Int. Journ. of Unconventional Computing, Vol. 19, pp. 219–244
Reprints available directly from the publisher
Photocopying permitted by license only
DOI: 10.32908/ijuc.v19.120824

Design and Simulation of a Hyperdimensional Computing System with Memristive Associative Memory for Image Classification

KEVIN PIZARRO AND IOANNIS VOURKAS

Department of Electronic Engineering, Universidad Técnica Federico Santa María, 2390123, Valparaíso, Chile

Received: August 12, 2024. Accepted: August 19, 2024.

Data-intensive application tasks have always fueled research and development towards more powerful computing systems. In this context, the recently proposed framework of hyper-dimensional computing (HDC) is rapidly emerging to open new opportunities for the development of systems that perform cognitive tasks in hardware. The highly memory-centric nature of HDC was the key motivation for the in-memory computing hardware implementation approaches explored recently where memristive devices were used to locally implement logic operations. In this work, we explore using memristive devices to implement one of the fundamental modules of an HDC system, the "associative memory" (AM). We designed and simulated an HDC system in MATLAB software using a behavioral model for memristive devices and explored the performance of the HDC system in image classification tasks, using different AM implementations to enrich the representation of the image classes in the AM when they included up to 25% of noise. The simulation results also explored the impact of nonidealities of memristive devices and demonstrate the critical system design aspects to consider in such an implementation approach.

Keywords: Hyperdimensional computing, image classification, associative memory, single-layer neural network, perceptron, memristor, memristive device, modeling and simulation

*Corresponding author email: ioannis.vourkas@usm.cl

1 INTRODUCTION

Data-intensive application tasks have always fueled research and development towards more powerful computing systems that explore architectural aspects beyond conventional architectures [1–2]. Brain-inspired architectures are gaining considerable attention [3–4], especially for edge-computing approaches where devices carry out cognitive tasks on the edge using limited energy and computing resources. Moreover, advancements in deep learning (DL) have led to improved performance compared to conventional machine learning (ML) approaches in many applications [5–6]. Nevertheless, the necessary massive data movements between the processing and the storage modules considerably stress the conventional computing structures and motivate the development of novel computer architectures and methods [7–8].

In this context, the recently proposed framework of hyper-dimensional computing (HDC) [9] is rapidly emerging as an attractive alternative, broadening the opportunities for the development of systems that perform cognitive tasks in hardware [10–11]. HDC is based on the properties of hyper-dimensional spaces that could explain essential aspects of human perception and cognition. HDC encodes all data features to high-dimensional vectors (hyper-vectors) and performs efficiently the classification task using a reduced set of well-defined operations [12–13]. It provides competitive accuracy on prediction tasks with a much smaller model size and training time than conventional ML. Therefore, it has shown promise in many applications, including data classification, which is the focus of this work. However, HDC comes with considerable memory requirements since every specific data should be stored in a very long vector using thousands of bits, which has motivated recent exploration of implementation alternatives [14–15].

Moreover, the need for energy efficiency has motivated the exploration of alternative device technologies, such as resistive-switching (memristive) devices, which could store information in nonvolatile form using resistance values [16–17]. In fact, the main operation in the HDC domain involves manipulating large data patterns in memory. This highly memory-centric nature of HDC was the key motivation for the in-memory computing hardware implementation proposed in [18] where memristive devices were used to locally implement logic operations [19]–20].

Likewise, in this work, we explore using memristive devices to implement one of the fundamental modules of an HDC system, the “associative memory” (AM). We base our proposal on previous work in [21], where single-layer memristive neural networks were used to implement AM structures. We designed and simulated an HDC system in MATLAB software using a behavioral model for memristive devices [22], also considered in [23] for developing digital emulators of memristive devices in hardware.

Here, we explore the performance of the HDC system in image classification tasks, using different AM implementations to enrich the representation of the query image classes in the AM when they include up to 25% of noise. The simulation results also explore the impact of nonidealities of memristive devices and demonstrate the critical system design aspects to consider in such an implementation approach, highlighting the computing capacity of HDC and the promising features delivered by emerging device technologies, aiming toward more robust and comprehensive implementations in digital hardware.

2 BASICS OF HYPERDIMENSIONAL COMPUTING

When HDC is used for classification tasks, hyper-vectors (HVs) are selected to represent every symbol in a dataset. The symbols could be the letters of the alphabet or the pixel positions within an image. HVs are the general computing elements. In every HV, its elements are randomly generated, they are independent and identically distributed (i.i.d.). **Equation (1)** shows a D-dimensional HV where h_i stands for the element in position i within this vector.

$$HV = \langle h_1, h_2, \ldots, h_D \rangle \tag{1}$$

In the hyper-dimensional space, for instance using the dimension of D = 10000, any two arbitrary HVs are nearly orthogonal and such quasi-orthogonality of HVs enables HDC to represent and integrate information using a variety of simple operations, namely; multiplication, addition, and permutation. The multiplication (*binding*) is used to bind two HVs together, which is usually done using the bitwise XOR operation. The output HV is orthogonal to the HVs being bound. Moreover, the addition (*bundling*) is used to combine different HVs into a single HV, which is similar to each component used in the bundling. The values in each element of the final HV are binarized using the majority sum operation (i.e., component-wise majority). **Equation (2)** and **Equation (3)** show the definition of the abovementioned fundamental operations over HVs. The computing output of both *bundling* and *binding* maintains the property of HVs. Finally, pseudo-random permutation (*shifting*) is also applied, such as a circular shift, which aims at shuffling of the HV contents.

$$bundling\left(HV_i, HV_j\right) = \langle h_{i1} + h_{j1}, h_{i2} + h_{j2}, \ldots, h_{iD} + h_{jD} \rangle \tag{2}$$

$$binding\left(HV_i, HV_j\right) = \langle h_{i1} * h_{j1}, h_{i2} * h_{j2}, \ldots, h_{iD} * h_{jD} \rangle \tag{3}$$

Three main processes are required to establish an HDC model, named the *encoding* process, the *training* process, and the *inference* process. A fundamental component of an HDC model is the so called "Item Memory" (IM), created as follows: Every element in the input set of signals is randomly assigned a HV and saved in the IM. For instance, each pixel position in an input image is assigned a unique seed HV. From an image with n pixels, n quasi-orthogonal seed HVs are randomly generated, which are stored in the IM and are kept during the training and inference phases of the classification.

Encoding is the process to represent a data sample in the hyper-dimensional domain. Using the case of image again as an example, for the encoding of each pixel, HDC applies the *binding* operation to the corresponding seed HV with the HV corresponding to its value, as shown in **Equation (4)**, to finally produce the HV of pixel position x:

$$HV_{px} = binding\left(HV_{seed,x}, HV_{value,x}\right) \tag{4}$$

Next, the HDC encodes the entire image into a single image HV by applying *bundling* to all pixel HVs. During the *training* process, an HDC model is built over all the training samples to produce a prototype HV representing the entire class of category. For an m-class classification task, HDC first encodes each training sample image using the above-mentioned process. Eventually the system combines all encoded HVs in the training stage to form one HD vector representing each class. This information is stored in another fundamental module called "Associative Memory" (AM), whose essential function is to compare the incoming encoded query HV with the stored class HVs and return the closest class HV using appropriate similarity metrics. Both IM and AM represent HVs that are stored permanently in the memory.

Figure 1 shows an example of what an IM and an AM would look like, in this case representing digit patterns in 19×19-pixel images, assuming ten distinguishable image classes corresponding to the ten digits. It can be noted that each of the image pixels creates a HV within the IM. On the other hand,

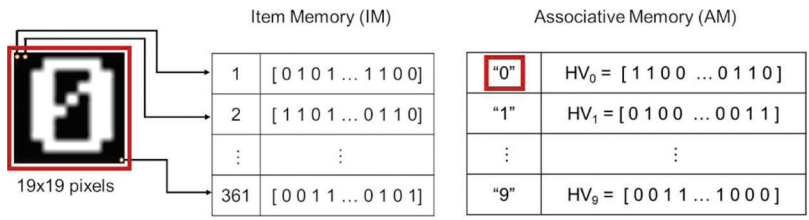

FIGURE 1
Example of an Item Memory and an Associative Memory created for ten image classes, representing the ten digits, concerning a dataset of images with 19×19 pixels

each class is associated with its corresponding HV in the AM. Finally, the *inference* process is used to determine the class of a sample input image. HDC first encodes the query image into a query HV following the same encoding strategy, which will be compared with all the class HVs to find the predicted class which demonstrates maximum similarity. To reveal the relationship between any two HVs, HDC utilizes the *Hamming distance* metric to measure their similarity. This comprises two phases, first a bitwise XOR operation followed by counting the resulting ones. If the *Hamming distance* between two HVs is close to 0, it means that the two HVs are highly correlated/similar. Eventually, the index of the class HV with the lowest *Hamming distance* indicates the prediction result.

3 DETAILS OF THE SIMULATION SETUP AND SYSTEM DESIGN METHODOLOGY

In this work we study the problem of binary image classification using a developed dataset with 19 × 19-pixel images, presenting the ten digits but with different levels of noise. There are a total of 6260 images including the 10 original images with 0% noise. It is important to define that noise is introduced with the random selection of a percentage of pixels in the original image to perform a flip on their value. An example of the type of images found in our dataset is shown in **Figure 2**. The system design and simulation was performed in MATLAB software (ver. R2024a). For the image classification system, the fundamental operations that make up a HDC system are first implemented. The IM is generated, whose main function is to act as a template to map the input information to the hyper-dimensional domain, yielding as output the image characteristics represented by the HVs. The number of

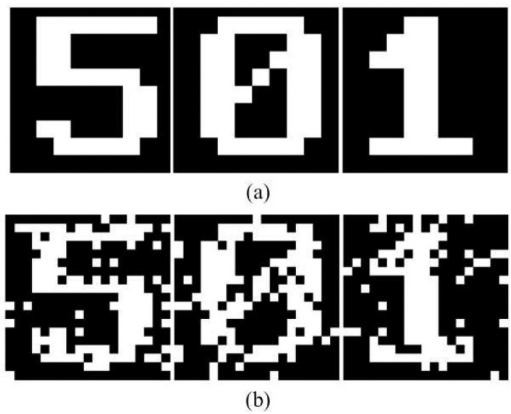

FIGURE 2
Samples of the 19 × 19 images of the dataset used in this work: (a) without noise and (b) with 15% of noise

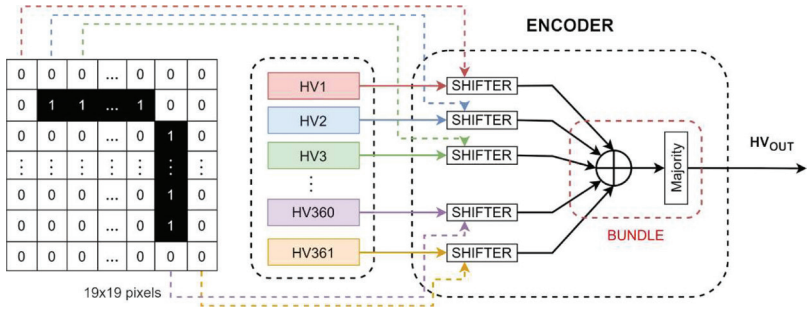

FIGURE 3
Block diagram exemplifying the operation of the encoding module within an HDC system

HVs stored in the IM is equivalent to the size of the image. It is important to note that only one IM should be created and kept fixed during system operation/simulation. The size of the IM depends on the size of the image (number of pixels) and the number of dimensions of the HVs, where we have considered $D = 1000$. For images with p pixels and HVs with D dimensions, the size of the IM will be $p \times D$ bits. For a 19×19-pixel image, it is $p = 361$. Likewise, the AM module is generated through the training process with samples of all the m different classes of data. Consequently, the AM will be formed by m HVs and its total memory size is $m \times D$ bits.

The IM is implemented using a MATLAB dictionary with 361 entries, one for each pixel of the 19×19 images, and 361 random HVs are created that are linked to each of the dictionary keys. In an analogous manner, the AM is generated for the 10 classes, which is subsequently trained and finally used for the inference phase. Finally, a similarity search must be performed within the AM. **Figure 3** summarizes the operation of the encoder. The permutation is implemented through shifters that perform a shift only if the corresponding pixel has a value of 1, otherwise it remains as it is in the IM. Then, the bundling operation is performed. For the 361 HVs, all the bits in the first position of each HV must be added and if the sum is greater than 50% of 361, then the output value in that bit is 1. This is repeated with all the bit positions of the HVs until a single output HV (HV_{OUT}) is formed.

We evaluated the performance of a HDC model implemented in software with $D = 1000$ dimensions. For the creation of the AM, the system was trained with a noise-free reference image for each class, while for the inference, the query images were randomly selected from the dataset and could include noise. The simulation results are summarized in **Figure 4** where we note that with up to 12% noise in the query images, the classification accuracy remains at 100%. As the noise increases up to 25%, the classification accuracy decreases. However, the minimum achieved accuracy is 96%, which shows that the software implementation of the HDC model operates correctly and will serve as a reference for the performance tests that follow in the rest of this work.

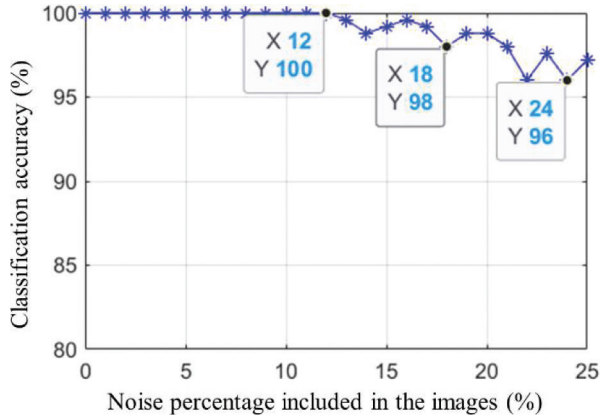

FIGURE 4
Simulation results showing the classification accuracy percentage of the HDC system in software for different levels of noise incorporated in the 19×19-pixel query images randomly chosen from the dataset. The values of specific datapoints are highlighted in the curve for readability reasons

To define the number of dimensions to be used for the HVs, we evaluated the performance of the HDC system by varying the number of dimensions between 500 and 10000, considering a fixed noise percentage in all the query images. Our analysis showed that for $D \geq 1000$, the maximum classification accuracy was always achieved. Such results justify the use of $D = 1000$ in our simulations, to limit the demand on memory resources to the minimum acceptable level that guarantees the successful operation of the system.

In the following sections we explain how different topologies of interest were assumed to build the associative memory module, particularly exploiting the technology of memristive devices. More specifically, our software implementation uses the concept of digital emulation of memristive devices according to [23], where it was shown that there is an equivalence between a behavioral mathematical model of memristors published in [22] and its potential implementation/execution in digital hardware. For each different topology to be evaluated, the performance is compared and analyzed based on the metrics of interest, which include the accuracy of the classification result, the tolerance to the variability that memristive devices may present, and also the ability to correctly classify the input query images when they include up to 25% noise.

4 DESIGN OF A MEMRISTIVE ASSOCIATIVE MEMORY FOR HDC MODELS

Here we present the design of an associative memory (AM) module based on memristive devices (or memristors [24]), which are two-terminal bipolar resistive-switching devices whose resistance can be modified by the applica-

tion of voltage pulses that generally exceed certain threshold values. We shortly present a mathematical model of a voltage-controlled memristive device proposed in [22], which was also used in [23] to achieve the digital emulation of such devices. Normally, models of such devices use at least two equations. One of them is shown in **Equation 5** which is the state-dependent Ohm's law. $i(t)$ is the current through the device and $v(t)$ the voltage across its terminals, whereas R is the device resistance (memristance), which here represents the only state variable. The second necessary equation is shown in **Equation 6** and reflects the rate of change of the state variable R, whose values are bounded by the parameters R_{ON} and R_{OFF}. Moreover, v_T is the threshold voltage that determines when the applied voltage is high enough to induce a change in the value of R. The process of decreasing the resistance is known as SET, while the opposite process is known as RESET. The constants α and β have a direct effect on the rate of change of R. Finally, $\theta(\cdot)$ is the step function.

$$i(t) = R^{-1} \cdot v(t) \tag{5}$$

$$\dot{R} = \beta \cdot v + \frac{1}{2} \cdot (\alpha - \beta) \cdot \left(|v + v_T| - |v - v_T| \right) \cdot \theta(R - R_{ON}) \cdot \theta(R_{OFF} - R) \tag{6}$$

Asymmetric thresholds can be used for the SET and RESET processes, i.e. have $v_{T,SET}$ different from $|v_{T,RESET}|$. Moreover, it is possible to define different variables β for the SET and RESET processes, and thus control the rate of change independently for each process via β_{SET} and β_{RESET}. Variability could also be added to the parameters to simulate non-idealities of physical memristive devices.

Figure 5 shows the general form of **Equation 6**, where it is important to note that there are three zones derived from the expansion of the equation itself. Zone 1 corresponds to the case when the voltage on the memristive device does not exceed any of the switching thresholds, so only parameter α will influence the switching rate if $\alpha \neq 0$. Zone 2 (RESET zone) corresponds to the case when the applied voltage exceeds the negative threshold, whereas zone 3 (SET zone) corresponds to the case when the applied voltage exceeds the positive threshold value.

To corroborate the operation of the model according to the abovementioned equations in MATLAB, a memristive device was simulated with the following values for its parameters: $\alpha = -0.1\text{G}\Omega/(\text{V}\cdot\text{s})$, $\beta_{SET} = -3\text{G}\Omega/(\text{V}\cdot\text{s})$, $\beta_{RESET} = -1\text{G}\Omega/(\text{V}\cdot\text{s})$, $v_{T,SET} = 1.5\text{V}$, $v_{T,RESET} = -0.5\text{V}$, $R_{OFF} = 10\text{K}\Omega$, $R_{ON} = 1\text{K}\Omega$, with initial state $R_{INIT} = 5\text{K}\Omega$. The device was subjected to a train of voltage pulses 10ns-wide. Different amplitude values were used to cover all cases, being lower or higher than the threshold values. The positive pulses were selected with 1V and 2V amplitude, whereas the negative pulses had −0.25V and −1.5V amplitude, respectively. In MATLAB, simulation advances

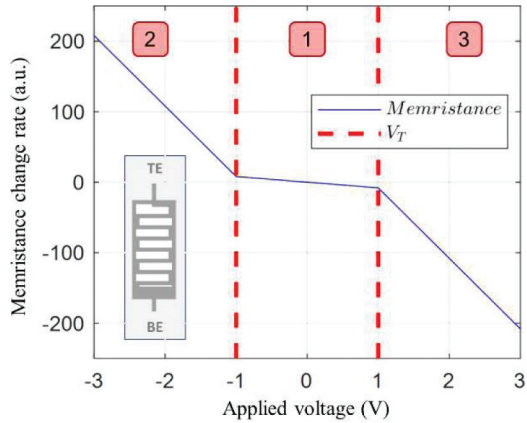

FIGURE 5
Example of the possible form of **Equation 6** which describes the memristance change rate as a function of the applied voltage. The vertical dashed lines are a guide to the eye denoting possible values for the SET and RESET thresholds. The inset shows the symbol of a memristive device. Zones 2 and 3 correspond to the RESET and SET, respectively

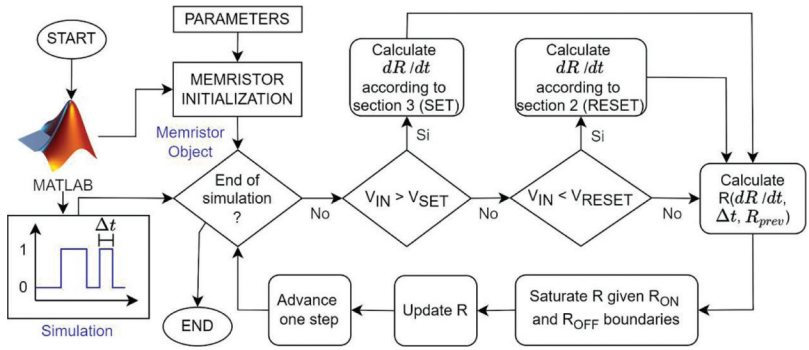

FIGURE 6
Flow chat showing the steps involved in the digital simulation of memristive devices in MATLAB

in user-defined timesteps Δt. **Figure 6** shows the flowchart for the digital emulation of a memristor in a MATLAB environment, according to guidelines in [23] where the authors implemented the same mathematical model postulated in [22] within a Field Programmable Gate Array (FPGA) and its execution advanced in discrete time steps, defined as multiples of the system clock period. In our case, using a high-level modeling language such as MATLAB, we seek an approximation equivalent to such implementation.

The simulation results are presented in **Figure 7**. First, when having an input voltage of 0V it can be noticed that the memristance does not change. Then, when applying a positive pulse of 1V, which is lower than $v_{\mathrm{T,SET}}$, a decrease of

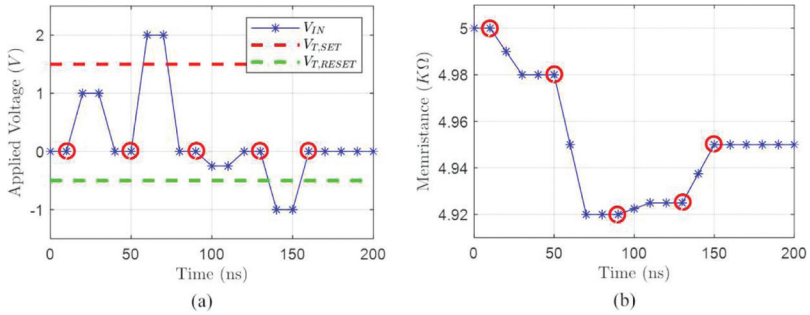

FIGURE 7
Simulation results for a memristive device submitted to a voltage pulse train. (a) the applied voltage. (b) the evolution of the memristance over time. Several data points are highlighted as a guide to the eye to improve readability

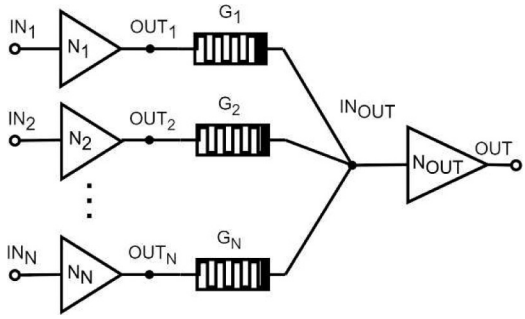

FIGURE 8
Schematic diagram of a perceptron circuit with N input neurons and one single output neuron. Synaptic connections between neurons are represented by the conductance of the memristive devices G_i.

20Ω in the memristance can be noticed. When applying positive pulses of 2V a much more drastic decrease of the memristance can be noticed (60Ω), which is because the parameter β_{SET} affects the rate of change. Likewise, similar changes can be seen for negative pulses whose values fall below or above the $v_{T,RESET}$ threshold. Finally, by keeping the applied voltage at 0V, the device maintains its last memristance value, demonstrating the desired nonvolatility.

The implementation of associative memory (AM) with memristive devices is one of the fundamental concepts within this work. **Figure 8** shows, in general terms, the topology of an N-input single layer neural network, known as "perceptron", which implements a linear classifier used in supervised learning tasks. Perceptron has two fundamental elements. The first is the synaptic connections, represented directly by the memristive devices with their conductance G_i (with i from 1 to N), which defines the weight of the connection between two neurons. The second element is the neuron N_i represented symbolically with a triangle in the schematics shown in this work.

Here we base our work on [21] where it was demonstrated that a perceptron can implement an associative memory (AM). In such topology, a strong connection corresponds to a memristive device in low resistance (low resistive state – LRS, or R_{ON}), whereas a weak connection corresponds to a high resistance (high resistive state – HRS, or R_{OFF}). Using a perceptron, having a strong connection only with the input where a reference signal is applied, it is possible to achieve the emergence of other strong connections during learning when different input signals are applied simultaneously with the reference signal, thus achieving the "association" of the different types of signals.

Moreover, according to [21], the operation of the simulated neurons has three fundamental states: the IDLE state, the EXCITED state, and the REFRACTORY state. The general behavior of the simulated neuron can be described using a state diagram as in **Figure 9**. Within the configuration parameters in the implementation of the neuron module we assume the resting time (IDLE_TIME) and the synaptic period (SYNAPSE_TIME). The IDLE_TIME indicates how long the neuron will be in the refractory period. Depending on the state in which the neuron is and on the input stimulus, an action potential (output voltage) could be triggered. However, when in the refractory period, even if there is a strong stimulus at the input, the neuron will not trigger an action voltage pulse. On the other hand, the SYNAPSE_TIME defines how long the neuron's output will remain high.

Each neuron starts in IDLE state, and awaits a stimulus that is greater than its activation threshold voltage V_T ($V_{T,NEURON}$) to transition to the EXCITED state. While at rest, the output voltage will be 0V. In the EXCITED state, the neuron output takes a value of 1V and remains in this state during SYNAPSE_TIME. Finally, in the REFRACTION state there is also 0V at the output. However, even if there is an input voltage greater than the activation threshold, the output remains at 0V. The neuron remains in the REFRACTION state during IDLE_TIME, which here we define according to [21].

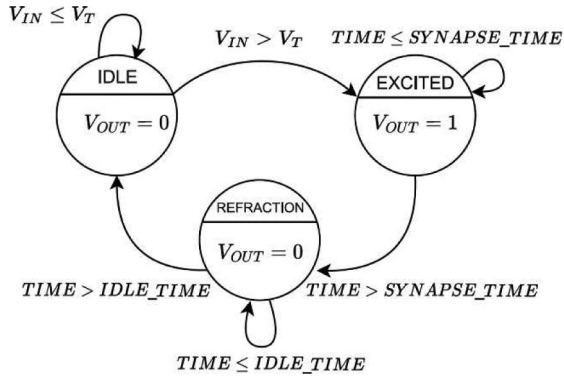

FIGURE 9
State diagram for the operation of a simulated neuron. Arrows reflect all possible state transitions, under the conditions described in text next to each arrow

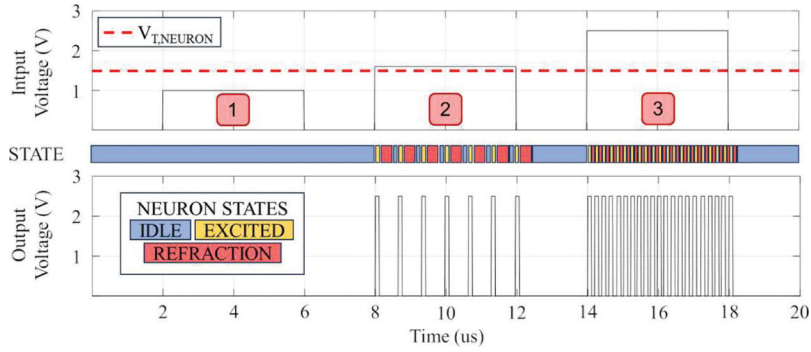

FIGURE 10
Simulation results showing the response of a neuron. (top) the applied input voltage. (bottom) the voltage at the output node of the neuron. The horizontal dashed line denotes the activation threshold of the neuron. (middle) a color map showing the evolution of the states of the neuron over time. Three scenarios are assumed: (1) when the input voltage cannot activate the neuron. (2) when the input voltage slightly exceeds the activation threshold. (3) when the input voltage is significantly higher than the activation threshold

Simulation results demonstrating the operation of the neuron model considered in this work, are presented in **Figure 10**. At first, when the input voltage is lower than the activation threshold the neuron remains in IDLE state and its output is at 0V. When the input voltage slightly exceeds the activation threshold, the neuron reacts passing through the IDLE, EXCITED and REFRACTION states continuously. In the EXCITED state, a voltage of −1V is forced on the input node of the neuron, something that will be necessary in the training/learning phase of the memristive associative memory modules evaluated in this work (not shown in the figure). Finally, when the input voltage is even higher than the activation threshold, the time that the neuron remains in the REFRACTION state is much shorter, thus responding at a much higher frequency.

5 A HYPERDIMENSIONAL COMPUTING SYSTEM FOR IMAGE CLASSIFICATION

For the design and simulation of an image classifier based on HDC, the flow of operations shown in **Figure 11** was considered. The associative memory (AM) module certainly is a key point to consider while exploring implementation alternatives that could improve or enrich the system´s performance.

In this work we seek to take advantage of the nature of memristive devices to propose an alternative implementation of the AM module within an HDC system. To better understand the steps to follow in the simulation of the system and the different phases of operation, the process flow in **Figure 12** is presented. The simulation starts by reading the dataset of images needed in

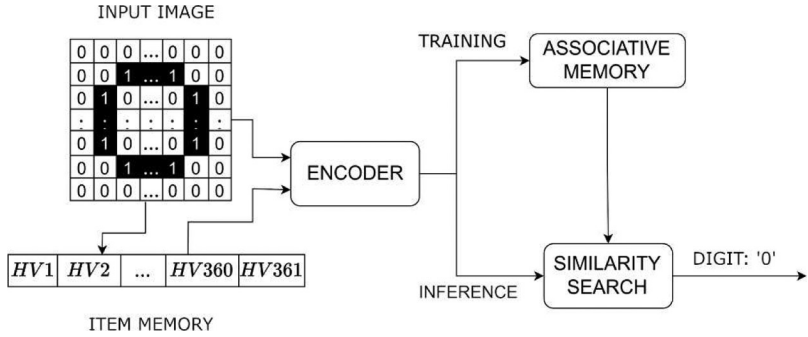

FIGURE 11
Diagram describing the overall flow of operations for image classification/inference with an HDC model.

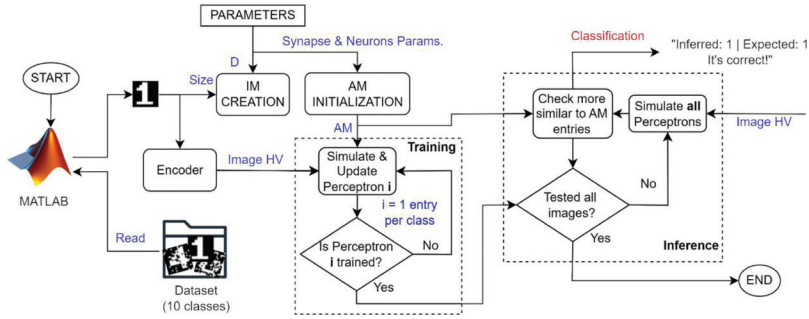

FIGURE 12
Flow chart showing the steps involved in the simulation of an HDC system for image classification in MATLAB

posterior stages. Given the defined parameters, both for the modules of the HDC system and for the memristive devices, the item memory (IM) is created and the AM is initialized with the instantiation of the perceptrons.

The training phase comes next, which aims to induce learning in each one of the perceptrons denoted as P_i (with i from 0 to 9), one for each existing class in the image dataset. It is important to note that the duration of the training for each perceptron depends on the number of dimensions of the HVs. The content of the HVs is processed serially and for each element (bit) of the HV a voltage pulse is applied to the perceptron inputs. After the perceptrons have been trained, the inference/classification phase begins where all the perceptrons must be used for every query image to be classified. This task could eventually exploit the parallelism in dedicated hardware implementations for fast execution, which however is out of the scope of this work. The two-input perceptron (HDC P2) is used as the base topology, whose performance we study below.

In the training phase, one of the perceptron inputs (IN_{REF}) receives the HV representing a noise-free reference image, whereas the other input (IN_{TRAIN})

receives the HV representing a randomly chosen image of the same class. In the initial configuration of the system, the synaptic connection with the IN_{REF} input is strong, whereas the synaptic connection with the IN_{TRAIN} input is weak (R_{TRAIN} in HRS), and the objective of this process is precisely to strengthen the weight of this connection by bringing the R_{TRAIN} resistance towards its R_{ON} limit value. At any given time, both the applied input signals will be contributing to the calculation of the voltage at the common node of the synaptic connections, which is the input node of the output neuron. However, the connection with the IN_{REF} input is stronger and therefore this signal has a greater contribution in the activation of the output neuron. The process can be better understood by observing **Figure 13** which shows a case where the same reference image is applied to both inputs of the perceptron during the training/learning phase.

When the output neuron is activated, in the next timestep, a voltage of −1V is forced on its input node. For this reason, if a 1V pulse is applied to the IN_{TRAIN} input, then the voltage drop across the memristor with resistance R_{TRAIN} becomes equal to 2V and exceeds its SET threshold, causing its resistance to decrease. To achieve the behavior described above, the parameter values of the neurons and the memristive devices must be appropriately chosen. In this case, we defined $v_{T,SET} = 1.5V$ and $v_{T,RESET} = -0.5V$, and the neurons were configured with an activation threshold $V_{T,NEURON} = 0.5V$. The output voltage of the neurons takes values of 0V or 1V to be consistent with the content of the binary HVs. We also define $R_{ON} = 100\Omega$ and $R_{OFF} = 10K\Omega$, a range that has often been observed in commercially available devices [20], [25], as it also happens with the case of asymmetric thresholds where $v_{T,SET}$ is here selected greater than $|v_{T,RESET}|$. Finally, the rate of change of the memristance must be defined through the parameters α, β_{SET} and β_{RESET}. Here we chose the rate of change to be symmetric for SET and RESET, unless stated differently, with $\beta_{SET} = \beta_{RESET}$.

In the classification stage, the IN_{TRAIN} input now becomes the IN_{QUERY} input where the applied HV represents the image to be classified. At the same time, the HV of an empty image, i.e. an image whose pixels all have zero value, is applied to the IN_{REF} input. This situation is presented in **Figure 14**. Under these conditions, and assuming a successful previous training, both

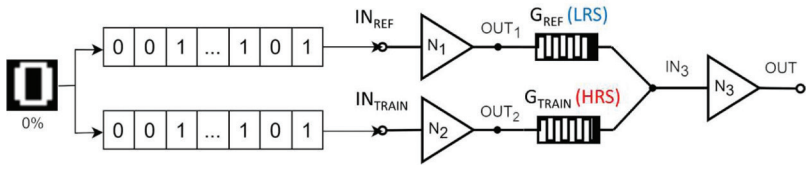

FIGURE 13
Diagram exemplifying the training/learning phase for a perceptron with two inputs. The connection where the reference signal is applied is strong (LRS) whereas the connection with the other neuron is initially weak (HRS)

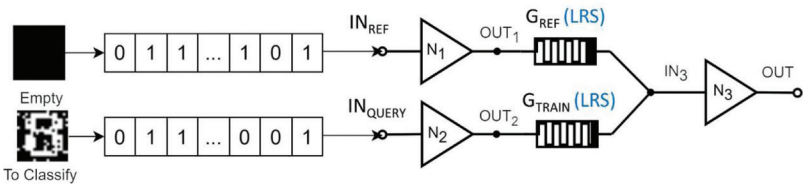

FIGURE 14
Diagram exemplifying the classification phase for a perceptron with two inputs. The connection where the reference signal is applied is by default strong (LRS) but the same occurs for the connection with the query input after successful learning.

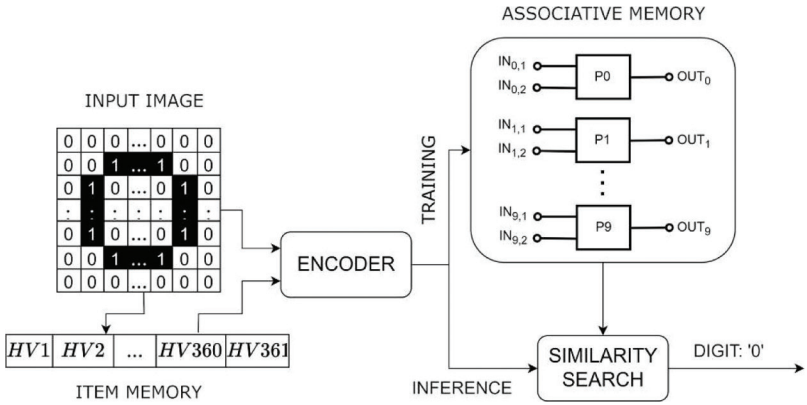

FIGURE 15
Diagram describing the overall flow of operations for image classification/inference with an HDC model, where the design of the memristive associative memory module is particularly highlighted, based on 2-input perceptrons

synaptic connections are strong with the resistance of the memristive devices in LRS at a value close to R_{ON}, and therefore both have the same weight in the contribution to the voltage at the input node of the output neuron. In this way, the activation of the output neuron will depend on the pulses applied at the IN_{QUERY} input.

With the introduction of a memristive AM module, the composition of the designed HDC system can be redefined as presented in **Figure 15**, where the AM module is shown to include a total of 10 perceptrons, one for each class present in the image dataset. However, in order to combine information from several different images and achieve a better representation of each class in the AM of the HDC system, an alternative strategy explored in this work consists in using perceptrons with more than 2 inputs.

More specifically, the possibility of having 4 inputs was evaluated, whose synaptic connections were trained based on images with different levels of noise in order to broaden the grade of representation of each class. In particular, the noise levels used were 0% (equivalent to a clean image), 5%, 10% and 15%.

Figure 16 shows the diagram of the 4-input perceptron using memristive devices, where the training inputs correspond to images with different percentage of noise. For the training phase, initially the synaptic connection of the IN_{REF} input is always strong (LRS), whereas all the other connections are weak (HRS). As for the neurons, all the parameters determined previously are maintained. It cannot be overlooked that those input neurons whose HVs correspond to images with lower levels of noise are more likely to be stimulated simultaneously with the reference input (or noise-free input), and therefore get the corresponding memristive devices toward LRS values closer to the R_{ON} limit. We generally expect that the resistance of the memristive devices at the end of the training period, would be of the form $R_1 \leq R_2 \leq R_3 \leq R_4$, where R_1 is R_{REF} and R_4 is the connection with the input where the applied signals represent the noisiest images (15%).

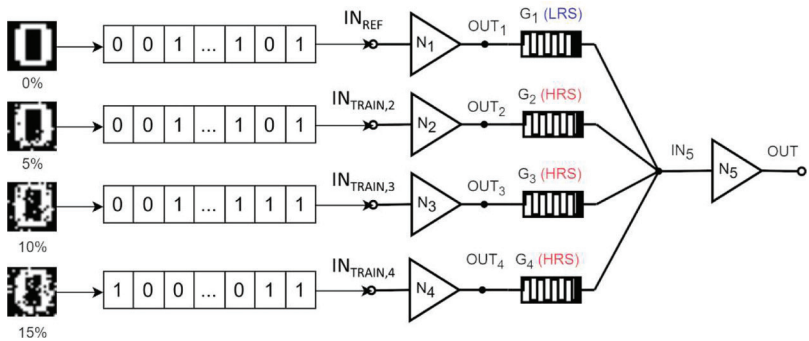

FIGURE 16
Diagram exemplifying the training/learning phase for a perceptron with four inputs. The connection where the reference signal is applied is strong (LRS) whereas the connection with the other neurons is initially weak (HRS)

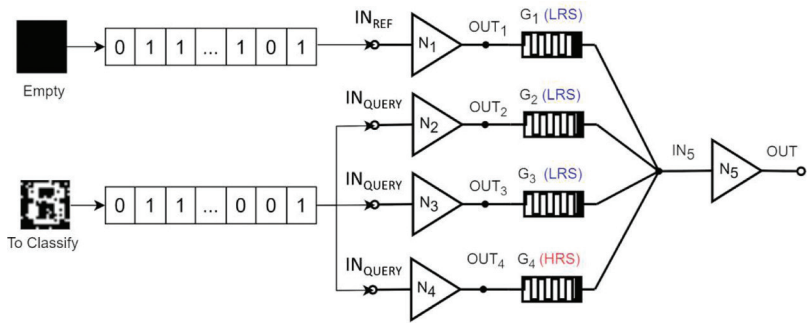

FIGURE 17
Diagram exemplifying the classification phase for a perceptron with four inputs. The connection where the reference signal is applied is by default strong (LRS) but the state of the rest of the synaptic connections depends on the previous learning process. As an example, here only 2 of the 3 training memristive devices are shown in LRS

For the inference phase in such extended perceptron topology, the process is very similar to the case with the two input perceptrons. As shown in **Figure 17**, the reference input image represents an empty image, whereas the other three inputs receive the same HV that represents the image to be classified. Thus, the strongest connections are expected to predominate in the activation of the output neuron.

Following the same strategy, it is possible to use perceptrons with as many inputs as necessary. Considering that the image dataset includes images with up to 25% of noise, in our analysis we included the case of an AM based on perceptrons with 6 inputs where HVs of images with 0%, 5%, 10%, 15%, 20% and 25% noise were applied. The operation in the training and inference phases is the same as that described previously for the case of 4 inputs.

The last process of the HDC system to be described is the similarity search, which is necessary in the classification stage. **Figure 18** shows a diagram that explains the operations that make up the similarity search and the way in which the module is connected to the AM. The query image information is passed through the encoder to obtain HV_{OUT}. Then, HV_{OUT} is passed as input to each perceptron of the AM for inference, thus obtaining the HVs denoted as OUT_X (with x from 0 to 9) which must then be compared with the HVs that were obtained from the training phase ($TRAINED_X$). To this end, a bitwise XOR operation is performed between $TRAINED_x$ and OUT_x with the result reflected in HV_COMP_x. This process is repeated for each pair of $TRAINED_X$ - OUT_X. Subsequently, all the bits of each HV_COMP_X are added to obtain the number of bits in which the $TRAINED_X$ - OUT_X pairs differ (sim_X). Finally, all sim_X values are compared and that with the lowest value corresponds to the inferred class.

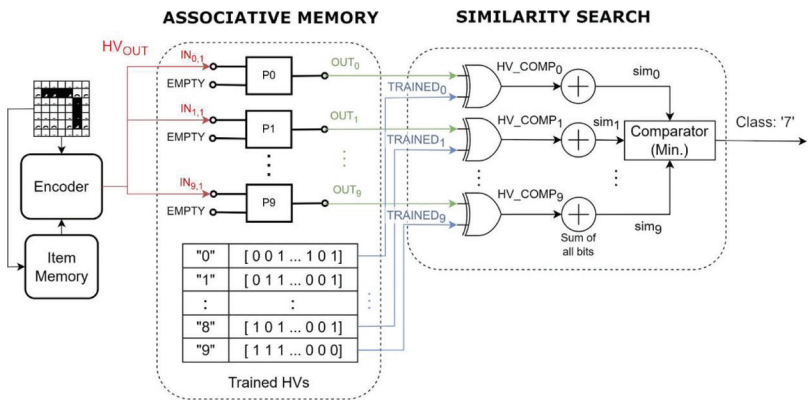

FIGURE 18
Diagram describing the overall flow of operations for similarity search between the query image and the contents of AM (trained HVs), within an HDC system used for image classification/inference. The connections between the similarity search module and the memristive AM module (here appearing with 2-input perceptrons) are highlighted

6 SIMULATION RESULTS

Through different configurations of the memristive devices, we explore the performance of the HDC system and the impact produced by the different switching rates of the devices to the effectiveness of the training/learning phase of the AM. We particularly focus on the asymmetric nature of the SET and RESET processes, related to the learning and forgetting rates that are occurring during training. Moreover, we assess the effect produced by the non-idealities of the memristive device technology, such as the variability of the SET/RESET thresholds, among others. The simulations were carried out for $D = 1000$ dimensions and 25 iterations of each experiment, using a total of 250 images with 10% noise in the inference phase. That is, 25 different images for each digit with 10% of pixels altered.

6.1 Impact of asymmetric switching rates on learning performance

We simulated the behavior of a perceptron with four inputs during the training phase. The simulation lasted 10μs, since $D = 1000$ and the applied pulse for every HV element was 10 ns wide. The memristive device that corresponds to the input where the reference signal was applied was the only one initialized in LRS. The other inputs receive the HVs that represent images with 5%, 10% and 15% noise, respectively. In **Figure 19** we observe the results for the case where the learning rate is greater than the forgetting rate. The values used were $\alpha = -10K\Omega/(V{\cdot}s)$ and $\beta_{RESET} = -5G\Omega/(V{\cdot}s)$ in the three simulated cases, but the value of β_{SET} varied with values −10, −15, and $-20G\Omega/(V{\cdot}s)$. It can be noted that the higher the value of β_{SET}, the better the training result since it is guaranteed that a greater number of memristive devices reach their LRS, thus improving the classification precision.

Similarly, the behavior of the same perceptron was evaluated when the learning rate was lower than the forgetting rate during training. In practical terms, this means that a negative voltage on the memristive devices could

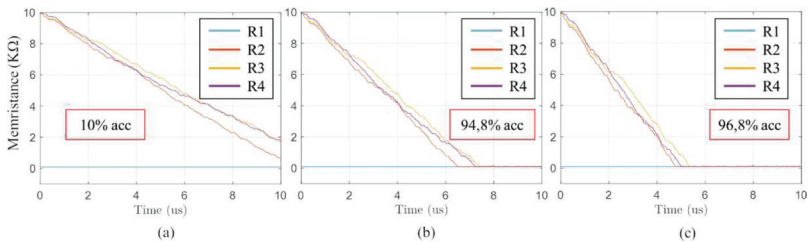

FIGURE 19
Simulation results showing the average memristance evolution over time for all the synaptic connections of a perceptron with four inputs during training, when (a) $\beta_{SET} = -10G\Omega/(V{\cdot}s)$, (b) $\beta_{SET} = -15G\Omega/(V{\cdot}s)$, and (c) $\beta_{SET} = -20G\Omega/(V{\cdot}s)$. In every case the best achieved classification accuracy for digit 0 with 10% of noise in the query images during inference, is informed in the inset.

cause a significant increase in their resistance, thus nullifying the previously acquired training result. The parameters that were kept fixed in each simulation are $\alpha = -10K\Omega/(V{\cdot}s)$ and $\beta_{SET} = -15G\Omega/(V{\cdot}s)$, whereas β_{RESET} varied taking the values −20, −25, and $-35G\Omega/(V{\cdot}s)$. In the simulation results shown in **Figure 20** we notice that, the higher the value of β_{RESET}, the worse the training process is, since the memristive devices do not manage to completely switch to their LRS. Consequently, the classification accuracy gets significantly worse (see inset). We repeated the same process for the case of a 6-input perceptron, this time using images with up to 25% noise. For the case where the learning rate was higher than the forgetting rate during the training period, we kept the same value of β_{RESET} in the three cases while varying the value of β_{SET}. In the simulation results of **Figure 21** we observe that the higher the value of β_{SET}, the better the training result and the more effective the classification process. All in all, the same trends as observed previously in **Figure 19** are also present here for the 6-input perceptron.

Finally, for the case where the learning rate is lower than the forgetting rate during training, more dispersed curves were obtained as it can be seen in

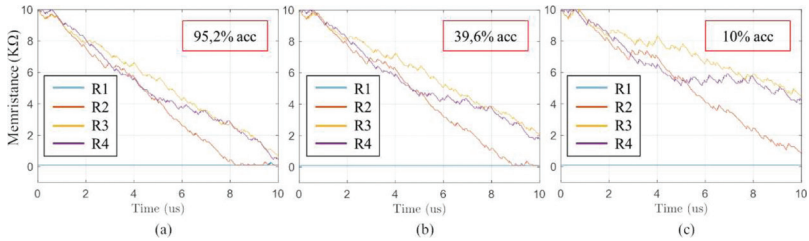

FIGURE 20
Simulation results showing the average memristance evolution over time for all the synaptic connections of a perceptron with four inputs during training, when (a) $\beta_{RESET} = -20G\Omega/(V{\cdot}s)$, (b) $\beta_{RESET} = -25G\Omega/(V{\cdot}s)$, and (c) $\beta_{RESET} = -35G\Omega/(V{\cdot}s)$. In every case the best achieved classification accuracy for digit 0 with 10% of noise in the query images during inference, is informed in the inset

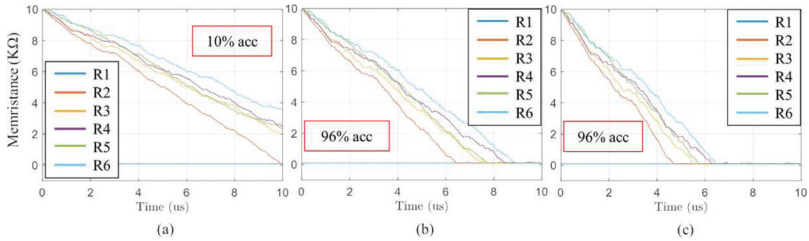

FIGURE 21
Simulation results showing the average memristance evolution over time for all the synaptic connections of a perceptron with six inputs during training, when (a) $\beta_{SET} = -10G\Omega/(V{\cdot}s)$, (b) $\beta_{SET} = -15G\Omega/(V{\cdot}s)$, and (c) $\beta_{SET} = -20G\Omega/(V{\cdot}s)$. In every case the best achieved classification accuracy for digit 0 with 10% of noise in the query images during inference, is informed in the inset

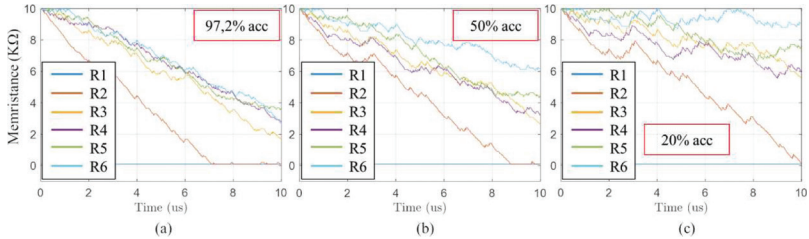

FIGURE 22
Simulation results showing the average memristance evolution over time for all the synaptic connections of a perceptron with six inputs during training, when (a) $\beta_{RESET} = -20G\Omega/(V{\cdot}s)$, (b) $\beta_{RESET} = -25G\Omega/(V{\cdot}s)$, and (c) $\beta_{RESET} = -35G\Omega/(V{\cdot}s)$. In every case the best achieved classification accuracy for digit 0 with 10% of noise in the query images during inference, is informed in the inset

the results shown in **Figure 22**, similar to the results shown previously in **Figure 20** for the case of a 4-input perceptron. The higher the value of β_{RESET}, the worse the training process is and the classification accuracy decreases substantially.

Everything considered, the most important conclusion of this exploration is that there are several configurations of the memristive devices that give positive results, i.e. where the achievable precision of classification is higher than 90%. To obtain such results, it is important to achieve a decrease in the resistance value of the memristive devices of the trained synaptic connections toward values close to the R_{ON} limit. However, our analysis showed that this is necessary to occur with at least one of the devices in the case of a 4-input perceptron, or at least two memristive devices when the perceptrons have 6 inputs.

6.2 Impact of nonidealities of memristive devices on learning performance

Real memristive devices show considerable variability in their switching behavior [26], unlike the ideal behavior considered so far in simulations. For this reason, in order to obtain more reliable simulation results, in this section we evaluate the response of the system in presence of variability in the memristive devices. One of the model parameters that can be considered to introduce variability is the threshold values for the SET and RESET processes. The parameter values used in the ideal scenario are $v_{T,SET} = 1.5V$ and $v_{T,RESET} = -0.5V$, taking into account that a voltage of 2V is applied for SET and a voltage of −1V is applied to the memristive devices for RESET. Therefore, it makes sense to consider variations for both SET and RESET thresholds that allow to observe the effect of variability without risking to nullify the SET and RESET processes, making sure that the applied voltage for SET and RESET will always be higher than the threshold even in the extreme cases of variability. In this context, we explore the response of the system in the most and the least favorable

conditions for learning, comparing them with the nominal case (the case without variability).

We simulate the behavior of a perceptron with four inputs during the training phase. Three scenarios are considered: the worst case, the nominal case, and the best case. The worst case is where incomplete training is observed, which is achieved with $v_{T,RESET} = -0.25V$ and $v_{T,SET} = 1.75V$ since the effect of the forgetting rate is maximized and the effect of the learning rate is minimized. Similarly, for the best case where early learning is achieved during the training period, the learning rate is maximized and the forgetting rate is minimized with $v_{T,RESET} = -0.75V$ and $v_{T,SET} = 1.25V$. In the results shown in **Figure 23** it can be seen that varying the threshold voltages in the model used for the memristive devices has the same effect as modifying the parameters β_{SET} and β_{RESET}, which determine the rate of variation of the resistance, and therefore, the speed with which the perceptron can be trained. This kind of exploration provides results which allow the system designer to evaluate the worst scenario for such HDC system, given an initial configuration for it, to further examine whether it is necessary to modify the training time or change the amplitude of the applied pulses during training to achieve better classification performance.

Another of the model parameters that can be considered for applying variability is the limiting resistance value reachable during SET and RESET, i.e. R_{ON} and R_{OFF}. In this way, the possibility of having devices whose high or low resistance state does not correspond to a fixed value, but instead to a variety of similar values is considered. However, for the result of the classification process, what determines the behavior of the system is the degree to which each memristive device has approached the R_{ON} resistance. Consequently, we perform simulations where we use a fixed R_{OFF} but where the R_{ON} value varies within specific margins. We define the rest of the simulation parameters as $D = 1000$, we apply 10% image noise, $\alpha = -10K\Omega/(V \cdot s)$, $\beta_{SET} = \beta_{RESET} =$

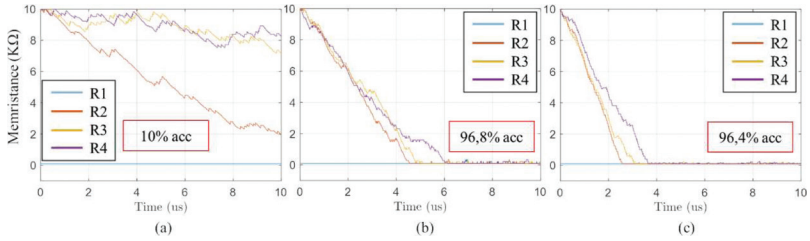

FIGURE 23
Simulation results showing the average memristance evolution over time for all the synaptic connections of a perceptron with four inputs during training, when extreme values were considered for the threshold parameter as follows: (a) $v_{T,RESET} = -0.25V$ and $v_{T,SET} = 1.75V$, (b) $v_{T,RESET} = -0.5V$ and $v_{T,SET} = 1.5V$, and (c) $v_{T,RESET} = -0.75V$ and $v_{T,SET} = 1.25V$. In every case the best achieved classification accuracy for digit 0 with 10% of noise in the query images during inference, is informed in the inset

LRS Limit Memristance Value		Average HDC Classification Accuracy		
$R_{1,ON}$ (Ω)	$R_{X,ON}$ (Ω)	HDC P2 (%)	HDC P4 (%)	HDC P6 (%)
100	250	10	69.2	94
60	140	10	94.4	94.8
85	115	10	92.8	94
100	100	82.4	94.8	94.8
115	85	94	93.6	95.2
140	60	94	93.2	95.6
250	100	94	93.2	95.6

TABLE 1
Summary of average achieved classification accuracy for different implementations of the associative memory when different values were assumed for the R_{ON} parameter of the memristive devices

$-24G\Omega/(V \cdot s)$, $v_{T,SET} = 1.5V$, $v_{T,RESET} = -0.5V$, $R_{OFF} = 10K\Omega$, and initialize the memristive devices with $R_1 = R_{ON}$, $R_X = R_{OFF}$ (with x = {2, 3, ..., 6}). We consider 2-input, 4-input, and 6-input perceptrons and vary the R_{ON} value with respect to the nominal value. R_1 refers to the memristive device of the reference input, whereas R_X refers to the memristive devices of the rest of the perceptron inputs.

Table 1 shows a summary of the simulation results. It can be noted that the impact of the theoretically worst case where it is $R_{1,ON} < R_{x,ON}$ (this forces the training memristive devices to achieve LRS values higher than that of the reference LRS value) is clearly observed for the HDC system whose AM used 2-input perceptrons (HDC P2). However, by including more inputs in the perceptron circuits, robustness to the difference between $R_{1,ON}$ and $R_{x,ON}$ is gained. In particular, the average classification accuracies were higher for the case of 6-input perceptrons (HDC P6), followed by the 4-input perceptron case (HDC P4), and finally by the implementation with HDC P2.

6.3 Impact of HV dimensions and of query image noise on classification accuracy

One of the essential tests when working with the HDC paradigm is to observe the performance of the system as the number of dimensions for each HV increases. In this direction, **Figure 24** shows the simulation results for the cases where the AM module used perceptrons with 2 (HDC P2), 4 (HDC P4), and 6 (HDC P6) inputs, compared to the original implementation of the system without considering the use of memristive associative memory (HDC). The classification of a total of 250 images with 10% noise was again pursued. It can be noted that, among the three variations under test for the AM module, the HDC P2 presents the worst results, whereas HDC P4 and HDC P6 have practically identical curves. Interestingly, both HDC P4 and HDC P6 achieve

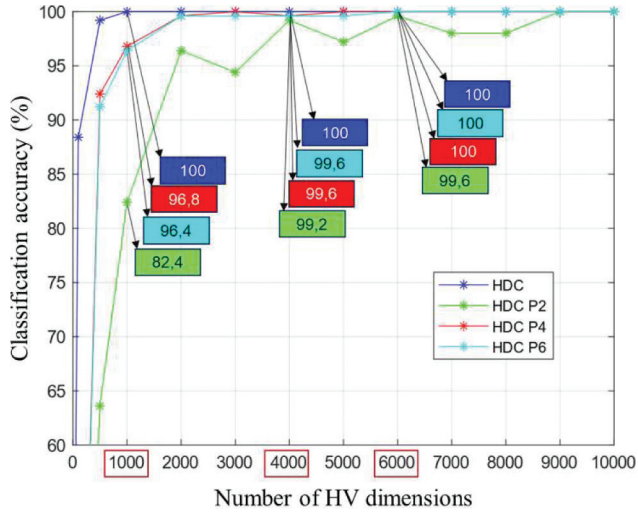

FIGURE 24
Simulation results for the average classification accuracy achieved for query images with up to 10% noise, comparing the original software implementation of HDC with the three alternatives considered for the AM module using perceptrons with 2 (HDC P2), 4 (HDC P4), and 6 (HDC P6) inputs

a classification accuracy that exceeds 99% for $D \geq 3000$. The observed difference in the performance of HDC P2 compared to the other two alternatives suggests that, when aiming to implement a robust classification system, it is necessary to consider noisy information in the training phase, thus more inputs in the memristive perceptron circuits. It is also intuitive to think that, with a larger number of dimensions used in the HVs, the need for hardware resources will increase, so there is a trade-off between the requirement for resources and the minimum desirable accuracy in the performance of the designed HDC system.

One of the most important and widely-documented characteristics of HDC is its robustness to noise. Therefore, as a final test in this study, we observed and analyzed the system´s performance depending on the noise levels present in the input images to be classified. We chose a fixed number of dimensions $D = 1000$ while varying the noise level present in the query images from 0% to 25% with a step of 1%. We performed 25 repetitions of the experiment for each noise level. The simulation results are shown in **Figure 25**. We observe that the proposed topologies that make use of perceptron-based associative memory demonstrate an earlier decrease in their performance and at a higher rate, compared to the original HDC implementation. It can be noted that HDC P2 is the implementation that shows the worst results, which can be explained by the fact that its training/learning was performed using only clean images without noise. On the other hand, the systems using HDC P4 and HDC P6 were trained with noisy images, together with a clean reference.

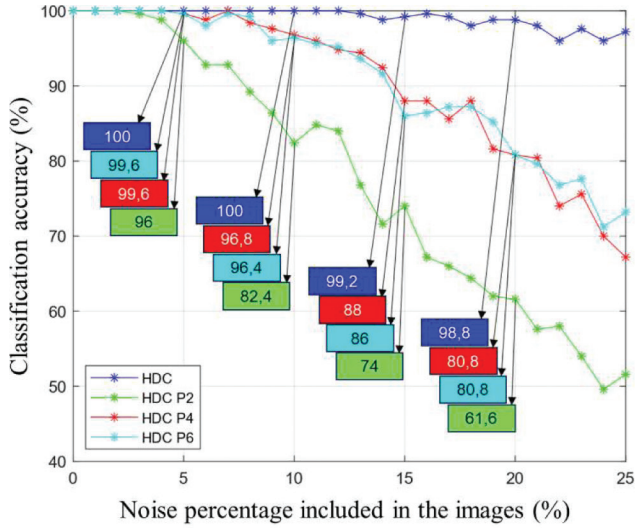

FIGURE 25
Simulation results for the average classification accuracy achieved for query images with up to 25% noise, comparing the original software implementation of HDC with the three alternatives considered for the AM module using perceptrons with 2 (HDC P2), 4 (HDC P4), and 6 (HDC P6) inputs

It should be noted that in the case of HDC P4, images with up to 15% noise were used in training, whereas for HDC P6 the noise considered in the images during training reached 25%. This explains the fact that in the graphs of **Figure 25** we notice a similar behavior between both for up to approximately 15% of image noise. From this point on, the performance of HDC P6 slightly exceeds that of HDC P4, achieving a 5% improvement in classification accuracy when the query images had the maximum amount of noise. It can also be observed that the accuracy of the original HDC implementation varies only between 95% and 100%, and that a slight decrease is observed only when the image noise exceeds 13%. In general, this HDC implementation demonstrates the best performance, which underlines the capacity of the HDC concept to be used in classification tasks.

CONCLUSIONS

Through supervised learning at circuit level, this work explored the design and implementation of associative memory modules for HDC systems based on memristive perceptron circuits. The proposed implementations fit seamlessly in the definition of an HDC system for classification tasks and performed well, compared to the original software implementation of a "pure" HDC. Nevertheless, the idea of expanding the number of inputs of the memristive percep-

trons to achieve a better representation of the image dataset classes led to better results, underlying the importance of considering this in potential implementations in hardware (HW). The simulation of memristive devices followed an approach compatible with execution in digital HW, so the proposed circuits could be considered for future HW accelerators of HDC classifiers. Still, the performance of the designed system could not exceed that of the pure software HDC implementation. However, further optimizations are possible, and a larger number of inputs in the perceptron circuits could decrease the performance gap. Moreover, such memristive associative memory, if implemented in HW, could certainly exploit parallelism of execution, and thus lead to very efficient, low-energy implementation.

ACKNOWLEDGMENTS

This work was funded by the Chilean government under grants FONDECYT Regular No. 1221747 and ANID-Basal FB0008.

REFERENCES

[1] T. N. Theis, and H.-S. P. Wong, "The end of Moore's law: A new beginning for information technology," *Comput. Sci. Eng*., vol. **19**, no. 2, pp. 41–50, 2017.

[2] A. Sebastian, *et al*., "Temporal correlation detection using computational phase-change memory," *Nat. Commun*., vol. **8**, no. 1115, 2017.

[3] T. F. Wu, *et al*., "Brain-inspired computing exploiting carbon nanotube FETs and resistive RAM: hyperdimensional computing case study," 2018 *IEEE International Solid-State Circuits Conference* (ISSCC), San Francisco, CA, USA, Feb. 11–15.

[4] D. Kleyko, and E. Osipov, "Brain-like classifier of temporal patterns," 2014 *International Conference on Computer and Information Sciences* (ICCOINS), Kuala Lumpur, Malaysia, June 3-5.

[5] V. Joshi, *et al*., "Accurate deep neural network inference using computational phase-change memory," *Nat. Commun*., vol. **11**, no. 2473, 2020.

[6] H. Alavizadeh, H. Alavizadeh, and J. Jang-Jaccard, "Deep Q-Learning Based Reinforcement Learning Approach for Network Intrusion Detection," *Computers*, vol. **11**, no. 3(41), 2022.

[7] F. Pinto, and I. Vourkas, "Robust Circuit and System Design for General-Purpose Computational Resistive Memories," *Electronics*, vol. **10**, no. 9, pp. 1074, 2021.

[8] I. Vourkas, G. Papandroulidakis, G. Ch. Sirakoulis, and A. Abusleme, "2T1M-Based Double Memristive Crossbar Architecture for In-Memory Computing," *Int. Journ. of Unconventional Computing*, vol. **12**, no. 4, pp. 265–280, 2016.

[9] P. Kanerva, "Hyperdimensional Computing: An Introduction to Computing in Distributed Representation with High-Dimensional Random Vectors," *Cogn. Comput*., vol. **1**, pp. 139–159, 2009.

[10] M. Imani, D. Kong, A. Rahimi, and T. Rosing, "VoiceHD: Hyperdimensional computing for efficient speech recognition," 2017 *IEEE International Conference on Rebooting Computing* (ICRC), Washington, DC, USA, Nov. 08–09.

[11] E. Hassan, Y. Halawani, B. Mohammad, and H. Saleh, "Hyper-Dimensional Computing Challenges and Opportunities for AI Applications," *IEEE Access*, vol. **10**, pp. 97651–97664, 2022.

[12] L. Ge, and K. K. Parhi, "Classification using hyperdimensional computing: A review," *IEEE Circuits Syst. Mag*., vol. **20**, no. 2, pp. 30–47, 2020.

[13] A. Kazemi, *et al*., "Achieving software-equivalent accuracy for hyperdimensional computing with ferroelectric-based in-memory computing," *Sci. Rep*., vol. **12**, no. 19201, 2022.

[14] S. Datta, R. Antonio, A. Ison, and J. Rabaey, "A Programmable Hyper-Dimensional Processor Architecture for Human-Centric IoT," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. **9**, no. 3, pp. 439 - 452, 2019.

[15] D. Liang, J. Shiomi, N. Miura, and H. Awano, "DistriHD: A Memory Efficient Distributed Binary Hyperdimensional Computing Architecture for Image Classification," 2022 *Asia and South Pacific Design Automation Conference* (ASP-DAC), Taipei, Taiwan, Jan. 17–20.

[16] M. Lanza, *et al*., "Memristive technologies for data storage, computation, encryption, and radio-frequency communication," *Science*, vol. **376**, no. 6597, Jun. 2022.

[17] I. Vourkas, M. Escudero, G. Ch. Sirakoulis, and A. Rubio, "Ubiquitous memristors in multi-level memory, in-memory computing, data converters, clock generation and signal transmission," in: P. Dimitrakis, I. Valov, and S. Tappertzhofen (Eds.) "*Metal oxides for non-volatile memory, materials, technology and applications*," Elsevier, 2022, pp. 445–463.

[18] G. Karunaratne, *et al*., "In-memory hyperdimensional computing," *Nature Electronics*, vol. 3, pp. 327–337, 2020.

[19] I. Vourkas, and G. Ch. Sirakoulis, "Emerging Memristor-based Logic Circuit Design Approaches: A Review," *IEEE Circuits Syst. Mag*., vol. **16**, no. 3 (3rd quarter), pp. 15–30, 2016.

[20] C. Fernandez, A. Cirera, and I. Vourkas, "Design Exploration of Threshold Logic in Memory and Experimental Implementation using Knowm Memristors," *Int. Journ. of Unconventional Computing*, vol. **18**, no. 2-3, pp. 249–267, Jun. 2023.

[21] Y. V. Pershin and M. Di Ventra, "Experimental demonstration of associative memory with memristive neural networks," *Neural Netw*., vol. 23, no. 7, pp. 881–886, 2010.

[22] Y. Pershin and M. Di Ventra "SPICE model of memristive devices with threshold," *Radioengineering*, vol. **22**, no. 2, pp. 485–489, 2013.

[23] V. Ntinas, I. Vourkas, A. Abusleme, G. Ch. Sirakoulis, and A. Rubio, "Experimental Study of Artificial Neural Networks Using a Digital Memristor Simulator", *IEEE Trans. Neural Networks and Learning Systems*, vol. **29**, no. 10, pp. 5098–5110, 2018.

[24] L. Chua, "Everything you wish to know about memristors but are afraid to ask," *Radioengineering*, vol. **24**, no. 2, pp. 319–368, 2015.

[25] K. A. Campbell, "Self-directed channel memristor for high temperature operation," *Microelectron. J*., vol. **59**, pp. 10–14, Jan. 2017.

[26] J. B. Roldán, *et al*, "Variability in Resistive Memories," *Adv. Intell. Syst*., no. 2200338, 2023.