ADVANCED INTELLIGENT SYSTEMS Open Access
www.advintellsyst.com

Automating Analogue AI Chip Design with Genetic Search

*Olga Krestinskaya, Khaled N. Salama, and Alex P. James**

Optimization of analogue neural circuit designs is one of the most challenging, complicated, time-consuming, and expensive tasks. Design automation of analogue neuromemristive chips is made difficult by the need to design chips at low cost, ease of scaling, high-energy efficiency, and small on-chip area. The rapid progress in edge AI computing applications generates high demand for developing smart sensors. The integration of high-density analogue computing AI chips as coprocessing units to sensors is gaining popularity. This article proposes a hardware–software codesign framework to speed up and automate the design of analogue neuromemristive chips. This work uses genetic algorithms with objective functions that take into account hardware nonidealities such as limited precision of devices, the device-to-device variability, and device failures. The optimized neural architectures and hyperparameters successfully map with the library of relevant neuromemristive analogue hardware blocks. The results demonstrate the advantage of proposed automation to speed up the analogue circuit design of large-scale neuromemristive networks and reduce overall design costs for AI chips.

1. Introduction

Electronic design automation in the era of AI is experiencing rapid transformations prompted by the increased demands of edge AI computing applications. The AI applications demand accelerated computing of deep neural networks and require device miniaturization to increase chip density. Low-power analogue computation using memristive systems provide increased chip density and ease of implementation in analogue and mixed-signal domains.[1] However, the manual optimization of neuromemristive systems deals with several issues of device variability and nonideal behavior that make the design process difficult. Bringing automation in the analogue circuit block selection for neural architectures can speed up the design process, reducing the overall cost of chip design and speeding up implementations.

The classical software-based approaches for hyperparameter optimization of the deep neural network are a resource-consuming and complicated task. Often there is no one right solution to such an optimization problem, and there are several optimization methods such as random search,[2] Bayesian optimization,[3–5] and evolutionary algorithms.[6,7] While these methods have proven to be effective to a variety of classification problems, they do not take into account hardware-specific issues required for AI chip implementation.[8,9] Among the evolutionary algorithms, we consider genetic algorithms[10–14] to be the most hardware friendly to implement. The random number of assignments, mutations, and crossovers can be implemented for an online learning chip if required for edge AI applications.

The successful automation of neuromemristive analogue AI chip design would require to consider hardware-specific issues and not just limit to the classification accuracy of the neural network configuration. Several nonideal behaviors of neuromemristive devices and networks[15,16] need to be considered in the design process. This includes the limited number of stable conductive states,[17] stochasticity and variation of the conductive states,[18,19] and device aging[20,21] and failures. We propose to bring the automation in circuit design by using the genetic algorithm for the selection of hyperparameters against hardware-specific objectives. Furthermore, we map the library of neuromemristive blocks to the optimal neural architecture. We show how the nonideality affects the selection of the network, and how the selected configurations can optimize the hardware design of a network for face (AR[22]), character (MNIST[23]), and fashion (Fashion-MNIST[24]) image datasets.

2. Background

2.1. Memristor and Memristor Nonidealities

Memristor device and crossbars[25–30] are a promising solution for analogue computing architectures, notably for the implementation of dot product computation in neural networks on hardware.[31–35] Nevertheless, nonideality in devices implies

O. Krestinskaya, Prof. K. N. Salama
Sensors Lab Computer, Electrical, and Mathematical Science and Engineering Division
King Abdullah University of Science and Technology (KAUST)
Thuwal, Saudi Arabia

O. Krestinskaya
Nazarbayev University
Nur-Sultan, Kazakhstan

Prof. A. P. James
Centre for Artificial General Intelligence and Neuromorphic Systems
Indian Institute of Information Technology and Management - Kerala (IIITM-K)
Technopark, Trivandrum, Kerala, India
E-mail: apj@ieee.org

The ORCID identification number(s) for the author(s) of this article can be found under https://doi.org/10.1002/aisy.202000075.

DOI: 10.1002/aisy.202000075

computing errors that require to be compensated through modified learning strategies or architectures. Several nonidealities cause the deterioration of the performance of memristive crossbar-based neural net architectures. **Figure 1** shows some nonidealities such as a limited number of stable resistive states,[36–38] conductance variation,[39–41] memristor aging issues,[21] endurance,[34,42] reliability issues,[43] and device failure.[35] Limiting the number of stable resistive state leads to low precision of dot product multiplication, which, in turn, reduces the memristive neural network accuracy. Figure 1a shows conductance variation, which can be generalized as a Gaussian distribution of resistive states. For devices made of different materials, the conductance variation may be different. In some architectures, conductance variation effect may be mitigated,[41] whereas for the others the result cannot be avoided.[17] Figure 1b shows memristor aging. With aging, removal of some resistive states with time occurs due to continual reprogramming of the device. And the device cannot be set to its initial R_{ON} and R_{OFF} states after several programming cycles.[21] Figure 1c shows device endurance and failure of the devices. In this case, memristive devices get stuck in R_{ON} and R_{OFF} states without possibility to reset them further. Moreover, some failed devices may be disconnected entirely from the crossbar line and do not conduct any current. Memristor failures may happen after several programming cycles or after fabrication due to imperfections of the fabrication process and can be solved by retraining the network.[44]

The other type of memristor nonideality consists of nonlinear weight distribution, asymmetry and nonlinear programming, and device-to-device and cycle-to-cycle variation.[39,45,46] The hardware noise and R_{OFF}/R_{ON} ratio can also influence the design of the memristive neural network architectures. In addition, memristive crossbars are affected by sneak path currents, wire resistances, and leakage currents.[47] Finally, signal integrity issues and electromagnetic effects in memristive crossbars can also cause the problems in the design,[48] which have not been extensively studied yet. In this work, we focus on avoiding this hardware-related issues before the design of the memristive

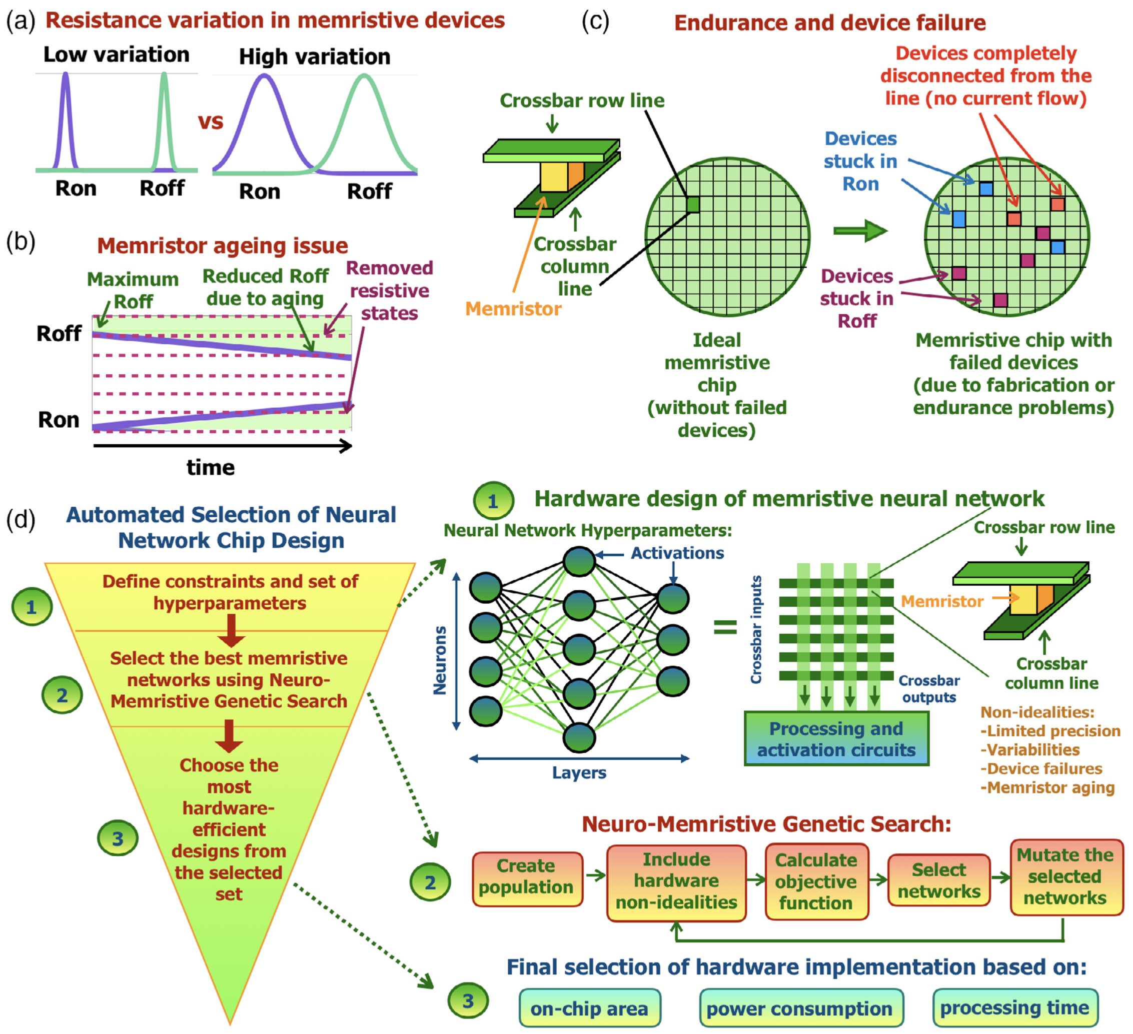

(a) Resistance variation in memristive devices
Low variation
High variation
vs
Ron
Roff
Ron
Roff
(b) Memristor ageing issue
Maximum Roff
Reduced Roff due to aging
Removed resistive states
Roff
Ron
time
(c) Endurance and device failure
Crossbar row line
Crossbar column line
Memristor
Ideal memristive chip (without failed devices)
Devices completely disconnected from the line (no current flow)
Devices stuck in Ron
Devices stuck in Roff
Memristive chip with failed devices (due to fabrication or endurance problems)
(d) Automated Selection of Neural Network Chip Design
1 Define constraints and set of hyperparameters
2 Select the best memristive networks using Neuro-Memristive Genetic Search
3 Choose the most hardware-efficient designs from the selected set
1 Hardware design of memristive neural network
Neural Network Hyperparameters:
Activations
Neurons
Layers
Crossbar inputs
Crossbar outputs
Processing and activation circuits
Crossbar row line
Memristor
Crossbar column line
Non-idealities:
-Limited precision
-Variabilities
-Device failures
-Memristor aging
Neuro-Memristive Genetic Search:
2 Create population
Include hardware non-idealities
Calculate objective function
Select networks
Mutate the selected networks
Final selection of hardware implementation based on:
3 on-chip area
power consumption
processing time

Figure 1. Memristor nonideality affecting the performance of memristive neural networks: a) resistance variation, b) device aging (when several resistive states are removed), and c) endurance and device failure. d) Design automation process.

neural network architecture, rather than mitigating the effects of nonideality. We propose to include memristor nonidealities, such as 1) a limited number of stable resistive states, 2) conductance variation, and 3) device failure, to the network hyperparameter selection algorithm, and optimize the neural network hardware prior to implementation. The proposed approach can be modified and extended to include the other hardware-related parameters and nonidealities. The parameters of several commonly used memristor devices are shown in Section 1, Supporting Information.

2.2. Hyperparameter Optimization Algorithms

The hyperparameter optimization approaches aim to identify an optimal set of parameters used to configure the network or identify learning algorithms that can lower the cost functions.[49] The automatic selection of hyperparameters eventually will remove human from the loop for neural network design and selection. In this optimization problem, the lowest error on the validation set is used to determine the most optimal hyperparameters. Also, the problem can be treated as either single- or multiobjective optimization problem.[50] In most cases, the single-objective scalar optimization cost function is used for practical implementation.[9]

There are three classes of hyperparameter optimization algorithms: 1) exhaustive search, 2) surrogate models, and 3) mix-mode search.[9] In all these cases, there are search assumptions and boundary conditions defined for hyperparameters and also takes into account any prior knowledge of the underlying problem.

The exhaustive search methods consist of a parallel grid search or random search of the space.[51] The grid search suffers from the curse of dimensionality due to parallel search problem.[52] This problem occurs because each of the hyperparameters need to be discretized. Here, the entire search space is also discretized as the Cartesian products between them. The learning and cost evaluation of each hyperparameter configuration are performed in parallel to select the best.[53]

The random search methods perform sampling of a search space as opposed to grid search where it is discretized with a Cartesian grid.[2] There is no endpoint for algorithmic computation other than setting a time limit to the algorithm. The random search also suffers from the curse of dimensionality when aiming to reach a preset fixed sampling density. As opposed to grid search, where the dependency between the hyperparameters can make it hard to find optimal, a random search does not get impacted with correlations between the hyperparameters. The random search can tolerate greater variations between the hyperparameters. Both grid and random search are easy to implement and do not require tuning. Although it can be noted that they do not guarantee a local minimum, in many realistic neural network implementations, this is not a significant concern.

Sequential Model-Based Optimization (SMBO)[54,55] is classified as a surrogate model. Here, the knowledge of the validation loss as a function of the hyperparameters is used for guessing the future tries in identifying where the local minima could be. Bayesian optimization[56] and Tree-structured Parzen Estimators (TPEs)[9,57] are two classical examples of this approach. In these methods, the learning has to be run till the end.

Hyperband[58] and Population-Based Training (PBT)[59] are specific algorithms used for hyperparameter tuning. Hyperband is a variation of random search using explore–exploit theory to find the time limit required to stop the algorithm.[58] PBT uses the ideas of genetic optimization during the stochastic gradient descent optimization step.[59] The learning is seen as dynamic with hyperparameters updated every n steps. The PBT combines the hyperparameter search with the learning, to arrive at an overall model, not just the optimal hyperparameter. As such with PBT, it is not possible to tune hyperparameters pertaining to model shape or structure. Fabolas is another algorithm to speed up Bayesian optimization for large datasets.[60] BOHB combines the Bayesian optimization and Hyperband,[61] with Hyperband used for guessing the number of configurations to try for the Bayesian optimization.[49]

The evolutionary processes in nature inspire genetic search. The search space for the hyperparameters of neural networks follows global optimization. The search can be stopped based on fitness value or limiting it to a certain number of predetermined iterations. The objective functions used in these algorithms aim for improving the accuracy or reducing the global loss of the given architecture.[62,63] When these optimized hyperparameters are used for a hardware implementation, they do not show the same accuracy or loss as expected with software simulations. This is primarily due to hardware scalability limitations and design constraints that come with nonideality. These optimization algorithms, in general, require substantial computational time, and often do not consider the hardware limitations.

3. Methodology

Figure 1d shows the overall design automation process proposed in this work. To achieve design automation, we use genetic algorithm[64] for the selection of hyperparameters required for implementing memristive crossbar-based neural network. Based on the selected parameters, we identify activation functions, an optimum size for implementing memristive crossbar and number of crossbars necessary for the overall architecture. The crossbar size and practical hardware implementations depend on the number of neurons and the number of layers in the network. Genetic search provides with the selection of several optimum designs that can tolerate nonideality of the hardware. The most efficient architecture can be selected based on minimum on-chip area, power, and processing time. After the selection of the most efficient architecture, an IC designer can proceed to designing the layout of the selected network, placing the selected components (circuits blocks), and routing and final optimization. The genetic search approach helps to automate initial design stages and can be used for any memristive devices considering corresponding parameters and nonidealities. The parameters of several commonly used memristors are shown in Section 1, Supporting Information.

In theory, the heart of the genetic algorithm can be represented as schema theorem[65] that bounds the expected growth/decay of the number n of schemata h within a population at generation G_i, as

$$n(h, G_i + 1) \geq n(h, G_i)\frac{f(h)}{\bar{f}}\left[1 - c\frac{\delta(h)}{l-1} - mo(h)\right] \tag{1}$$

where $f(h)$ is the average fitness of strings in h within the population, $\bar{f}$ is the average fitness of population, l is the string length, c is the crossover probability, $\delta(h)$ is the length of schema h, $o(h)$ is the order of schema, and m is the probability of mutation. The schema grows when the fitness value (i.e., average accuracy) is above average, is relatively short, and is of low order. This can be quantified as schema grows relative to the growth factor ϕ, $\phi = f(h)/\bar{f}[1 - c\delta(h)/(l-1) - mo(h)]$. Roughly, when $\phi \geq 1$, the future generations will contain an increased number of strings, and new stings created through recombination. At the end of each iteration, the number of networks with hyperparameters (strings) that qualify this criteria is selected and new population formed.

The nonideality of the hardware can be incorporated either by including the nonideal effects into the population or by modifying the fitness function. In neural networks, as performance accuracy is the most important objective, the fitness function evaluates average accuracy as its primary criteria. In addition to this, the performance metrics such as power and area is evaluated, such that the performance accuracy is not compromised.

The **Algorithm 1** shows the genetic algorithm that aims to take into account hardware nonideality for hyperparameter optimization. The number of generations G shows how many times the set of networks evolves to select the optimum network, where the population is the number of networks in a generation. When the number of generations $G = 1$, the hyperparameters are defined. And p networks in a population with randomly selected hyperparameters created and trained. The testing/validation of the networks was performed by including hardware nonideality, such as the limited number of stable resistive states, memristance variation, and device failure. The memristive crossbars implement both positive and negative weights. We restrict the possible range of all memristive crossbars w to the range $[x_1, x_2]$ to limit the number of stable resistive states (levels), and quantize positive and negative weights to L states (Algorithm 1, line 9). The equation in Algorithm 1 (line 12) is used to introduce memristor variation. Here, g_m is a Gaussian distribution with mean μ and standard deviation of $\sigma \times 0.01$. We also randomly disconnect F percentage of memristors in crossbars to simulate memristor failure. The experiment repeats for i iterations and average accuracy calculated.

The best x_b percentage and the worst x_w percentage networks are selected based on the average accuracy obtained. These networks are used as parents for the next generation to produce $100 - x_b - x_w$ percentage of children networks based on the parameters of the parents, where x_m percentage of the networks are mutated by randomly changing one of the hyperparameters. The parent and generated children networks in combination are used as a population of the next generation. The process repeats until it reaches an optimal set of network configurations. The library of analogue neuromemristive circuit blocks maps to the selected network configuration. The system chooses the most suitable circuit blocks for crossbars and activation functions to complete the hardware design.

4. Results and Discussion

This section provides performance analysis of the proposed method, possibilities to generalize and extend the scope of hardware implementations of the selected networks. The additional explanation of analogue memristor-based neural network design

Algorithm 1: Genetic algorithm with crossbar nonideality.

```
for $G$ generations do
  if $G = 1$ then
    Define a set of network hyperparameters
    Create a population of $p$ random networks
  Train all networks
  for all networks do
    if nonideality = true then
      Limit the range of all $w$ to $[x_1, x_2]$ and quantize for $L$ levels within this range
      for $i$ iterations do
        for all $w$ do
          Introduce memristance variation as $w = w + g_m(\mu = 0, \sigma \times 0.01)$
          for $i$ iterations do
            Randomly fail $F$ percentage of memristive weights
            Calculate network accuracy and add to the list of accuracies
    Calculate average network accuracy from the list
  Retain best $x_b$ of the best networks and $x_w$ of the worst networks (parents networks) to produce the next generation
  From the selected networks produce $100 - x_b - x_w$ of new networks (children networks)
  Mutate $x_m$ of children networks
  Create new population combining parents and children networks
```

and implementation on hardware is provided in Section 2, Supporting Information, which includes the main design considerations and the way to move from a single device to systems architecture.

4.1. Performance Analysis of Genetic Search

The deep neural networks require dataset for training and optimizing the network hyperparameters. MNIST,[23] Fashion-MNIST,[24] and first ten classes of AR Face recognition database[22] are used for our experiments. MNIST and Fashion-MNIST databases are known for benchmarking image classification problems. The simulations indicated that we select an optimum network with the hyperparameters, as shown in **Table 1**. We assume the least complex case of a fully connected network, where the size and activation of all hidden layers are the same. The explanation of the network architecture and correlation to the hardware are further discussed in Section 2.2, Supporting Information.

The dense layer parameter selection for the convolutional neural network (CNN) is demonstrated based on Table 1. Here, CNNs have a fixed four convolutional layers having [32-32-64-128] filters for face recognition application. The training stage used category-based cross-entropy loss function and Adamax optimizer.[66] The number of training epochs for MNIST, Fashion-MNIST, and AR database was set to 25, 70, and 100, respectively. Furthermore, $x_\mathrm{b} = 40\%$ of best and $x_\mathrm{w} = 10\%$ networks are retained after each generation, and $x_\mathrm{m} = 20\%$ of children networks are mutated. The weights restriction applies for the range of $[-1,1]$ for MNIST and Fashion-MNIST dataset and $[-0.2, 0.2]$ for AR dataset, which, in turn, are mapped to memristive crossbars. The limited number of stable resistive states maps with the weight to the hardware implementation of the memristive networks.

Figure 2a,b shows performance comparison for ideal and nonideal neuromemristive networks. The results report the tests with MNIST and Fashion-MNIST database. Here, we compare average generation accuracy for ideal networks and nonideal neural networks with $L = 2$, $\sigma = 25$, $F = 5\%$, $i = 50$, and $p = 20$ for ten generations and show the best three network configurations selected after ten generations. Figure 2c compares the selection of ideal network and nonideal neural networks for face recognition problem with AR database. Here, the neural network configuration is with $L = 64$, $\sigma = 3$, $F = 1\%$, $i = 100$, and $p = 20$. The figure shows the average number of neurons, activation functions, and the number of layers for the top three networks selected in each generation. The number of layers and neurons tends to increase with each generation for most of the cases. The selection of activation functions for the best three networks also varies. For example, it can be seen that for MNIST database ReLU is selected as an activation function for a hidden layer in the best networks. In contrast, for a nonideal crossbar-based network, Tanh is selected. The increased dot product error caused by hardware nonidealities and variations requires different activation functions to achieve comparable inference accuracy. The activation function selected in the ideal case cannot tolerate and compensate for this error. However, the activation function selected for the nonideal hardware components provides better compensation of dot product errors and, to a certain extent, can tolerate hardware variation resulting in the best possible performance accuracy. The selected activation functions for the top three networks vary in the first —three to four generations and remain the same after —four to five generations. The best performing networks selected for ideal and nonideal cases have different optimal network configurations. Therefore, it is essential to include the hardware nonideality into the selection of the architecture and genetic search can be successfully applied.

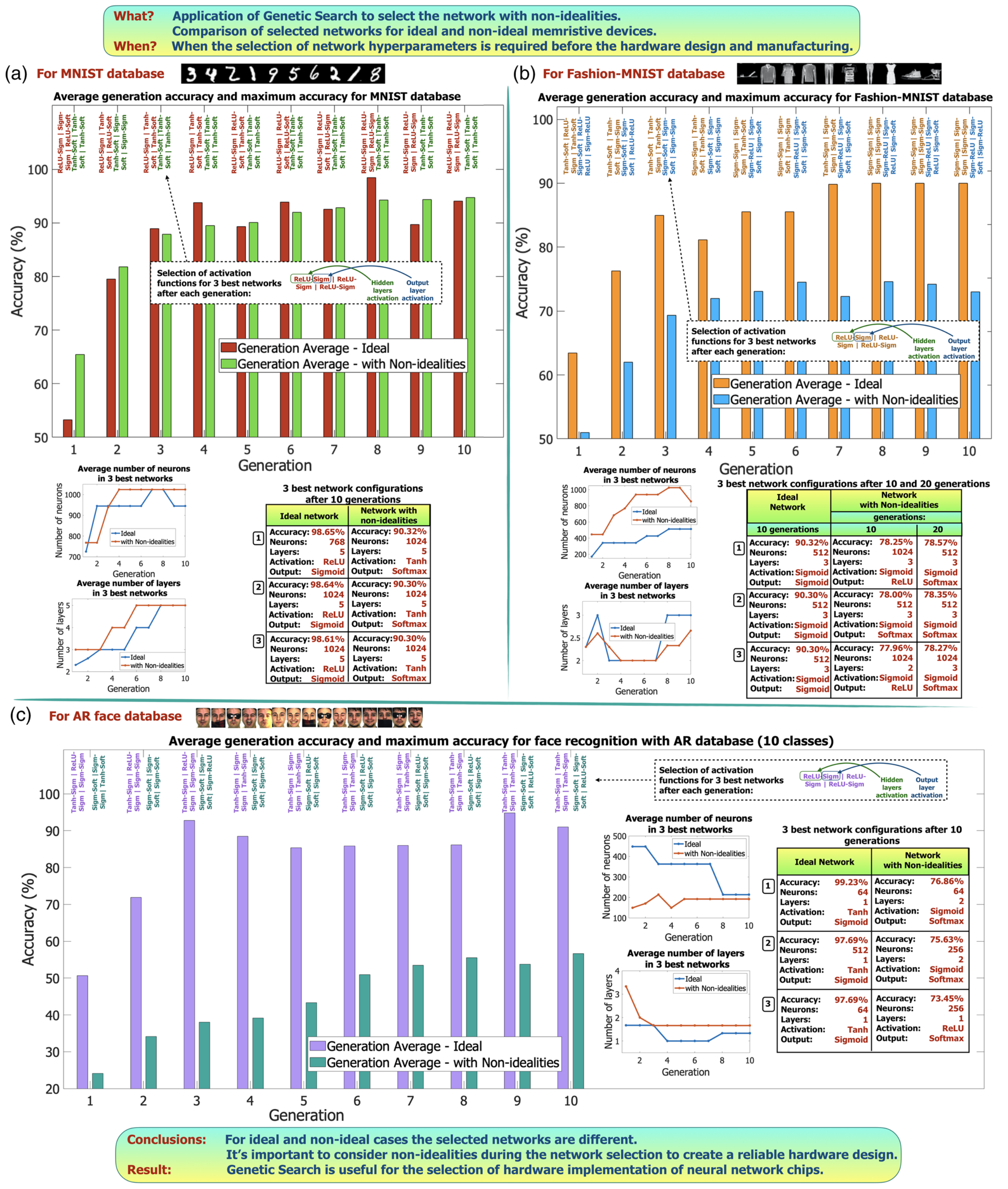
Figure 3 shows simulations for the selection of the hardware architecture, including a particular nonideality into each simulation. Such simulations are useful when a specific hardware nonideality cannot be avoided or mitigated, and the design is required to be resistant to it. The simulations are performed for MNIST and Fashion-MNIST database with $p = 20$ for the networks with $L = 2$, $\sigma = 25$, $F = 5\%$, and $i = 50$. The figure shows the average generation accuracy and maximum accuracy obtained in each generation. The comparison shows the top two selected networks obtained after first and tenth generations. The tenth generation results in configurations with higher accuracy along with first and second best networks. The proposed method allows selecting a set of network configurations that can tolerate hardware nonideality. The genetic search in the simulated experiments is at least 1.8 times faster than the grid search and training of the architectures.

Figure 4a–c shows the performance of the genetic algorithm with the variation of the number of networks in a population of p. The experiment uses Fashion-MNIST database for simulating nonideal case with the limited number of stable resistive state $L = 2$. The inference stage for accuracy calculation includes memristance variation with $\sigma = 10$ for $i = 30$ random iterations. The time required to achieve relatively high-performance accuracy was analyzed based on the average generation accuracy rather than maximum possible accuracy in each generation. The first generation of networks are randomly selected. The use of maximum accuracy as an objective metric may not be sufficient indication of generalized performance of the algorithm.

Figure 4a shows the average accuracy of the networks in a single generation for a given size of the population. Figure 4b shows the maximum accuracy in each generation. The average generation accuracy and maximum accuracy increase with the number of generations. The table in Figure 4c shows 1) the number of generations required to achieve the accuracy of approximately 80% for a different number of networks in a population, 2) the total number of networks to train and achieve 80% of accuracy, 3) approximate training time in terms of t, where t is an average training time of a single network, and 4) approximate maximum memory required to store all the networks during the training. Our experiments also consider the training time

Table 1. Hyperparameters used in the simulation.

Parameter	List of variables
Number of neurons in hidden layer	[64, 128, 256, 512, 768, 1024]
Number of layers	[1, 2, 3, 4, 5]
Activation function for hidden layers	[ReLU, Tanh, Sigmoid]
Activation function for output layer	[ReLU, Tanh, Sigmoid, Softmax]



What? Application of Genetic Search to select the network with non-idealities.
Comparison of selected networks for ideal and non-ideal memristive devices.
When? When the selection of network hyperparameters is required before the hardware design and manufacturing.

(a) 3 best network configurations after 10 generations

	Ideal network	Network with non-idealities
1	Accuracy: 98.65% Neurons: 768 Layers: 5 Activation: ReLU Output: Sigmoid	Accuracy: 90.32% Neurons: 1024 Layers: 5 Activation: Tanh Output: Softmax
2	Accuracy: 98.64% Neurons: 1024 Layers: 5 Activation: ReLU Output: Sigmoid	Accuracy: 90.30% Neurons: 1024 Layers: 5 Activation: Tanh Output: Softmax
3	Accuracy: 98.61% Neurons: 1024 Layers: 5 Activation: ReLU Output: Sigmoid	Accuracy: 90.30% Neurons: 1024 Layers: 5 Activation: Tanh Output: Softmax

(b) 3 best network configurations after 10 and 20 generations

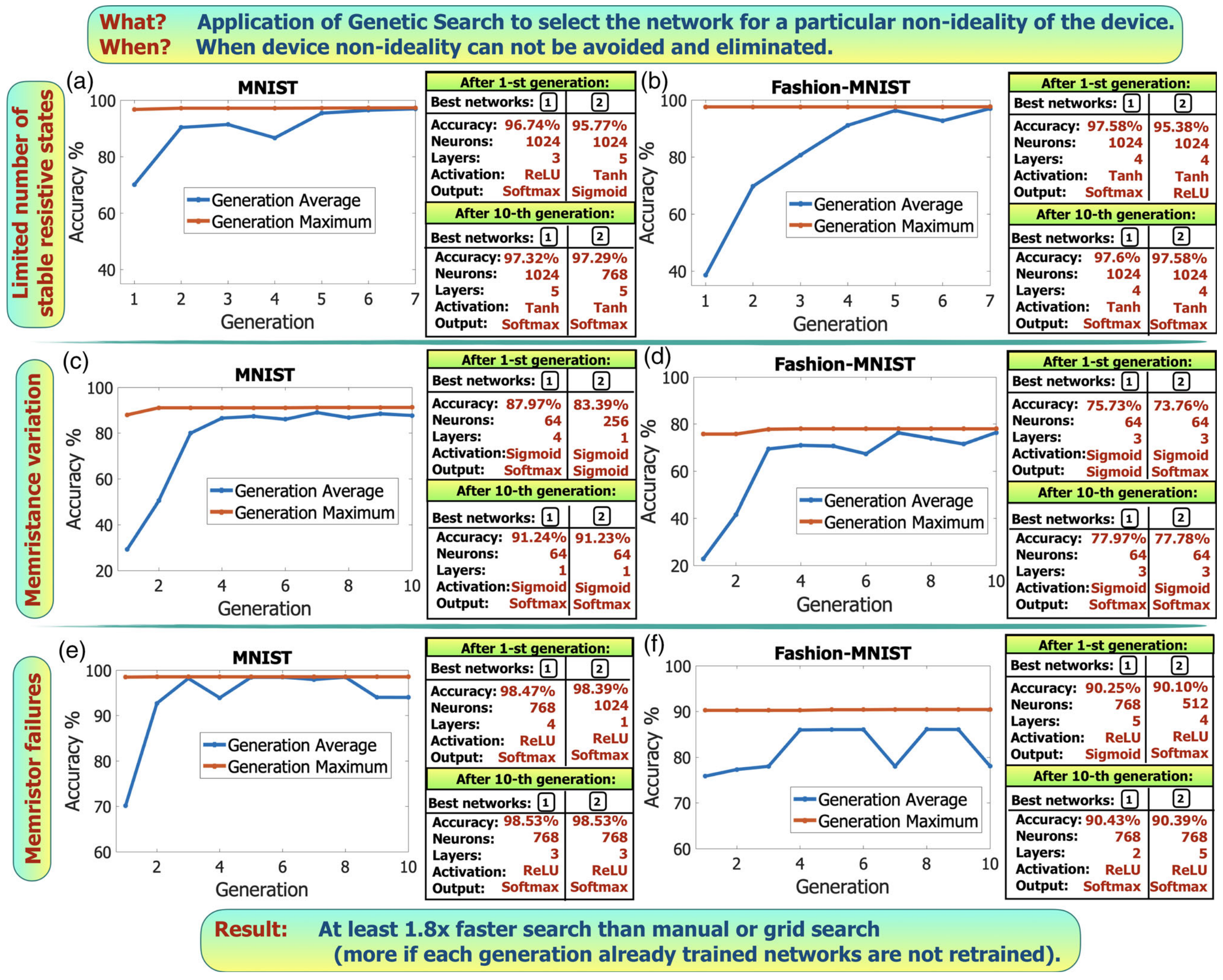
	Ideal Network	Network with Non-idealities	
		generations:	
	10 generations	10	20
1	Accuracy: 90.32% Neurons: 512 Layers: 3 Activation: Sigmoid Output: Sigmoid	Accuracy: 78.25% Neurons: 1024 Layers: 3 Activation: Sigmoid Output: ReLU	78.57% 512 3 Sigmoid Softmax
2	Accuracy: 90.30% Neurons: 512 Layers: 3 Activation: Sigmoid Output: Sigmoid	Accuracy: 78.00% Neurons: 512 Layers: 3 Activation: Sigmoid Output: Softmax	78.35% 512 3 Sigmoid Softmax
3	Accuracy: 90.30% Neurons: 512 Layers: 3 Activation: Sigmoid Output: Sigmoid	Accuracy: 77.96% Neurons: 1024 Layers: 2 Activation: Sigmoid Output: ReLU	78.27% 1024 3 Sigmoid Softmax

(c) 3 best network configurations after 10 generations

	Ideal Network	Network with Non-idealities
1	Accuracy: 99.23% Neurons: 64 Layers: 1 Activation: Tanh Output: Sigmoid	Accuracy: 76.86% Neurons: 64 Layers: 2 Activation: Sigmoid Output: Softmax
2	Accuracy: 97.69% Neurons: 512 Layers: 1 Activation: Tanh Output: Sigmoid	Accuracy: 75.63% Neurons: 256 Layers: 2 Activation: Sigmoid Output: Softmax
3	Accuracy: 97.69% Neurons: 64 Layers: 1 Activation: Tanh Output: Sigmoid	Accuracy: 73.45% Neurons: 256 Layers: 1 Activation: ReLU Output: Softmax

Conclusions: For ideal and non-ideal cases the selected networks are different.
It's important to consider non-idealities during the network selection to create a reliable hardware design.
Result: Genetic Search is useful for the selection of hardware implementation of neural network chips.

Figure 2. Comparison of average generation accuracy for ideal neural network and memristive crossbar-based neural network with nonidealities for a) MNIST database and b) Fashion-MNIST database. c) Comparison of average generation accuracy for ideal neural network and memristive crossbar-based neural network with nonidealities for face recognition with AR database, where convolutional layer is fixed and dense layer configuration is selected using genetic search.



(a) MNIST

After 1-st generation:		
Best networks:	1	2
Accuracy:	96.74%	95.77%
Neurons:	1024	1024
Layers:	3	5
Activation:	ReLU	Tanh
Output:	Softmax	Sigmoid

After 10-th generation:		
Best networks:	1	2
Accuracy:	97.32%	97.29%
Neurons:	1024	768
Layers:	5	5
Activation:	Tanh	Tanh
Output:	Softmax	Softmax

(b) Fashion-MNIST

After 1-st generation:		
Best networks:	1	2
Accuracy:	97.58%	95.38%
Neurons:	1024	1024
Layers:	4	4
Activation:	Tanh	Tanh
Output:	Softmax	ReLU

After 10-th generation:		
Best networks:	1	2
Accuracy:	97.6%	97.58%
Neurons:	1024	1024
Layers:	4	4
Activation:	Tanh	Tanh
Output:	Softmax	Softmax

(c) MNIST

After 1-st generation:		
Best networks:	1	2
Accuracy:	87.97%	83.39%
Neurons:	64	256
Layers:	4	1
Activation:	Sigmoid	Sigmoid
Output:	Softmax	Sigmoid

After 10-th generation:		
Best networks:	1	2
Accuracy:	91.24%	91.23%
Neurons:	64	64
Layers:	1	1
Activation:	Sigmoid	Sigmoid
Output:	Softmax	Softmax

(d) Fashion-MNIST

After 1-st generation:		
Best networks:	1	2
Accuracy:	75.73%	73.76%
Neurons:	64	64
Layers:	3	3
Activation:	Sigmoid	Sigmoid
Output:	Sigmoid	Softmax

After 10-th generation:		
Best networks:	1	2
Accuracy:	77.97%	77.78%
Neurons:	64	64
Layers:	3	3
Activation:	Sigmoid	Sigmoid
Output:	Softmax	Softmax

(e) MNIST

After 1-st generation:		
Best networks:	1	2
Accuracy:	98.47%	98.39%
Neurons:	768	1024
Layers:	4	1
Activation:	ReLU	ReLU
Output:	Softmax	Softmax

After 10-th generation:		
Best networks:	1	2
Accuracy:	98.53%	98.53%
Neurons:	768	768
Layers:	3	3
Activation:	ReLU	ReLU
Output:	Softmax	Softmax

(f) Fashion-MNIST

After 1-st generation:		
Best networks:	1	2
Accuracy:	90.25%	90.10%
Neurons:	768	512
Layers:	5	4
Activation:	ReLU	ReLU
Output:	Sigmoid	Softmax

After 10-th generation:		
Best networks:	1	2
Accuracy:	90.43%	90.39%
Neurons:	768	768
Layers:	2	5
Activation:	ReLU	ReLU
Output:	Softmax	Softmax

Figure 3. a,b) Average and maximum accuracy in a generation and selected network configurations for crossbar-based networks with limited number of stable resistive states $L = 2$. c,d) Average and maximum accuracy in a generation and selected network configurations for crossbar-based networks with Gaussian memristance variation $\sigma = 25$. e,f) Average and maximum accuracy in a generation and selected network configurations for crossbar-based networks with $F = 5\%$ possibility of memristor failure.

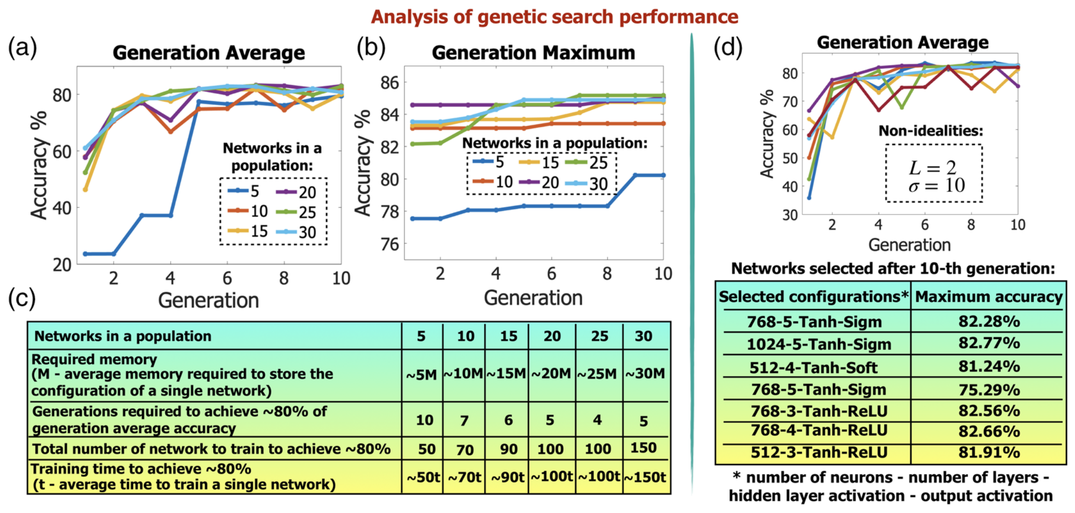
and memory for running genetic algorithms. In the estimation of the training time, we assumed that the networks in a single generation are trained sequentially. The efficient selection of the hardware architecture complies with the trade-off in the possible number of generations. The number of generations determines the hyperparameter selection speed and population size. Furthermore, implying how many networks should be trained in total to reach high-performance accuracy.

Any random selection of the initial population in the genetic algorithm can affect the network selection. We test this assertion by the selection of the network configuration under the same conditions for seven networks with nonideality ($L = 2$, $\sigma = 10$, and population $p = 10$) and different initialization in Figure 4d. The average generation accuracy for all seven cases follows the same trend with similar accuracy. The algorithm selects on average 768 neurons, —four to five layers, and the same Tahn activation function for the hidden layer for all the different random initializations, whereas the output layer function varied between Sigmoid and ReLU.

4.2. Generalized Scope of Proposed Genetic Search

The proposed approach can be used to design different memristor-based neural network architectures considering various hardware aspects and nonidealities. The number of network hyperparameters can be further extended. The genetic search does not depend on the training/optimization algorithm, and this algorithm can be selected based on designer's preferences. Moreover, selection of the training algorithm can also be considered as a part of the hyperparameter selection process.

Figure 5 shows how the scope and generality of the proposed approach can be expanded. The number of the hyperparameters selected during the optimization can be increased, as shown in Figure 5a. In comparison to the simulations shown in Section 4.1



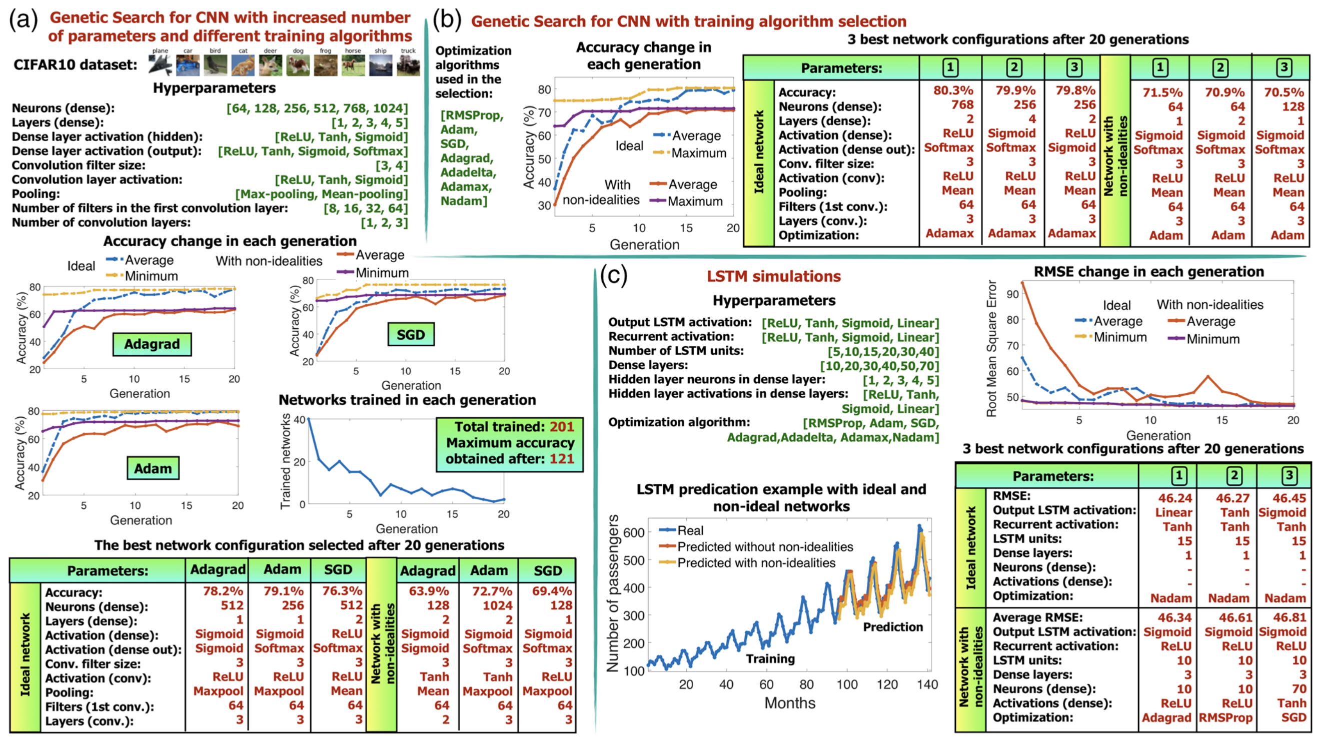
Networks in a population	5	10	15	20	25	30
Required memory (M - average memory required to store the configuration of a single network)	~5M	~10M	~15M	~20M	~25M	~30M
Generations required to achieve ~80% of generation average accuracy	10	7	6	5	4	5
Total number of network to train to achieve ~80%	50	70	90	100	100	150
Training time to achieve ~80% (t - average time to train a single network)	~50t	~70t	~90t	~100t	~100t	~150t

Networks selected after 10-th generation:

Selected configurations*	Maximum accuracy
768-5-Tanh-Sigm	82.28%
1024-5-Tanh-Sigm	82.77%
512-4-Tanh-Soft	81.24%
768-5-Tanh-Sigm	75.29%
768-3-Tanh-ReLU	82.56%
768-4-Tanh-ReLU	82.66%
512-3-Tanh-ReLU	81.91%

* number of neurons - number of layers - hidden layer activation - output activation

Figure 4. Effect of the number of networks in a population on the performance of genetic algorithm for Fashion-MNIST database: a) generation average, b) generation maximum, and c) required number of generations and trained networks. d) Effect of random initialization on the performance of optimization algorithm.



Hyperparameters

Neurons (dense):	[64, 128, 256, 512, 768, 1024]
Layers (dense):	[1, 2, 3, 4, 5]
Dense layer activation (hidden):	[ReLU, Tanh, Sigmoid]
Dense layer activation (output):	[ReLU, Tanh, Sigmoid, Softmax]
Convolution filter size:	[3, 4]
Convolution layer activation:	[ReLU, Tanh, Sigmoid]
Pooling:	[Max-pooling, Mean-pooling]
Number of filters in the first convolution layer:	[8, 16, 32, 64]
Number of convolution layers:	[1, 2, 3]

The best network configuration selected after 20 generations

	Parameters:	Adagrad	Adam	SGD		Adagrad	Adam	SGD
Ideal network	Accuracy:	78.2%	79.1%	76.3%	Network with non-idealities	63.9%	72.7%	69.4%
	Neurons (dense):	512	256	512		128	1024	128
	Layers (dense):	1	1	2		2	2	1
	Activation (dense):	Sigmoid	Sigmoid	ReLU		Sigmoid	Sigmoid	Sigmoid
	Activation (dense out):	Sigmoid	Softmax	Softmax		Sigmoid	Softmax	Softmax
	Conv. filter size:	3	3	3		3	3	3
	Activation (conv):	ReLU	ReLU	ReLU		Tanh	Tanh	ReLU
	Pooling:	Maxpool	Maxpool	Mean		Mean	Maxpool	Maxpool
	Filters (1st conv.):	64	64	64		64	64	64
	Layers (conv.):	3	3	3		2	3	3

3 best network configurations after 20 generations

	Parameters:	1	2	3		1	2	3
Ideal network	Accuracy:	80.3%	79.9%	79.8%	Network with non-idealities	71.5%	70.9%	70.5%
	Neurons (dense):	768	256	256		64	64	128
	Layers (dense):	2	4	2		1	2	1
	Activation (dense):	ReLU	Sigmoid	ReLU		Sigmoid	Sigmoid	Sigmoid
	Activation (dense out):	Softmax	Softmax	Sigmoid		Softmax	Softmax	Softmax
	Conv. filter size:	3	3	3		3	3	3
	Activation (conv):	ReLU	ReLU	ReLU		ReLU	ReLU	ReLU
	Pooling:	Mean	Mean	Mean		Mean	Mean	Mean
	Filters (1st conv.):	64	64	64		64	64	64
	Layers (conv.):	3	3	3		3	3	3
	Optimization:	Adamax	Adamax	Adamax		Adam	Adam	Adam

Hyperparameters

Output LSTM activation:	[ReLU, Tanh, Sigmoid, Linear]
Recurrent activation:	[ReLU, Tanh, Sigmoid, Linear]
Number of LSTM units:	[5,10,15,20,30,40]
Dense layers:	[10,20,30,40,50,70]
Hidden layer neurons in dense layer:	[1, 2, 3, 4, 5]
Hidden layer activations in dense layers:	[ReLU, Tanh, Sigmoid, Linear]
Optimization algorithm:	[RMSProp, Adam, SGD, Adagrad,Adadelta, Adamax,Nadam]

3 best network configurations after 20 generations

	Parameters:	1	2	3
Ideal network	RMSE:	46.24	46.27	46.45
	Output LSTM activation:	Linear	Tanh	Sigmoid
	Recurrent activation:	Tanh	Tanh	Tanh
	LSTM units:	15	15	15
	Dense layers:	1	1	1
	Neurons (dense):	-	-	-
	Activations (dense):	-	-	-
	Optimization:	Nadam	Nadam	Nadam
Network with non-idealities	Average RMSE:	46.34	46.61	46.81
	Output LSTM activation:	Sigmoid	Sigmoid	Sigmoid
	Recurrent activation:	ReLU	ReLU	ReLU
	LSTM units:	10	10	10
	Dense layers:	3	3	3
	Neurons (dense):	10	10	70
	Activations (dense):	ReLU	ReLU	Tanh
	Optimization:	Adagrad	RMSProp	SGD

Figure 5. a) Selection of CNN architecture for object recognition for CIFAR-10 dataset with nine hyperparameters. b) Selection of CNN architecture considering the training/optimization algorithm as a network hyperparameter. c) Example of genetic search application for memristive LSTM architecture design.

where only four hyperparameters are selected during optimization (Table 1), Figure 5a shows the CNN simulations for the selection of nine CNN hyperparameters for object recognition application using CIFAR-10 dataset.[67] The hyperparameter set shown in Table 1 (Figure 5a) includes convolution filter size, convolution activation function, type of pooling in convolution layer (mean-pooling or max-pooling), number of convolution layers, and the number of convolution filters in the first convolution layer (the number of filters doubled in the subsequent convolutional layers). In addition, we illustrate that the genetic search does not depend on the training algorithm and perform the simulations for three different training/optimization methods:

adaptive gradient algorithm (Adagrad),[68] adaptive moment estimation (Adam),[66] and stochastic gradient descent (SGD).[69] As the number of hyperparameters increased, we also increased the population size to 40 networks per population ($p = 40$). The nonidealities have been set to $L = 64$, $\sigma = 2$, and $F = 1\%$ to emulate Ta/hfO$_2$/Pd devices (the additional parameters are shown in Figure S1, Supporting Information). Figure 5a shows the best selected networks after 20 generations for ideal and nonideal cases, minimum and maximum accuracy per generation, and number of new networks trained for each generation. For all three training algorithms, the network configuration producing maximum accuracy is found within the first five to six generations. The number of new trained networks decreases each generation (shown in Figure 5a for the nonideal case with Adam-based optimization) as genetic search converges to a particular solution. The total number of networks trained for 20 generations is 201 and the best accuracy is found after training of 121 networks. However in the grid-search, 51 840 networks have to be trained to arrive at the best solution. The results show that the proposed genetic search is efficient when the number of hyperparameters increases. This is an important feature for the selection of nonideal hardware solutions for neural network implementation, as a designer can increase the number of network parameters and even include various hardware-related features to the selection set without compromising on hyperparameter selection speed.

In the initial design stages of a memristive neural network architecture for a new application, the training/optimization method leading to the highest accuracy or any other performance metrics is unknown. Therefore, at this stage, the selection of the training algorithm can also be included as a parameter to the set of hyperparameter. The simulation of such case is shown in Figure 5b, where three best CNN architectures for ideal and nonideal cases are shown for the same problem as shown in Figure 5a. The additional parameters in the optimization algorithm are added to the set of network hyperparameters. The set of optimization algorithms includes Adagrad, Adam, SGD, RMSProp,[70] Adadelta,[71] Adamax,[66] and Nadam.[72] For ideal and nonideal cases, different optimization algorithms are selected; activation functions, the number of layers, and neurons also vary. The selection of optimization method increases the required number of generations to achieve maximum accuracy to 15 and 10 in ideal and nonideal cases, respectively. Smaller number of generations are required to achieve maximum accuracy in nonideal case, as the maximum performance accuracy is restricted by memristor nonidealities, mainly limited precision, and has a low chance to increase after certain number of generations even after modifying the architecture. The selection of the training algorithm as a parameter in the genetic search can also be useful for the design of the architecture with on-chip training.

The application scope of the proposed genetic search can be expanded to any hardware implementation of memristor-based neural network, where memristors are used to implement neural network weights, e.g., Spiking Neural Network,[73] Long Short Term Memory (LSTM),[74] and Hierarchical Temporal Memory.[75] Figure 5c shows an example of genetic search application for the selection of LSTM architecture with nonidealities for the number of airline passengers prediction.[74] The set of hyperparameters includes LSTM activation functions (output and recurrent), number of LSTM units, dense layers number and activations, and optimization algorithms. The overall LSTM theory and setup are explained in Section 2.3, Supporting Information. The output dense layer has a single output, which indicates the predicted number of passengers. The optimization setup was modified to minimize root mean square error (RMSE) between real and predicted outputs. The parameters of nonidealities are set to $L = 64$, $\sigma = 2$, $F = 1\%$, and the population $p = 3$. Figure 5c shows the set of hyperparameters, average and minimum RMSE for ideal and nonideal cases, three best selected networks, and the difference between ideal and nonideal predicted outputs for the best selected architectures. RMSE in ideal and nonideal cases for the best selected architectures is the same, indicating that it is possible to select LSTM architecture where memristor nonidealities do not deteriorate the performance. For nonideal case, the selected architectures have larger dense layer part, different activation functions and optimization algorithms, and smaller number of LSTM units.

The results shown in this section illustrate that the application scope of the proposed genetic search for memristive neural network design can be expanded to any architecture with memristive weights. The number of hyperparameter can be increased and training-related details can be added to the list of hyperparameters to be selected. The proposed genetic search method can also be modified considering various hardware-related aspects.

4.3. Analysis of the Selected Hardware Implementations and the Effects on the Final Design

We observe that with our proposed approach accuracy of the best three selected networks is almost equal. In contrast, the network configurations may vary (Figure 2b). The selected number of layers, number of neurons, and activation functions affect the on-chip area and power consumption of memristive-CMOS hybrid analogue neural networks.[17,76,77] **Figure 6** and **Table 2** compare the hardware parameters of three best memristive networks for Fashion-MNIST considering memristor nonideality from Figure 2b. Figure 6 shows the example of a hardware implementation of the selected networks, considering the parallel processing of all crossbar columns and sequential processing of each crossbar. Each network weight is represented by two memristors to ensure negative weights. And each column contains two operational amplifiers (op-amps) to subtract the outputs of two columns and convert to a single output. The op-amp output is connected to the activation function followed by the next layer crossbar. More hardware specific background details are furnished in Supporting Information.

In Table 2, the memristive crossbar area is calculated as $A_{\mathrm{m}}[(784 + 10)n + n^2(l - 2)]$, where n is the number of neurons, l is the number of layers, A_{m} is a memristor area, and the network has 784 inputs and 10 output. The CMOS area includes the area of activation functions and op-amps. The area for this is calculated as $n(l-1)A_{\mathrm{hidden}} + 10A_{\mathrm{output}} + (n(l-1) + 10)A_{\mathrm{opamps}}$, where A_{hidden}, A_{output}, and A_{opams} are the areas of activation functions in hidden layer and output layer and the area of two op-amp circuit (Figure 6), respectively. Because the processing of the crossbars is performed sequentially (e.g., three-layer network is processed in three steps), the largest layer in the

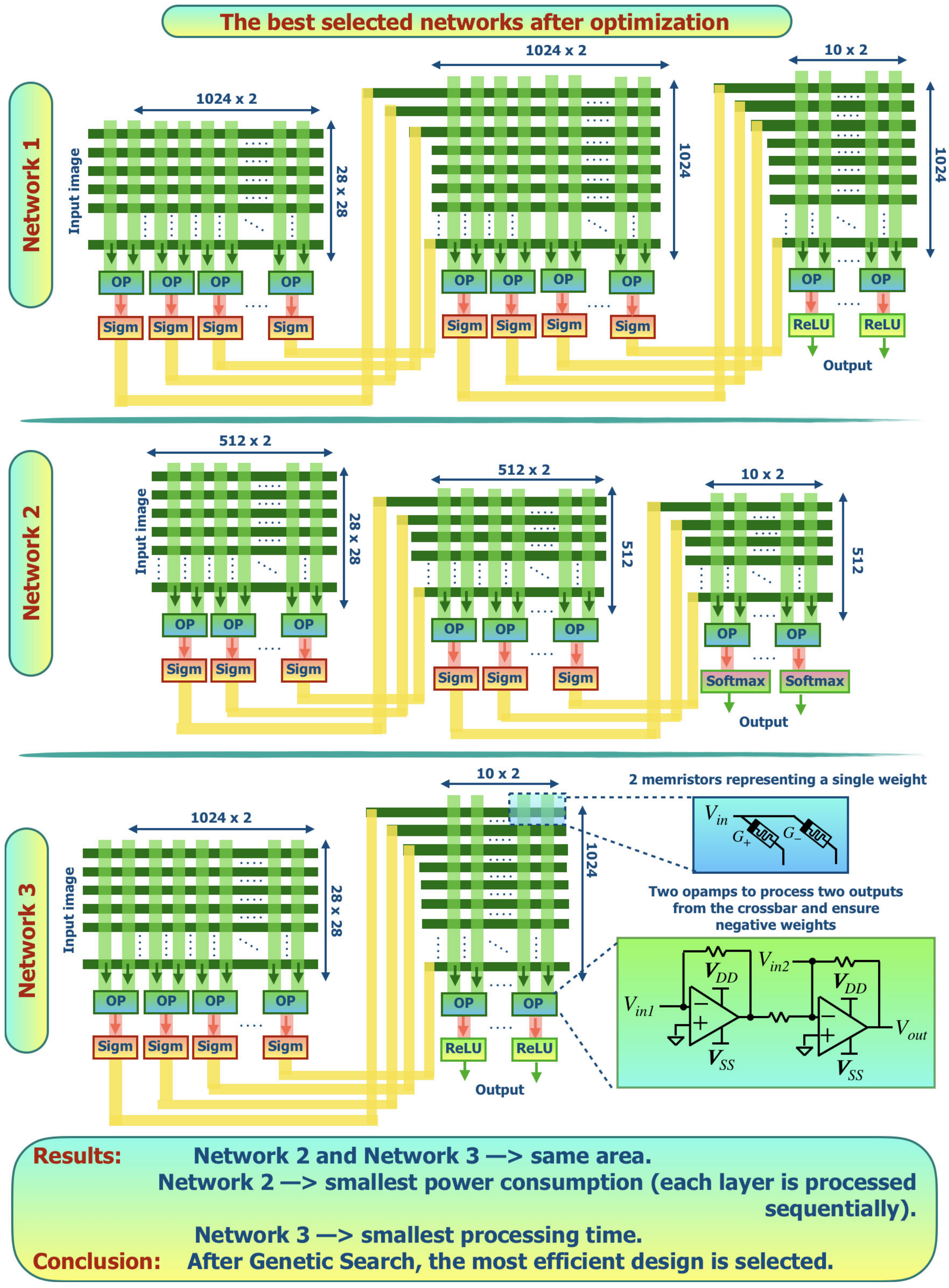

The best selected networks after optimization
Network 1
Input image
1024 x 2
28 x 28
OP
Sigm
1024 x 2
1024
10 x 2
1024
ReLU
Output
Network 2
512 x 2
28 x 28
512 x 2
512
10 x 2
512
Softmax
Output
Network 3
1024 x 2
28 x 28
10 x 2
1024
ReLU
Output
2 memristors representing a single weight
V_{in}
G_+
G_-
Two opamps to process two outputs from the crossbar and ensure negative weights
V_{in1}
V_{in2}
V_{DD}
V_{SS}
V_{out}
Results: Network 2 and Network 3 —> same area.
Network 2 —> smallest power consumption (each layer is processed sequentially).
Network 3 —> smallest processing time.
Conclusion: After Genetic Search, the most efficient design is selected.

Figure 6. Hardware implementation of the selected networks from Table 2. After the genetic search, the optimum and most efficient hardware implementation can be selected based on the designer's preferences.

network has the maximum power consumption. Table 2 and Figure 6 show that the best selected networks have a different area and power parameters. The implementation of the second network shows as the most feasible on hardware due to the smallest power consumption. In contrast, the third network illustrates the highest processing speed.

Table 2. Comparison of top three selected networks for Fashion-MNIST database considering crossbar nonidealities (A_m, memristor area; A_{sigm}, sigmoid area; A_{relu}, ReLU area; A_{op2}, area of two op-amps (Figure 6); A_{soft}, softmax area; P_{sigm}, sigmoid power; P_{op2}, power consumption of two op-amps).

Network configuration[a)]	Crossbars area	CMOS area[b)]	Max CMOS power[c)]
1024-3-Sigm-ReLU	$1\,861\,632 \times 2A_m$	$2048A_{sigm} + 10A_{relu} + 2058A_{op2} \approx 2048A_{sigm} + 2058A_{op2}$	$\approx 1024(P_{sigm} + P_{op2})$
512-3-Sigm-Softmax	$668\,672 \times 2A_m$	$1024A_{sigm} + 10A_{soft} + 1034A_{op2} \approx 1024A_{sigm} + 1034A_{op2}$	$\approx 512(P_{sigm} + P_{op2})$
1024-2-Sigm-ReLU	$813\,056 \times 2A_m$	$1024A_{sigm} + 10A_{relu} + 1034A_{op2} \approx 1024A_{sigm} + 1034A_{op2}$	$\approx 1024(P_{sigm} + P_{op2})$

a)Number of neurons–number of layers–hidden layer activation output activation; b)CMOS output area is calculated only for activation functions; c)Maximum power is considered for parallel processing in the largest crossbar.

The proposed genetic search allows selecting several network configurations considering hardware nonideality. This can be used to identify the most optimum architecture in terms of on-chip area and power consumption. This approach can speed up the selection of analogue neuromemristive standard cells and design process for implementing application-specific neural chips for edge devices.

5. Conclusion

We presented a genetic search approach to accelerate the application-specific neuromemristive AI chip design process. The optimum hardware uses the existing library of neuromemristive standard cells. The optimization process of the network architecture considers the nonideal behavior of neuromemristive networks. The proposed system selects different network hyperparameters for ideal networks and networks with crossbar nonideality. Introducing nonideality showed that the proposed genetic search ends up choosing architecture with more number of neurons and network layers. The method is useful for speeding up in arriving at the preliminary design of the neural network architecture before performing the circuit-level simulations of the memristive neural network architecture. The circuit-level simulations with nonlinear memristive devices can take an enormous amount of time. Traditionally, the memristor models do not consider all the required parameters to illustrate a limited number of stable resistive states and the possibility of device failure. Therefore, the genetic search is a promising solution to speed up neuromemristive hyperparameters selection. In addition, we show that the most optimum neural network in terms of on-chip area and power consumption may be selected based on the results of the algorithm. The generalization of the approach to include different types of neural networks was also presented. The proposed approach can help to arrive at optimized systems speeding up the development of analogue coprocessors. Some of the possible edge applications include the multimodal signature, iris, fingerprint, face recognition with more classes, and other biometric applications.

Supporting Information

Supporting Information is available from the Wiley Online Library or from the author.

Acknowledgements

Research reported in this publication was supported by the AI Initiative, King Abdullah University of Science and Technology (KAUST). The support provided through 2019 research internship program at KAUST is acknowledged.

Conflict of Interest

The authors declare no conflict of interest.

Keywords

genetic algorithm, hyperparameter selection, memristive networks, memristor nonidealities, neuromorphic chip design

Received: April 11, 2020
Revised: May 27, 2020
Published online: June 25, 2020

[1] M. Rahimi Azghadi, Y.-C. Chen, J. K. Eshraghian, J. Chen, C.-Y. Lin, A. Amirsoleimani, A. Mehonic, A. J. Kenyon, B. Fowler, J. C. Lee, Y.-F. Chang, *Adv. Intell. Syst.* **2020**, *2*, 1900189.
[2] J. Bergstra, Y. Bengio, *J. Mach. Learn. Res.* **2012**, *13*, 281.
[3] J. Snoek, O. Rippel, K. Swersky, R. Kiros, N. Satish, N. Sundaram, M. Patwary, M. Prabhat, R. Adams, in *Int. Conf. Machine Learning*, PMLR, Lille, France **2015**, pp. 2171–2180.
[4] T. Kim, J. Lee, Y. Choe, *IEEE Access* **2020**, *8*, 17605.
[5] Y. Xiao, H. Wang, W. Xu, *IEEE Trans. Neural Netw. Learn. Syst.* **2015**, *26*, 2182.
[6] S. R. Young, D. C. Rose, T. P. Karnowski, S.-H. Lim, R. M. Patton, in *Proc. of the Workshop on Machine Learning in High-Performance Computing Environments*, ACM, Austin, TX **2015**, pp. 1–5.
[7] F. Friedrichs, C. Igel, *Neurocomputing* **2005**, *64*, 107.
[8] D. B. Fogel, in *BCEC*, World Scientific Press, Sweden **1997**, pp. 1–11.
[9] J. S. Bergstra, R. Bardenet, Y. Bengio, B. Kégl, *Advances in Neural Information Processing Systems*, Curran Associates Inc., Granada, Spain **2011**, p. 2546.
[10] L. Hu, L. Qin, K. Mao, W. Chen, X. Fu, *IEEE Sen. J.* **2016**, *16*, 1158.
[11] G. Leng, T. M. McGinnity, G. Prasad, *IEEE Trans. Fuzzy Syst.* **2006**, *14*, 755.
[12] F. H. F. Leung, H. K. Lam, S. H. Ling, P. K. S. Tam, *IEEE Trans. Neural Netw.* **2003**, *14*, 79.
[13] R. Zhang, J. Tao, *IEEE Trans. Ind. Electron.* **2018**, *65*, 5882.
[14] S. Roh, W. Pedrycz, S. Oh, *IEEE Trans. Ind. Electron.* **2007**, *54*, 2219.
[15] M. A. Zidan, A. M. Eltawil, F. Kurdahi, H. A. Fahmy, K. N. Salama, *IEEE Trans. Nanotechnol.* **2014**, *13*, 274.

[16] M. Al-Shedivat, R. Naous, G. Cauwenberghs, K. N. Salama, *IEEE J. Emerg. Sel. Top. Circuits Syst.* **2015**, *5*, 242.
[17] O. Krestinskaya, K. N. Salama, A. P. James, *IEEE Trans. Circuits Syst. I: Regular Pap.* **2018**, *66*, 719.
[18] R. Naous, M. Al-Shedivat, K. N. Salama, *IEEE Trans. Nanotechnol.* **2015**, *15*, 15.
[19] R. Naous, M. Al-Shedivat, E. Neftci, G. Cauwenberghs, K. N. Salama, in *IEEE Int. Symp. on Circuits and Systems (ISCAS)*, IEEE, Montreal **2016**, pp. 2078–2081.
[20] S. N. Mozaffari, K. P. Gnawali, S. Tragoudas, in *Proc. of the 14th IEEE/ACM Int. Symp. on Nanoscale Architectures*, ACM, Athens, Greece **2018**, pp. 25–30.
[21] S. Zhang, G. L. Zhang, B. Li, H. H. Li, U. Schlichtmann, in *Design, Automation & Test in Europe Conf. & Exhibition (DATE)* IEEE, Florence **2019**, 1751–1756.
[22] A. M. Martinez, R. Benavente, *The AR Face Database*, CVC Technical Report #24, **1998**.
[23] L. Deng, *IEEE Signal Process. Mag.* **2012**, *29*, 141.
[24] H. Xiao, K. Rasul, R. Vollgraf, arXiv preprint arXiv:1708.07747 **2017**.
[25] S.-M. S. Kang, S. Shin, *Memristors and Memristive Systems*, Springer, New York, NY **2014**, pp. 301–325.
[26] I. Vourkas, G. Papandroulidakis, G. C. Sirakoulis, A. Abusleme, *IJUC* **2016**, *12*, 265.
[27] M. Hu, H. Li, Y. Chen, Q. Wu, G. S. Rose, R. W. Linderman, *IEEE Trans. Neural Netw. Learn. Syst.* **2014**, *25*, 1864.
[28] I. Vourkas, G. C. Sirakoulis, *Handbook of Memristor Networks*, Springer, Cham **2019**, pp. 973–1004.
[29] S. Pi, C. Li, H. Jiang, W. Xia, H. Xin, J. J. Yang, Q. Xia, *Nat. Nanotechnol.* **2019**, *14*, 35.
[30] T. M. Taha, R. Hasan, C. Yakopcic, in *27th IEEE Int. System-on-Chip Conf. (SOCC)*, IEEE, Las Vegas, NV **2014**, 383–389.
[31] L. Zheng, S. Shin, S.-M. S. Kang, in *IEEE Int. Symp. on Circuits and Systems (ISCAS)*, IEEE, Lisbon **2015**, pp. 1150–1153.
[32] I. Vourkas, G. C. Sirakoulis, *Memristor-Based Nanoelectronic Computing Circuits and Architectures*, Vol. *19*, Springer, Cham **2016**.
[33] V. Milo, G. Malavena, C. Monzio Compagnoni, D. Ielmini, *Materials* **2020**, *13*, 166.
[34] D. Ielmini, S. Ambrogio, *Nanotechnology* **2019**, *9*, 092001.
[35] O. Krestinskaya, B. Choubey, A. James, *Sci. Rep.* **2020**, *10*, 1.
[36] Z. Sun, E. Ambrosi, G. Pedretti, A. Bricalli, D. Ielmini, *IEEE Trans. Electron Dev.* **2020**, *67*, 1466.
[37] C. Li, D. Belkin, Y. Li, P. Yan, M. Hu, N. Ge, H. Jiang, E. Montgomery, P. Lin, Z. Wang, W. Song, J. P. Strachan, M. Barnell, Q. Wu, R. S. Williams, J. J. Yang, Q. Xia, *Nat. Commun.* **2018**, *9*, 1.
[38] O. Krestinskaya, A. Irmanova, A. P. James, in *IEEE Int. Symp. on Circuits and Systems*, IEEE, Sapporo **2019**, 1–5.
[39] B. Yan, B. Li, X. Qiao, C.-X. Xue, M.-F. Chang, Y. Chen, H. Li, *Adv. Intell. Syst.* **2019**, *7*, 1900068.
[40] H. Nili, G. C. Adam, B. Hoskins, M. Prezioso, J. Kim, M. R. Mahmoodi, F. M. Bayat, O. Kavehei, D. B. Strukov, *Nat. Electron.* **2018**, *1*, 197.
[41] D. Querlioz, O. Bichler, C. Gamrat, in *Int. Joint Conf. on Neural Networks*, IEEE, San Jose, CA **2011**, pp. 1775–1781.
[42] B. Yan, M. Liu, Y. Chen, K. Chakrabarty, H. Li, in *IEEE Int. Electron Devices Meeting (IEDM)*, IEEE, San Francisco, CA **2019**, 14–5.
[43] K. Chakrabarty, T.-Y. Ho, H. Li, U. Schlichtmann, *Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik*, Dagstuhl, Germany **2019**.
[44] C. Liu, M. Hu, J. P. Strachan, H. Li, in *54th ACM/EDAC/IEEE Design Automation Conf. (DAC)*, IEEE, Austin, TX **2017**, pp. 1–6.
[45] Y. Wang, L. Yu, S. Wu, R. Huang, Y. Yang, *Adv. Intell. Syst.* **2020**, *2*, 3.
[46] P.-Y. Chen, X. Peng, S. Yu, *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.* **2018**, *37*, 3067.
[47] R. Naous, M. A. Zidan, A. Sultan, K. N. Salama, *Microelectron. J.* **2016**, *54*, 48.
[48] A. P. James, *Nat. Electron.* **2019**, *2*, 268.
[49] M. Feurer, F. Hutter, *Automated Machine Learning*, Springer, Cham **2019**, pp. 3–33.
[50] S. C. Smithson, G. Yang, W. J. Gross, B. H. Meyer, in *Proc. of the 35th Int. Conf. on Computer-Aided Design*, ACM, Austin, TX **2016**, pp. 1–8.
[51] N. Amos, C. Stewart, P. Bhat, C. Cretsinger, E. Won, W. Dharmaratna, H. Prosper, in *Computing in High Energy Physics' 95: CHEP'95*, World Scientific, Singapore **1996**, pp. 215–219.
[52] F. J. Pontes, G. Amorim, P. P. Balestrassi, A. Paiva, J. R. Ferreira, *Neurocomputing* **2016**, *186*, 22.
[53] Á. B. Jiménez, J. L. Lázaro, J. R. Dorronsoro, *Innovations in Hybrid Intelligent Systems*, Springer, Berlin, Heidelberg **2007**, pp. 120–127.
[54] R. Bardenet, M. Brendel, B. Kégl, M. Sebag, in *Int. Conf. Machine Learning*, JMLR, Atlanta, GA **2013**, pp. 199–207.
[55] M. Wistuba, N. Schilling, L. Schmidt-Thieme, in *Joint European Conf. on Machine Learning and Knowledge Discovery in Databases*, Springer, Cham **2015**, pp. 104–119.
[56] M. Pelikan, D. E. Goldberg, E. Cantú-Paz, in *Proc. of the Genetic and Evolutionary Computation Conf. GECCO-99*, Vol. *1*, Morgan Kaufmann Publishers, Orlando, FL **1999**, pp. 525–532.
[57] F. Oveisi, S. Oveisi, A. Erfanian, I. Patras, *IEEE Trans. Neural Netw. Learning Syst.* **2011**, *23*, 127.
[58] L. Li, K. Jamieson, G. DeSalvo, A. Rostamizadeh, A. Talwalkar, *J. Mach. Learn. Res.* **2017**, *18*, 6765.
[59] M. Jaderberg, V. Dalibard, S. Osindero, W. M. Czarnecki, J. Donahue, A. Razavi, O. Vinyals, T. Green, I. Dunning, K. Simonyan, C. Fernando, arXiv preprint arXiv:1711.09846 **2017**.
[60] A. Klein, S. Falkner, S. Bartels, P. Hennig, F. Hutter, in *Int. Conf. on Artificial Intelligence and Statistics (AISTATS)*, JMLR, Fort Lauderdale, FL **2017**, pp. 528–536.
[61] S. Falkner, A. Klein, F. Hutter, in *Int. Conf. on Machine Learning*, PMLR, Stockholm, Sweden **2018**, pp. 1437–1446.
[62] X. Yao, in *Proc. of IEEE Int. Symp. on Parallel Algorithms Architecture Synthesis*, IEEE, Aizu-Wakamatsu **1997**, pp. 282–291.
[63] S. Ding, C. Su, J. Yu, *Artif. Intell. Rev.* **2011**, *36*, 153.
[64] D. Whitley, *Statistics Comput.* **1994**, *4*, 65.
[65] L. Altenberg, *Foundations of Genetic Algorithms*, Vol. *3*, Elsevier, San Francisco, CA **1995**, pp. 23–49.
[66] D. P. Kingma, J. Ba, in *Int. Conf. on Learning Representations (ICLR)*, San Diego, CA **2015**.
[67] A. Krizhevsky, Learning Multiple Layers of Features from Tiny Images. Technical Report TR-2009, University of Toronto, Toronto, **2009**.
[68] J. Duchi, E. Hazan, Y. Singer, *J. Mac. Learn. Res.* **2011**, *12*, 2121.
[69] I. Sutskever, J. Martens, G. Dahl, G. Hinton, in *Int. Conf. on Machine Learning*, JMLR, Atlanta, GA **2013**, pp. 1139–1147.
[70] G. Hinton, N. Srivastava, K. Swersky, **2012**, *14*, 8.
[71] M. D. Zeiler, arXiv preprint arXiv:1212.5701 **2012**.
[72] T. Dozat, in *Int. Conf. on Learning Representations (ICLR)*, Caribe Hilton, San Juan, Puerto Rico **2016**.
[73] P. Wijesinghe, A. Ankit, A. Sengupta, K. Roy, *IEEE Trans. Emerg. Top. Comput. Intell.* **2018**, *2*, 345.
[74] C. Li, Z. Wang, M. Rao, D. Belkin, W. Song, H. Jiang, P. Yan, Y. Li, P. Lin, M. Hu, N. Ge, J. P. Strachan, M. Barnell, Q. Wu, R. S. Williams, J. J. Yang, Q. Xia, *Nat. Mach. Intell.* **2019**, *1*, 49.
[75] O. Krestinskaya, I. Dolzhikova, A. P. James, *IEEE Trans. Emerg. Top. Comput. Intell.* **2018**, *2*, 380.
[76] Q. Xu, S. Chen, B. Yu, F. Wu, in *Proc. of the 2018 on Great Lakes Symp. on VLSI*, ACM, Chicago, IL **2018**, pp. 451–454.
[77] Q. Xia, J. J. Yang, *Nat. Mater.* **2019**, *18*, 309.