; Strukov, D.B. Pattern classification by memristive crossbar circuits using ex situ and in situ training. *Nat. Commun.* **2013**, *4*, 1–7. [CrossRef] [PubMed]
6. Kim, H.; Mahmoodi, M.R.; Nili, H.; Strukov, D.B. 4K-memristor analog-grade passive crossbar circuit. *Nat. Commun.* **2021**, *12*, 1–11. [CrossRef] [PubMed]
7. Huang, A.; Zhang, X.; Li, R.; Chi, Y. Memristor neural network design. In *Memristor and Memristive Neural Networks*; James, A.P., Ed.; IntechOpen: Rijeka, Croatia, 2018; Chapter 12. [CrossRef]

8. Yuan, G.; Ma, X.; Ding, C.; Lin, S.; Zhang, T.; Jalali, Z.S.; Zhao, Y.; Li, J.; Soundarajan, S.; Wang, Y. An Ultra-Efficient Memristor-Based DNN Framework with Structured Weight Pruning and Quantization Using ADMM. In Proceedings of the 2019 IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED), Lausanne, Switzerland, 29–31 July 2019; pp. 1–6. [CrossRef]
9. Gao, B.; Zhou, Y.; Zhang, Q.; Zhang, S.; Yao, P.; Xi, Y.; Liu, Q.; Zhao, M.; Zhang, W.; Liu, Z.; et al. Memristor-based analogue computing for brain-inspired sound localization with in situ training. *Nat. Commun.* **2022**, *13*, 2026. [CrossRef] [PubMed]
10. Fouda, M.E.; Lee, S.; Lee, J.; Eltawil, A.; Kurdahi, F. Mask Technique for Fast and Efficient Training of Binary Resistive Crossbar Arrays. *IEEE Trans. Nanotechnol.* **2019**, *18*, 704–716. [CrossRef]
11. Burr, G.W.; Shelby, R.M.; Sidler, S.; Di Nolfo, C.; Jang, J.; Boybat, I.; Shenoy, R.S.; Narayanan, P.; Virwani, K.; Giacometti, E.U.; et al. Experimental demonstration and tolerancing of a large-scale neural network (165,000 synapses) using phase-change memory as the synaptic weight element. *IEEE Trans. Electron Devices* **2015**, *62*, 3498–3507. [CrossRef]
12. Pedro, M.; Martin-Martinez, J.; Rodriguez, R.; Gonzalez, M.; Campabadal, F.; Nafria, M. A flexible characterization methodology of RRAM: Application to the modeling of the conductivity changes as synaptic weight updates. *Solid-State Electron.* **2019**, *159*, 57–62. [CrossRef]
13. Veksler, D.; Bersuker, G.; Vandelli, L.; Padovani, A.; Larcher, L.; Muraviev, A.; Chakrabarti, B.; Vogel, E.; Gilmer, D.C.; Kirsch, P.D. Random telegraph noise (RTN) in scaled RRAM devices. In Proceedings of the 2013 IEEE International Reliability Physics Symposium (IRPS), Monterey, CA, USA, 14–18 April 2013. [CrossRef]
14. Vadivel, R.; Ali, M.S.; Joo, Y.H. Robust H-infinity performance for discrete time T-S fuzzy switched memristive stochasticneural networks with mixed time-varying delays. *J. Exp. Theor. Artif. Intell.* **2021**, *33*, 79–107. [CrossRef]
15. Zhang, C.; Li, P.; Sun, G.; Guan, Y.; Xiao, B.; Cong, J. Optimizing FPGA-based accelerator design for deep convolutional neural networks. In Proceedings of the 2015 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, Monterey, CA, USA, 22–24 February 2015; Association for Computing Machinery: New York, NY, USA, 2015; pp. 161–170. [CrossRef]
16. Simons, T.; Lee, D.J. A review of binarized neural networks. *Electronics* **2019**, *8*, 661. [CrossRef]
17. Li, F.; Zhang, B.; Liu, B. Ternary weight networks. *arXiv* **2016**, arXiv:1605.04711.
18. Kim, S.; Kim, H.D.; Choi, S.J. Impact of Synaptic Device Variations on Classification Accuracy in a Binarized Neural Network. *Sci. Rep.* **2019**, *9*, 1–7. [CrossRef] [PubMed]
19. Fouda, M.E.; Lee, S.; Lee, J.; Kim, G.H.; Kurdahi, F.; Eltawi, A.M. IR-QNN Framework: An IR Drop-Aware Offline Training of Quantized Crossbar Arrays. *IEEE Access* **2020**, *8*, 228392–228408. [CrossRef]
20. Zhao, X.; Liu, L.; Si, L.; Pan, K.; Sun, H.; Zheng, N. Adaptive Weight Mapping Strategy to Address the Parasitic Effects for ReRAM-based Neural Networks. In Proceedings of the 2021 IEEE 14th International Conference on ASIC (ASICON), Kunming, China, 26–29 October 2021; pp. 1–4.
21. Vahdat, S.; Kamal, M.; Afzali-Kusha, A.; Pedram, M. Reliability Enhancement of Inverter-Based Memristor Crossbar Neural Networks Using Mathematical Analysis of Circuit Non-Idealities. *IEEE Trans. Circuits Syst.* **2021**, *68*, 4310–4323. [CrossRef]
22. Boquet, G.; Macias, E.; Morell, A.; Serrano, J.; Miranda, E.; Vicario, J.L. Offline training for memristor-based neural networks. In Proceedings of the 28th European Signal Processing Conference (EUSIPCO2020), Amsterdam, The Netherlands, 18–22 January 2021.
23. Aguirre, F.L.; Suñé, J.; Miranda, E. SPICE Implementation of the Dynamic Memdiode Model for Bipolar Resistive Switching Devices. *Micromachines* **2022**, *13*, 330. [CrossRef] [PubMed]
24. Goodfellow, I.; Bengio, Y.; Courville, A. *Deep Learning*; The MIT Press: Cambridge, MA, USA, 2016.
25. Deng, L. The mnist database of handwritten digit images for machine learning research. *IEEE Signal Process. Mag.* **2012**, *29*, 141–142. [CrossRef]
26. Xiao, H.; Rasul, K.; Vollgraf, R. Fashion-MNIST: A Novel Image Dataset for Benchmarking Machine Learning Algorithms. *arXiv* **2017**, arXiv:cs.LG/1708.07747.
27. Abadi, M.; Agarwal, A.; Barham, P.; Brevdo, E.; Chen, Z.; Citro, C.; Corrado, G.S.; Davis, A.; Dean, J.; Devin, M.; et al. TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems. 2015. Available online: https://research.google/pubs/pub45166/ (accessed on 17 December 2021).
28. Prakash, A.; Park, J.; Song, J.; Lim, S.; Park, J.; Woo, J.; Cha, E.; Hwang, H. Multi-state resistance switching and variability analysis of HfO x based RRAM for ultra-high density memory applications. In Proceedings of the 2015 International Symposium on Next-Generation Electronics (ISNE), Taipei, Taiwan, 4–6 May 2015; pp. 1–2.