Enhanced methods for Evolution in-Materio Processors

Benedict. A. H. Jones
Department of Engineering
Durham University
Durham, DH1 3LE, UK
benedict.jones@durham.ac.uk

Noura Al Moubayed
Department of Computer Science
Durham University
Durham, DH1 3LE, UK
noura.al-moubayed@durham.ac.uk

Dagou A. Zeze
Department of Engineering
Durham University
Durham, DH1 3LE, UK
d.a.zeze@durham.ac.uk

Chris Groves
Department of Engineering
Durham University
Durham, DH1 3LE, UK
chris.groves@durham.ac.uk

***Abstract*—Evolution-in-Materio (EiM) is an unconventional computing paradigm, which uses an Evolutionary Algorithm (EA) to configure a material's parameters so that it can perform a computational task. While EiM processors show promise, slow manufacturing and physical experimentation hinder their development. Simulations based on a physical model were used to efficiently investigate three specific enhancements to EiM processors which operate as classifiers. Firstly, an adapted Differential Evolution algorithm that includes batching and a validation dataset. This allows more generational updates and a validation metric which could tune hyper-parameters. Secondly, the introduction of Binary Cross Entropy as an objective function for the EA, a continuous fitness metric with several advantages over the commonly used classification error objective function. Finally, the use of regression to quickly assess the material processor's output states and produce an optimal readout layer, a significant improvement over fixed or evolved interpretation schemes which can 'hide' the true performance of a material processor. Together these enhancements provide guidance on the production of more flexible, better performing, and robust EiM processors.**

***Index Terms*—Batching, binary cross entropy, evolution in-materio processors, evolutionary materials, evolvable processors, material kernel**

I. Introduction

Challenges to the development and further improvement of traditional CMOS technology [1] has led to a growing interest in unconventional computing methods. Evolution in-Materio (EiM) is one such approach which seeks to exploit a material's physical properties to perform a computational task. EiM is inspired by the complex functions that simple nucleotides can perform when configured by evolution into a genome [2]. The electronic capability of an EiM processor is discovered through the use of an Evolutionary Algorithm (EA) which seeks to improve the performance of the system. This EA attempts to harness the innate computational ability present in the material processor using only a few configurable parameters, thereby conforming to the complexity engineering approach [3]. EiM processors have been used to achieve a range of applications such as logic gates [4]–[7] and classification [8]–[10]. While EiM processors show promise, they still require significant development before any mainstream adoption. Recent work has addressed some fundamental questions about the effect that simple algorithm and material changes have on classification performance [11], and establishes a framework to perform EiM. In this paper, we present three techniques which can be applied to EiM processors to leverage significantly better performance for classification.

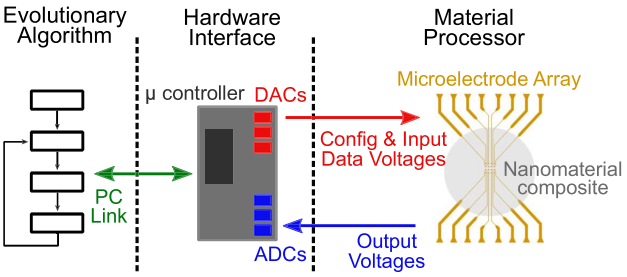

Fig. 1. Schematic diagram of a generic physical EiM processor.

EiM processors are generally formed from three constituent parts: (i) a material whose characteristics can be altered via external stimuli, (ii) a hardware interface which can apply inputs and read outputs from the material, and (iii) a device which can host and execute an EA to optimise the material. Materials are generally selected for their configurability and complexity, such as metallic nanoparticles [7], [12], liquid crystals [13], [14], single walled carbon nanotubes [5], [9], and dopant networks [6], [15]; all of which are material networks that can be configured by the application of voltages. As such, EiM processors are generally fabricated by depositing the chosen nanomaterial on a microelectrode array, which is used to apply and measure voltages, similarly to the design depicted in Fig. 1. The work presented in this paper focuses on three enhancements to the EiM paradigm, specifically relating to how the EA exploits the material processor. We now discuss these three contributions in turn.

EiM systems are limited by the available memory provided by the microprocessors that control the hardware interface which interacts with the nanomaterial electrode array. Examples from previous work include an Mbed with 32KB of RAM [10] or the MECOBO with 128KB RAM [16]. As execution speeds within in-materio processors improve, possibly up to 100MHz [6] or more, it becomes increasingly important to

978-1-6654-2332-8/21/$31.00 ©2021 IEEE
DOI 10.1109/ICRC53822.2021.00026

make efficient use of the memory available, to avoid unnecessary waiting and keep the (in-materio) processing unit running at full capacity. Similarly, Artificial Neural Network (ANN) systems often have limited local fast GPU memory preventing large datasets being executed all at once. Instead, batching (or mini-batching) is commonly used to split up the data into smaller groups or 'batches' [17], this has several benefits including a smaller memory footprint, more network updates and helping to prevent the model from over fitting. Our first contribution is the examination of the effect of introducing batching to a Differential Evolution EA, which to the best of our knowledge is used to exploit EiM for the first time.

EiM processors for classification are commonly achieved via the readout and interpretation of physical voltages. These signals will be subject to noise from the hardware, power supplies etc., meaning the placement of the EiM classifiers decision boundary is of great importance. Some recent work introduced a confidence measure for a carbon-nanotube/liquid crystal classifier [18] relating the physical output signals with a Figure of Merit. These results suggest that the 'depth' of an assigned data instance into a particular class from the physical classifiers decision boundary contains useful information. Our second contribution is the proposal that incorporating this Figure of Merit into the EA's objective function will lead to EiM systems with superior decision boundary placement compared to systems operating with the commonly implemented classification error [10], [18], [19]. To this end, we investigate the adaptation of Binary Cross Entropy (BCE) as a modified objective function for an EiM's exploiting EA for the first time.

Ensuring that a material is consistently and successfully exploited for EiM has proved challenging. Previous work has shown that if a material's outputs are inappropriately interpreted the performance of an EiM processor may be hidden [11]. However, materials with monotonically increasing current-voltage (IV) characteristics often require some interpretation process of combination and/or comparison to solve complex classification problems. Other materials, such as those with IV characteristics containing negative differential resistance (NDR) [6], [7], can achieve more complex output states which may require less complex interpretation. EiM processors have therefore traditionally contained some evolvability in the output interpretation, e.g. output weights [11], [19] or evolvable thresholds [10]. Our final contribution is to enhance the assessment and exploitation of the material. We investigate the introduction of a regression step into the EA algorithm, used to produce a readout layer for the material. This results in an optimised output layer every time a population member is tested, a significant improvement compared to evolving the output layer. It should be noted that the use of regression brings the EiM computational paradigm one step closer to Reservoir Computing (RC) [20]. Indeed, by adapting an EiM processor to temporal data, it could be classed as physical RC with an evolvable reservoir.

The paper first outlines the background and basic operation of EiM processors for classification in §II. This is followed by the implementation of the three proposed enhancements in §III, namely the introduction of batching, BCE as an objective function, and a regression step. The testing methods, simulated material networks and considered datasets are outlined in §IV. The results of the three proposed enhancements are discussed in §V. Finally, we summarise the paper and conclude in §VI.

II. Background

EiM exploits nanomaterials using an optimisation algorithm such that they can perform useful tasks. Since in-Materio processors are analogue and generally lack an analytical model, population-based derivative-free optimisation algorithms are used, rather than gradient based algorithms [9]. EAs are a subset of evolutionary computing, consisting of population-based metaheuristic search algorithms, making them ideal for EiM. EAs take advantage of biologically inspired operations such as reproduction, mutation, recombination and natural selection [21]. Many types of EAs have been used for EiM such as Evolutionary Strategies [19], Genetic Algorithms [7], Differential Evolution [9], [10] or Particle Swarm Optimisation [22], [23]. In particular, Differential Evolution (DE) is an easily implemented and effective optimisation algorithm for exploiting EiM processors. DE is a derivative-free, stochastic, population-base, heuristic direct search method [21], [24] which only requires a few robust control variables [25] and is attractive for real parameter optimisation [26].

In this paper, we combine a DE algorithm with a material simulation (developed in [11]) which allows significantly faster testing and analysis of EiM processors than physical manufacturing and experimentations would allow. Full details are available elsewhere [24], [26], but briefly, the DE algorithm uses the greedy criterion that involves evaluating the fitness of each member of a generation's population, with those members of the population with better fitness being more likely to proceed to the next generation. The characteristics of the population thus change gradually over time due to the random mutation of characteristics and cross-over with other population members. Every member of the population is represented by a vector of decision variables $\mathbf{X}$. This decision vector contains configuration parameters which the EA optimises each generation. In this paper the decision vector is defined as:

$$\mathbf{X} = [V_{c1}\ V_{c2}\ ...\ V_{cP}\ G_{sh}\ l_1\ l_2\ ...\ l_R\ m_1\ m_2\ ...\ m_Q]^T \quad (1)$$

where T is the vector transpose. The included configuration parameters are as follows [11]: Input "configuration" voltage stimuli V_{cp} applied to a node p, where the total number of configuration nodes is P. The shuffle gene G_{sh} which allows for reassignment of input electrodes to access different inter-node arrangements. Input weights $l_r \in [-1, 1]$ that scale the input voltages V_r^{in} applied at the data driven input electrodes r due to a corresponding input attribute a_r, such that:

$$V_r^{in}(k) = l_r \times a_r(k), \quad (2)$$

where k is a given data instance and the total number of data driven input electrodes is R. Output weights $m_q \in [-2, 2]$

for each output electrode q, which allows flexibility in how the network combines a particular data instances' resulting material outputs into a single overall network response Y, as follows:

$$Y(k) = \sum_{q}^{Q} m_q V_q^{out}(k), \tag{3}$$

where Q is the total number of output nodes and V_q^{out} is the material's output voltage. The generated network response Y is then used to designate a class label to the processed data instance using a simple threshold:

$$\widehat{class}(k) = \begin{cases} 2, & \text{if } Y(k) \geq 0 \\ 1, & \text{if } Y(k) < 0 \end{cases}, \tag{4}$$

where $\widehat{class}$ is the predicted class label. The operation of the EiM processor is shown in Fig. 2, illustrating how the system is affected by the configuration parameters.

During each generation, every member of the population is evaluated using the training data and an associated fitness is calculated using the EA's objective function. Commonly, in EiM systems, the objective function Φ is defined as the mean error of a dataset (i.e., the classification error):

$$\Phi = \frac{1}{K} \sum_{k=1}^{K} e(k), \tag{5}$$

where k is an instance within a dataset of total length of K.

Each data input instance produces an error value $e(k)$ of 0 or 1 for correct or incorrect classification, respectively. Generally, the dataset D, containing R attributes $a_1, a_2, \ldots, a_R$, is split into two subsets: a training set D^{train} and a test set D^{test}. The training set is used to evaluate and update the population during the evolutionary optimisation, and a test set is used to evaluate the final best population member p_{best}. The pseudocode describing the DE optimisation process is presented in Algorithm 1, where $f(pop, Dataset)$ is a fitness function

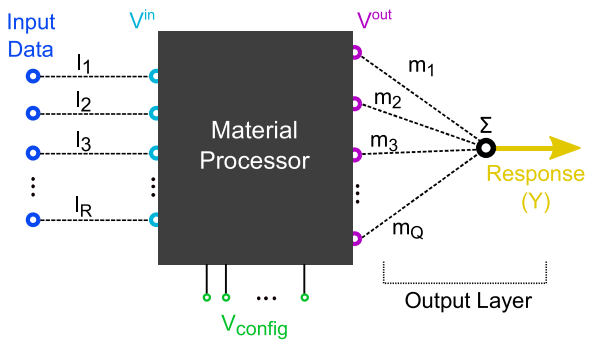

Fig. 2. Illustration of an EiM processor structure. Input data is applied to the material as voltages. The output voltages are summed to generate an overall response (Y) which is used to determine the class. If enabled input weights (l_r) and output weights (m_q) are applied. A shuffle gene can re-arrange the application of the input nodes (both input data and configuration nodes)

(e.g., (5)) evaluating member(s) of a particular population on a data subset, and $BestFitness(pop, Dataset)$ returns the population member with the best fitness on selected data subset.

Algorithm 1:

Pseudocode for DE based EiM.

```
Initialise the population p;
Evaluate initial population f(p, D^train);
p_best = BestFitness(p, D^train);
epoch = 0;
while epoch < MaxNumberEpochs do
    Generate trial population t;
    Evaluate trial population f(t, D^train);
    Update population p with respect to t;
    p_best = BestFitness(p, D^train);
    epoch = epoch + 1;
end
Evaluate p_best using the test data f(p_best, D^test);
```

III. Model Development

A. Batching

Batching (or mini-batching) is a technique used in ANNs [17], [27], which involves dividing up the training data into smaller groups known as batches. The use of small batch sizes leads to a significantly smaller memory footprint, and in the case of ANNs is found to improve generalisation performance [17]. Each batch is fed into the ANN and used to update its parameters. When all the batches (and therefore all the training data) has been used, then we say that a single epoch has occurred. Previous work has introduced batching into a DE algorithm for Neural Network Optimisation [28]. Using similar principles, we implement batching within DE for an EiM processor as follows: The dataset D is split into three subsets: a training set D^{train}, a validation set D^{valid}, and a test set D^{test}. The training set is split equally into N balanced batches $B_0, B_1, \ldots, B_{N-1}$. These batches are used sequentially to train the population, where each batch results in a single generational update. The validation set is used to provide a uniform evaluation of the population at the end of an epoch and maintain the best 'global' member. If the population has produced a member which yields a better validation fitness than previously, that member is selected as the new global best member G_{best}. While not implemented here, we note that validation fitness could also be used to tune the system's hyper-parameters. The test set is unseen data used to evaluate the global best member once some termination criteria is met.

The pseudocode describing the DE optimisation process is presented in Algorithm 2. Here we must clarify that to enable a reliable comparison between the parent/previous generation population (which was evaluated using a batch B_{n-1}) and a new trial/child population (evaluated using the current batch B_n), it is necessary to compare the trial fitness to a re-evaluated parent fitness (using the current batch B_n). Therefore, by using batching we are doubling the number of training data computations being carried out. We define a special case when $N = 1$, where there is only one batch which

is the entire training set D^{train}. In this case, there is no need to re-evaluate the parent population, and the new algorithm is differentiated from the original Algorithm 1 by its use of the validation set.

Algorithm 2:

Pseudocode for DE based EiM with batching.

```
Initialise the population p;
Generate N batches B_0, B_1, ..., B_(N-1);
Evaluate initial population f(p, B_0);
G_best = ∞;
epoch = 0;
n = 1;
while epoch < MaxNumberEpochs do
    if N > 1 then
        Evaluate population p on new batch f(p, B_n);
    end
    Generate trial population t;
    Evaluate trial population f(t, B_n);
    Update population p with respect to t;
    p_best = BestFitness(p, B_n);
    if n ≥ N − 1 then
        if BestFitness(p, D^valid) ≤ G_best then
            G_best = BestFitness(p, D^valid);
        end
        n = 0;
        epoch = epoch + 1;
    else
        n = n + 1;
    end
end
Evaluate G_best using the test data f(G_best, D^test);
```

B. Cross Entropy Loss

As mentioned in the introduction, classification error is a commonly used objective function for EiM processors which evaluates a data input instance by assigning an error value $e(k)$ of 0 or 1 for correct or incorrect classification respectively. While this is a flexible approach shown to work for many EiM systems, it provides a discrete fitness evaluation where members of the same fitness cannot be differentiated. Considering a very simple, linearly separable problem, it is easy to see how a decision boundary may achieve a zero classification error, but still have varying qualities in operation due to the inevitable presence of noise, as shown in Fig. 3. Instead, by introducing a continuous metric associated to the distance of the data from the decision boundary, similar to the Figure of Merit or the 'confidence' that it is within the correct class [18], a more consistently favourable decision boundary should be achieved, as shown in Fig. 3b.

Binary Cross Entropy (BCE) or log loss is an established loss function for machine learning binary classification tasks. Cross entropy generates larger loss values as the predicted probability of a label diverges from the value of the actual label. To adapt BCE as an objective function for EiM systems, the raw output of the EiM processor must be first constricted

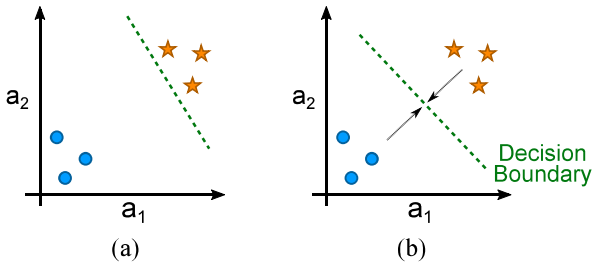

Fig. 3. Example decision boundary for the (a) discrete classification error objective function where evolved systems with 100% accuracy may be susceptible to noise, and (b) Binary Cross Entropy objective function which uses information from the classified data to maximise the likelihood of successful classification.

to $\in [0, 1]$. To do this a sigmoid function $\sigma(k)$ is used such that for a particular input data instance k:

$$\sigma(k) = \frac{1}{1 + e^{-Y(k)}}, \tag{6}$$

where $Y(k)$ is the network response of the system defined in (3). The entropy or log loss $H(k)$ is defined as:

$$H(k) = \begin{cases} -\ln(1 - \sigma(k)), & \text{if } class(k) = 1 \\ -\ln(\sigma(k)), & \text{if } class(k) = 2 \end{cases}, \tag{7}$$

where $\ln()$ is the natural logarithm and $class(k)$ is the actual label associated with the data instance. Therefore, the adapted BCE objective function that the system attempts to minimise is defined as:

$$\Phi^{BCE} = \frac{1}{K} \sum_{k=1}^{K} H(k). \tag{8}$$

This new cross entropy based objective function is designed to penalise classified instances which are both confident (i.e., far from the decision boundary) and wrong, but reward classified instances which are both confident and right, as shown in Fig. 4

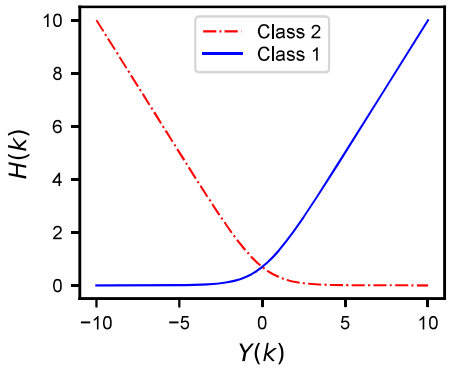

Fig. 4. Entropy generated by the EiM processor's output response.

C. Regression Generated Output Layer

Recent work has shown that the use of output weights is important in EiM processors [11] allowing a material's output states to be combined and produce more complex decision boundaries which can significantly improve the results for classification problems. This suggests that the interpretation scheme used to extract information from an EiM material's outputs is of great importance, and if one such scheme is inappropriately chosen it may "hide" the true computational performance possessed by the material.

As discussed in the introduction, interpretation schemes for EiM processors have typically been fixed, or contain evolvable parameters such as classification thresholds or output weights. Here, we consider the use of ridged regression to efficiently generate and optimise a readout (i.e., output) layer. There are two notable changes to the algorithm brought about by the introduction of regression: (i) a readout layer is generated during the evaluation of a population member in the training phase, and (ii) the regression will have its own separate loss function which does not necessarily align with the EA objective function. The generated readout layers must be stored and updated in a vector, which corresponds to the current population, so that the best decision vector and readout layer combination is tracked and may be recalled when evaluated on the test (or validation) data set.

IV. Testing Methodology

A. Material Simulation & Algorithm Evaluation

This paper focuses on investigating algorithm adaptations which boost the performance of EiM processors. To this end, the simulated material was selected such that it would not be the limiting factor. Here we focus our attention on a simulated Diode Random Network (DRN) material processor [11]. The DRN is based on randomly interconnected diode and resistor networks, it acts as an exemplar of a highly non-linear material. These networks contain either voltage driven input nodes, or measured output voltage nodes, calculated using a SPICE DC analysis. The DRN consists of a diode of random orientation is in series with a randomly selected resistor between every node pair. The rapid changes in conductivity when a diode is turned on allows for a flexible and non-linear classification decision boundary. The DRN is physically realisable using discrete circuit components and its properties are common in nanomaterials. Therefore, the DRN is a reasonable representation of an EiM material with good complexity which will not limit the performance and hinder the investigation of the proposed algorithm adaptations. Other IV characteristics are possible [11] but these are outside the scope of the current paper. An example of a five node (i.e., electrode) material is given in Fig. 5.

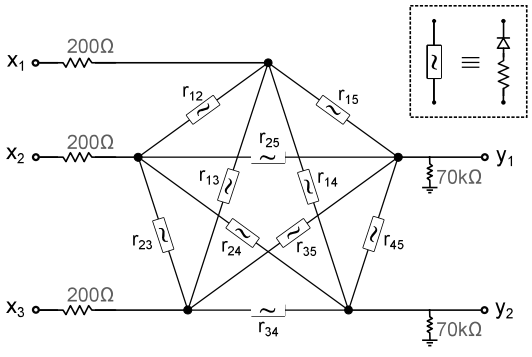

Fig. 5. An example five node DRN material, where each node is connected to every other node via a resistor and diode. In this example, two nodes are behaving as outputs (y_1, y_2) and three nodes as inputs (x_1, x_2, x_3) which could be allocated as either data driven or evolvable configuration voltages.

To exploit the material as an EiM processor and perform classification, DE is used to optimise the vector of decision variables (i.e., (1)). In this work, a DE/best/1/bin algorithm is used [26] with a mutation factor of $F = 0.8$, crossover rate of $CR = 0.8$, and population of 20. The DE algorithm was then altered to include batching, BCE and the intermediate regression step.

To ensure a fair comparison between the proposed algorithms, fifteen DRN materials were generated, each with twelve nodes which correspond to electrodes in a physical system, and these were used in all the experiments. Therefore, differences in performance can only be attributed to the algorithm exploiting the materials. However, EAs are stochastic in nature which leads to variations in convergence speed. To mitigate this randomness during experimentation, the algorithm under consideration is repeated five times on each of the fifteen materials. The results from these seventy-five executions are used to evaluate the proposed algorithms.

In this paper we consider datasets containing different numbers of attributes. It is important that the material electrodes (i.e., simulated material nodes) are appropriately assigned. Assuming the selected dataset D contains R attributes, then R nodes were allocated as input nodes. To help ensure exploitability, the inputs are then projected to higher dimensional outputs. Here, $R+1$ nodes were allocated as output nodes. The remaining nodes are allocated as configuration voltage input stimuli, as seen in Fig. 2.

B. Datasets

To assess the performance of the EiM classifiers a pair of two attribute binary datasets was generated. The first is a simple linearly separable 2D dataset (2DDS), containing one thousand data instances with two attributes, a_1 and a_2, and two classes, 1 and 2, as seen in Fig. 6a. Similarly, the second dataset contains a thousand data instances, but generated concentrically, as seen in Fig. 6b, and will be referred to here as the c2DDS. The c2DDS is a more challenging dataset, requiring the algorithm to exploit non-linearities within the material processors to produce an enclosed decision boundary. Lastly, the Banknote UCI dataset [29] is also used in this work as an example of a real world, four attribute binary classification problem containing 1372 data instances.

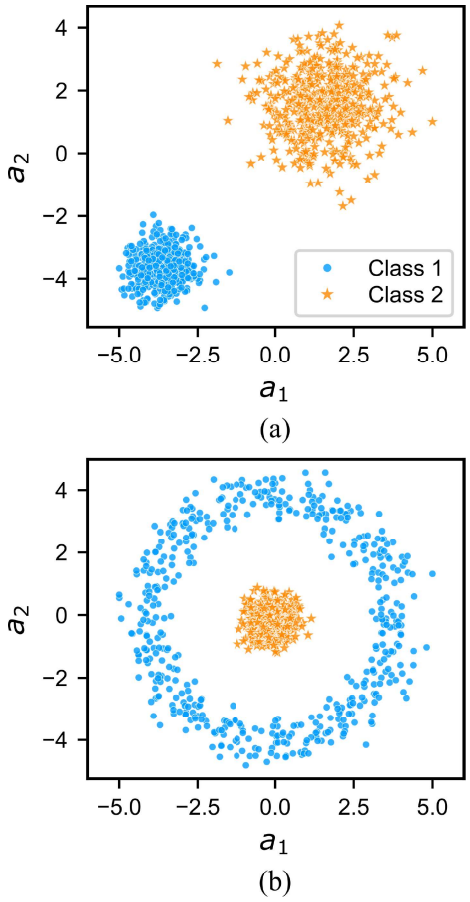

Fig. 6. The randomly generated (a) 2DDS and (b) c2DDS datasets.

V. Results and Discussion

A. Batching

Algorithm 2 (i.e., the batching algorithm) needs to recompute the parent fitness on each new batch and therefore processes double the amount of training data per epoch than Algorithm 1 (i.e., the original algorithm) or the special case when the number of batches $N = 1$. Recalling how the data is split into the subsets for the different algorithms (§IV-B), the number of computations per epoch n_e for Algorithm 1 is:

$$n_e = 0.8n_d, \tag{9}$$

where n_d is the number of data instances in the selected dataset D. For Algorithm 2, when $N > 1$, the number of computations per epoch $n_e^{batching}$ is:

$$n_e^{batching} = 2 \times \underbrace{0.6n_d}_{train} + \underbrace{0.2n_d}_{valid} = 1.4n_d. \tag{10}$$

Therefore, if the algorithms were compared for an identical number of epochs, Algorithm 2 would process significantly more data in the allotted evolutionary period. Instead, the systems were evolved until $8{\times}10^5$ data instances had been processed, at which point the EA terminated.

The more challenging c2DDS and BankNote datasets were considered and used to compare the Algorithm 1 with the special case $N = 1$ and various other batch sizes used for Algorithm 2. The evolution of the mean best population member (p_{best}) training fitness is shown in Fig. 7a & 7c for the c2DDS and BankNote datasets respectively. Fig. 7b & 7d show the corresponding change in mean validation accuracy of Algorithm 2's global best member (G_{best}), or in the case of Algorithm 1 simply the mean training accuracy of the p_{best}.

Algorithm 2, with $N = 1$, shows comparable results to the original Algorithm 1. As the number of batches N was increased, the EiM processors converged substantially faster, and a higher validation and final test fitness is achieved. Significantly, the introduction of batching leads to an increased number of generations per epoch. The number of computations (i.e., number of data instances processed) used for a single generational update γ for the original Algorithm 1 is:

$$\gamma = 0.8n_d, \tag{11}$$

and the number of computations per generation for the batching Algorithm 2 (when $N > 1$) is on average:

$$\gamma^{batching} = \frac{1.4n_d}{N}. \tag{12}$$

The large increase in performance is attributed to this faster rate of useful generational updates. This should allow physical EiM devices to be trained more quickly and be more computationally efficient. However, if the batch size is too small (e.g., batches containing six data instances or fewer), not enough information from the dataset D will be present, leading to wild variation in training population fitness. This can result in poor population updates and a lack of EA convergence to an acceptable solution.

B. Cross Entropy Loss

The EiM processors were used to classify the 2DDS using the classification error objective function (5), over 40 epochs, with Algorithm 2 and $N = 1$. This process was then repeated using the BCE objective function (8). A histogram of the processor output response Y generated by the test data from all the repetitions is plotted in Fig. 8a. The error evolved system produced output responses much closer to the decision boundary (i.e., $Y = 0$). The BCE evolved systems generate output responses much further away from the decision boundary, meaning these EiM processors are placing boundaries between the data much more successfully.

The BCE objective function requires the output layer (3) to evolve and exploit the sigmoid (6) to learn which data instances are considered 'deep' or not. Therefore, insufficient flexibility within the output layer (such as the maximum and minimum output weight), could limit the final fitness an EiM processor could achieve. We also note that Algorithm 2 grants the system design more flexibility, with the opportunity to select different objective functions for the training and validation evaluations. However, in this work the selected objective function was used for both training and validation fitness scores.

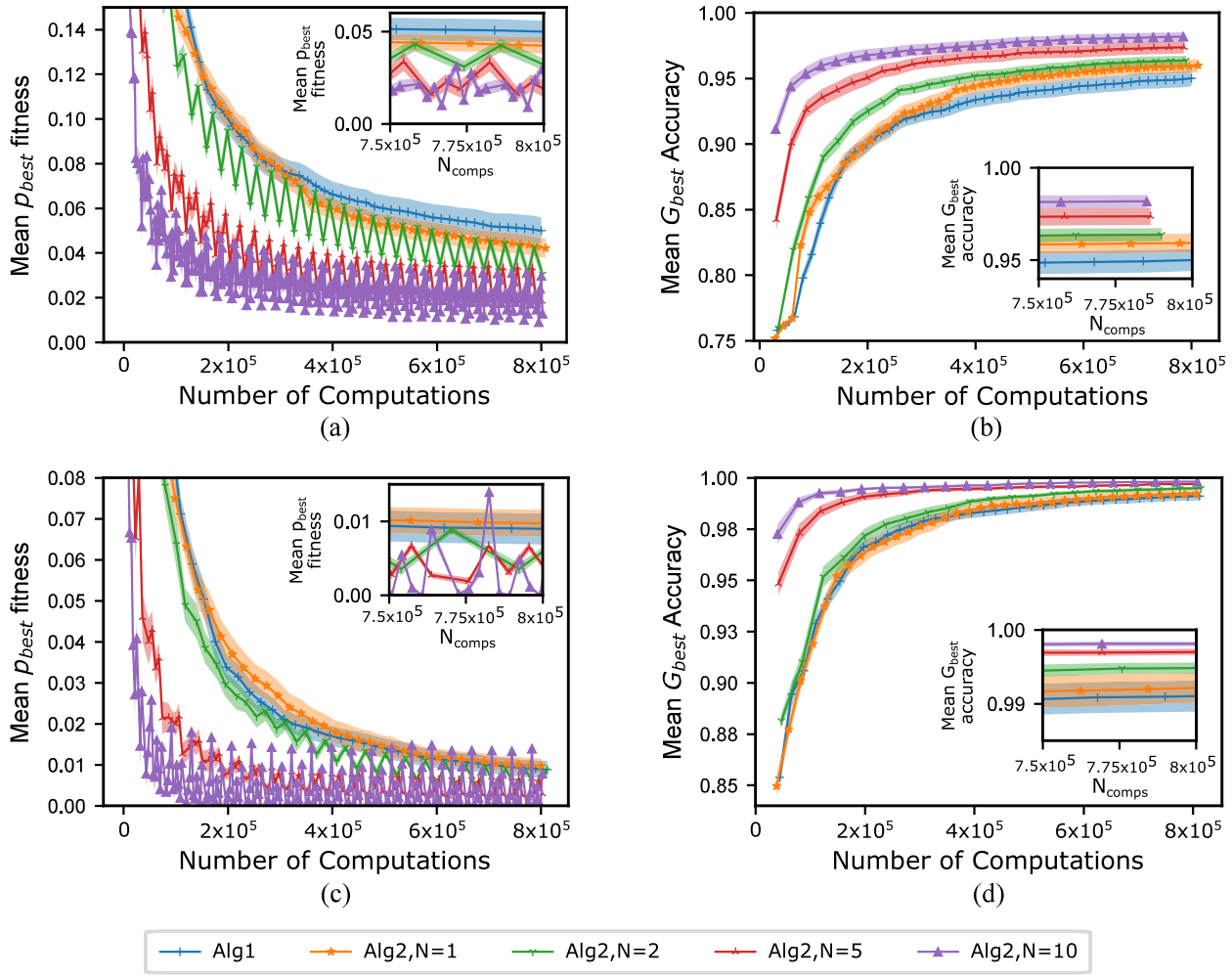

Fig. 7. Evolution of the (a) mean best population member training fitness and (b) mean global best member validation accuracy, with standard error, comparing the different batch sizes on the c2DDS. This was repeated to examine the evolution of the (c) mean best population member training fitness and (d) mean global best member validation accuracy, with standard error, comparing the different batch sizes on the Banknote dataset. Inset graphs show the convergence during the final 0.5×10^5 computations, to more easily distinguish between the approaches.

To examine the different systems' resilience to noise, the test data is combined with varying levels of Gaussian noise $X \sim N(0, \sigma^2)$ and then evaluated on the trained systems. The same set of random seeds is used such that a fair comparison between the systems can be made. The mean test accuracy is plotted against a variation of σ, as seen in Fig. 8b. As expected, the BCE evolved system is significantly more resilient to the input noise due to the decision boundary being placed in a location which provides a better probabilistic assignment of class. While the addition of input noise has been used to illustrate the superior quality of boundary placement achieved by the BCE objective function, this is still considering a system evolved under ideal conditions. Further work, with simulations accounting for more realistic sources of noise, would help establish the importance and effect of noise during the training & operation of physical EiM processors.

C. Regressed EiM

Standard EiM processors using an output layer defined and evolved in the decision vector $\mathbf{X}$ were used to solve the c2DDS and Banknote dataset. Algorithm 2 was used with $N = 1$ for 60 epochs, using the classification error objective function. The change in the mean validation accuracy of the systems' global best member is tracked in Fig. 9a and Fig. 9b for the c2DDS and Banknote dataset respectively. The EiM processors were then re-trained, but with ridge regression as an intermediate step to generate the output layer, referred to here as Regressed EiM. To ensure a fair comparison, ridge regression was used without fitting an intercept, so only a readout layer using output weights (and no bias) is generated. After just one epoch, the Regressed EiM processors achieved a 23.9% and 14.7% better mean validation accuracy for the c2DDS and Banknote datasets respectively. The performance of the

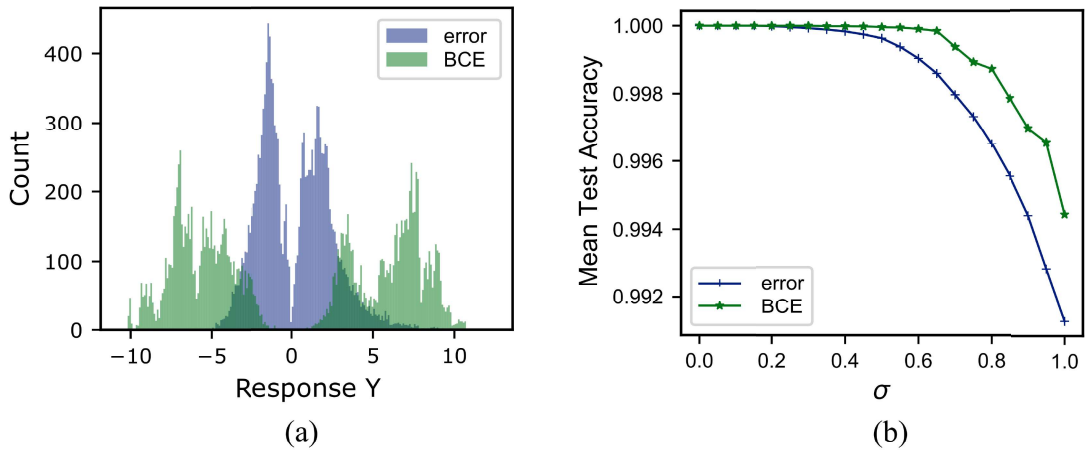

Fig. 8. (a) Histogram of the test data responses for all material and algorithm repetitions trained on the 2DDS. (b) The effect of increasing the standard deviation of Gaussian noise (added to the test data) on the mean test accuracy of the differently trained EiM systems.

Standard EiM processors is clearly tied to the interpretation scheme, in this case dependant on an output layer defined by the output weights contained in the decision vector. Possibly well performing configured materials are being hidden by poorly performing output weights. By contrast, the use of ridge regression allows the output layer to be independently optimised, and therefore ensure that material outputs are exploited fully for the target problem.

As the evolutionary period progresses the mean accuracy of the Standard EiM processors rapidly increases. This is due to the optimisation of both the material configuration and the interpretation scheme. However, in the case of Regressed EiM, the performance gains are only driven by an improvement of the material configuration. After 60 epochs, the mean validation accuracy of the Regressed EiM processors increased by 7.21% and 1.84% for the c2DDS and Banknote dataset respectively. For these simulated material processors, only small gains in fitness, and therefore accuracy, are made during the evolutionary process. However, we expect that the benefits of optimising the material's configuration parameters and stimuli will be closely related to the computational problem and material processor's properties. Indeed, an EiM processors' voltage stimuli, input weights and electrode allocation have been shown to provide distinct (if limited) benefits [11].

Ridge regression does not impose limits on the output weights, as is necessary for the DE algorithm's decision vector. Ridge regression can also be used with a fitted intercept (i.e., a bias), which further improved performance. The introduction of a regression stage allows for the efficient generation of an exploiting readout layer, without the need to optimise output configuration parameters within the slower EA process. We propose that the Regressed EiM presents a much better representation of the true capabilities of the material processor, and significantly improves the speed of convergence to an acceptable classification accuracy. These benefits should translate directly to practical examples of EiM systems, such as SWCNT networks [4], [5], [9]. The regression optimised output weights are not limited, and this flexibility may also allow smaller more subtle differences in a nanomaterials output voltages to be identified and exploited.

The test data results are reported in Table 1 and suggest that the Standard EiM processors overfitted slightly for the Banknote dataset (Fig. 9b). Here, the EiM processors are also compared to the application of logistic and ridge regression on the raw data, which both the Standard EiM and Regressed EiM approaches outperform. The material is acting similarly to a kernel, transforming the input data into a useful, higher dimensional representation. This is very similar to the behaviour of reservoirs in RC [20]. Indeed, if an EiM processor was extended to process temporal data, then it could be described as a physical RC with an evolvable reservoir. From the perspective of conventional computing, nanomaterials often contain undesirable temporal properties such as hysteresis, charge leakage, etc. Within the EiM computing paradigm, these properties can be exploited to produce unconventional processors. We hypothesise that extending the EiM paradigm to RC will lead to more flexible reservoirs and performance gains, warranting further research.

TABLE I
TEST RESULTS FOR THE EVOLVED AND REGRESSED EIM.

Dataset	Processor Type	mean(Φ)	std(Φ)	Best Φ
c2DDS	Standard EiM	0.055	0.050	**0.000**
	Regressed EiM	0.004	0.017	**0.000**
	Ridge Regression	-	-	0.490
	Logistic Regression	-	-	0.490
Banknote	Standard EiM	0.009	0.010	**0.000**
	Regressed EiM	0.004	0.002	**0.000**
	Ridge Regression	-	-	0.018
	Logistic Regression	-	-	0.015

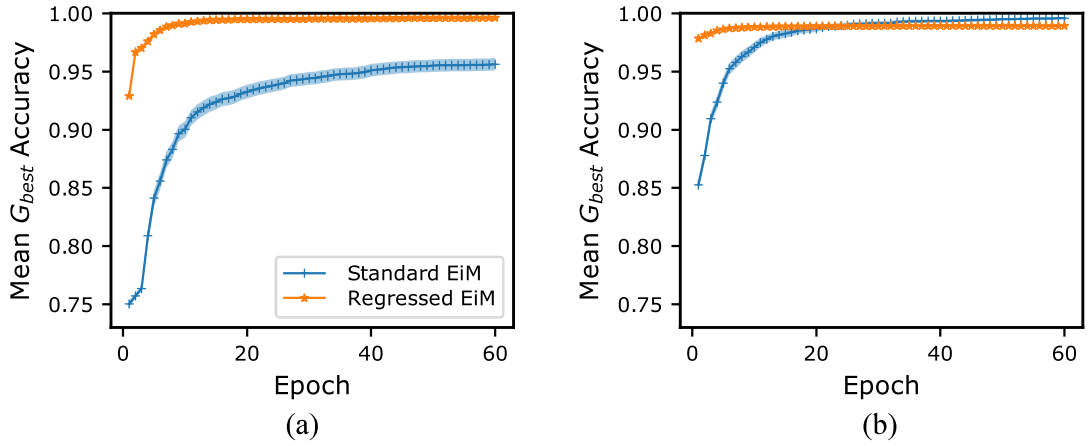

Fig. 9. Evolution of the mean global best member validation accuracy, with standard error, comparing the Standard and Regressed EiM methods on the (a) c2DDS and (b) Banknote dataset.

VI. Conclusion

Evolution in-Materio (EiM) processors are a promising unconventional computing paradigm where materials can be configured to perform computational tasks. However, investigating the EiM computational framework is challenging on physical systems due to slow fabrication and testing. Here, a simulation of a physically realisable circuit is paired with Differential Evolution (DE) to produce an EiM processor. This is used to efficiently experiment and explore improvements to the EiM framework and algorithm.

The DE algorithm was adapted to enable batching of the training data. This involved the introduction of a validation step at the end of an epoch to provide a uniform evaluation of the population. Smaller batch sizes were found to converge more quickly to a final solution due to the reduced number of computations needed per generational update. However, extremely small batch sizes were found to cause poor generational updates as not enough information from the dataset was represented.

Binary Cross Entropy (BCE) was introduced to replace the commonly used classification error objective function, and was shown to generate more noise resistant EiM classifiers. BCE is a continuous fitness (i.e., loss) metric which uses information from the classified data instance's distance from the decision boundary. This ensures that a successful comparison between a trial population member and its parent will always be possible, leading to more reliable convergence, and also superior decision boundary placement.

Regression was used as an additional intermediate step to train the output (i.e., readout) layer of the material processor. This replaced the evolution of output weights, previously defined in the DE decision vector. These generated output layers needed to correspond to, and be maintained alongside, the DE's population of solutions. Using ridged regression, this technique was found to far outperform the standard EiM algorithm. This highlights that slow to evolve or inappropriate interpretation schemes can "hide" the performance of well configured materials.

EiM processors have the potential to harness complex nanomaterial properties to produce efficient unconventional computing devices. This work shows how the traditional EiM algorithm can be adapted to produce more robust and better performing systems, ensuring physical nanomaterial processors are fully exploited. These enhancements bring EiM processors one step closer to future real-world applications.

References

[1] T. M. Conte, E. P. DeBenedictis, P. A. Gargini, and E. Track, "Rebooting Computing: The Road Ahead," *Computer*, vol. 50, no. 1, pp. 20–29, Jan. 2017.

[2] J. Miller and K. Downing, "Evolution in materio: Looking beyond the silicon box," in *Proceedings 2002 NASA/DoD Conference on Evolvable Hardware*. Alexandria, VA, USA: IEEE Comput. Soc, 2002, pp. 167–176.

[3] N. Ganesh, "Rebooting Neuromorphic Hardware Design – A Complexity Engineering Approach," *arXiv:2005.00522 [cs]*, Sep. 2020.

[4] A. Kotsialos, M. K. Massey, F. Qaiser, D. A. Zeze, C. Pearson, and M. C. Petty, "Logic gate and circuit training on randomly dispersed carbon nanotubes." *International journal of unconventional computing.*, vol. 10, no. 5-6, pp. 473–497, Sep. 2014.

[5] M. K. Massey, A. Kotsialos, F. Qaiser, D. A. Zeze, C. Pearson, D. Volpati, L. Bowen, and M. C. Petty, "Computing with carbon nanotubes: Optimization of threshold logic gates using disordered nanotube/polymer composites," *Journal of Applied Physics*, vol. 117, no. 13, p. 134903, Apr. 2015.

[6] T. Chen, J. van Gelder, B. van de Ven, S. V. Amitonov, B. de Wilde, H.-C. R. Euler, H. Broersma, P. A. Bobbert, F. A. Zwanenburg, and W. G. van der Wiel, "Classification with a disordered dopant-atom network in silicon," *Nature*, vol. 577, no. 7790, pp. 341–345, Jan. 2020.

[7] S. K. Bose, C. P. Lawrence, Z. Liu, K. S. Makarenko, R. M. J. van Damme, H. J. Broersma, and W. G. van der Wiel, "Evolution of a designless nanoparticle network into reconfigurable Boolean logic," *Nature Nanotechnology*, vol. 10, no. 12, pp. 1048–1052, Dec. 2015.

[8] K. D. Clegg, J. F. Miller, M. K. Massey, and M. C. Petty, "Practical issues for configuring carbon nanotube composite materials for computation," in *2014 IEEE International Conference on Evolvable Systems*, Dec. 2014, pp. 61–68.

[9] M. K. Massey, A. Kotsialos, D. Volpati, E. Vissol-Gaudin, C. Pearson, L. Bowen, B. Obara, D. A. Zeze, C. Groves, and M. C. Petty, "Evolution of Electronic Circuits using Carbon Nanotube Composites," *Scientific Reports*, vol. 6, no. 1, p. 32197, Oct. 2016.

[10] E. Vissol-Gaudin, A. Kotsialos, C. Groves, C. Pearson, D. Zeze, and M. Petty, "Computing Based on Material Training: Application to Binary Classification Problems," in *2017 IEEE International Conference on Rebooting Computing (ICRC)*. Washington, DC: IEEE, Nov. 2017, pp. 1–8.

[11] Benedict Jones, D. A. Zeze, and C. Groves, "Towards intelligently designed Evolvable Nanomaterial Processors," *Unpublished*, 2021.

[12] K. Greff, R. M. J. van Damme, J. Koutnik, H. J. Broersma, J. O. Mikhal, C. P. Lawrence, W. G. van der Wiel, and J. Schmidhuber, "Using neural networks to predict the functionality of reconfigurable nano-material networks," in *International Journal on Advances in Intelligent Systems*, vol. 9. IARIA, Jan. 2017, pp. 339–351.

[13] S. Harding and J. Miller, "Evolution in materio: A tone discriminator in liquid crystal," in *Proceedings of the 2004 Congress on Evolutionary Computation (IEEE Cat. No.04TH8753)*, vol. 2, Jun. 2004, pp. 1800–1807 Vol.2.

[14] S. Harding and J. F. Miller, "Evolution In Materio: Evolving Logic Gates in Liquid Crystal," *International Journal of Unconventional Computing*, vol. 3, pp. 243–257, 2007.

[15] H.-C. Ruiz-Euler, U. Alegre-Ibarra, B. van de Ven, H. Broersma, P. A. Bobbert, and W. G. van der Wiel, "Dopant Network Processing Units: Towards Efficient Neural-network Emulators with High-capacity Nanoelectronic Nodes," *arXiv:2007.12371 [cs, stat]*, Jul. 2020.

[16] O. R. Lykkebø, S. Harding, G. Tufte, and J. F. Miller, "Mecobo: A Hardware and Software Platform for In Materio Evolution," in *Unconventional Computation and Natural Computation*, ser. Lecture Notes in Computer Science, O. H. Ibarra, L. Kari, and S. Kopecki, Eds. Cham: Springer International Publishing, 2014, pp. 267–279.

[17] D. Masters and C. Luschi, "Revisiting Small Batch Training for Deep Neural Networks," *arXiv:1804.07612 [cs, stat]*, Apr. 2018.

[18] E. Vissol-Gaudin, A. Kotsialos, C. Groves, C. Pearson, D. Zeze, M. Petty, and N. Al Moubayed, "Confidence Measures for Carbon-Nanotube / Liquid Crystals Classifiers," in *2018 IEEE Congress on Evolutionary Computation (CEC)*, Jul. 2018, pp. 1–8.

[19] M. Dale, S. Stepney, J. F. Miller, and M. Trefzer, "Reservoir computing in materio: A computational framework for in materio computing," in *2017 International Joint Conference on Neural Networks (IJCNN)*, May 2017, pp. 2178–2185.

[20] G. Tanaka, T. Yamane, J. B. Héroux, R. Nakane, N. Kanazawa, S. Takeda, H. Numata, D. Nakano, and A. Hirose, "Recent advances in physical reservoir computing: A review," *Neural Networks*, vol. 115, pp. 100–123, Jul. 2019.

[21] A. N. Sloss and S. Gustafson, "2019 Evolutionary Algorithms Review," *arXiv:1906.08870 [cs]*, Jun. 2019.

[22] E. Vissol-Gaudin, A. Kotsialos, M. K. Massey, D. A. Zeze, C. Pearson, C. Groves, and M. C. Petty, "Data Classification Using Carbon-Nanotubes and Evolutionary Algorithms," in *Parallel Problem Solving from Nature – PPSN XIV*, ser. Lecture Notes in Computer Science, J. Handl, E. Hart, P. R. Lewis, M. López-Ibáñez, G. Ochoa, and B. Paechter, Eds. Cham: Springer International Publishing, 2016, pp. 644–654.

[23] ——, "Training a Carbon-Nanotube/Liquid Crystal Data Classifier Using Evolutionary Algorithms," in *Unconventional Computation and Natural Computation*, ser. Lecture Notes in Computer Science, M. Amos and A. CONDON, Eds. Cham: Springer International Publishing, 2016, pp. 130–141.

[24] R. Storn and K. Price, "Differential Evolution – A Simple and Efficient Heuristic for global Optimization over Continuous Spaces," *Journal of Global Optimization*, vol. 11, no. 4, pp. 341–359, Dec. 1997.

[25] M. E. H. Pedersen, "Good Parameters for Differential Evolution," 2010.

[26] S. Das and P. N. Suganthan, "Differential Evolution: A Survey of the State-of-the-Art," *IEEE Transactions on Evolutionary Computation*, vol. 15, no. 1, pp. 4–31, Feb. 2011.

[27] Y. Bengio, "Practical recommendations for gradient-based training of deep architectures," *arXiv:1206.5533 [cs]*, Sep. 2012.

[28] M. Baioletti, G. Di Bari, A. Milani, and V. Poggioni, "Differential Evolution for Neural Networks Optimization," *Mathematics*, vol. 8, no. 1, p. 69, Jan. 2020.

[29] E. K. T. Dheeru Dua, "UCI Machine Learning Repository," http://archive.ics.uci.edu/ml, 2017.