

Durham E-Theses

Harnessing Evolution in-Materio as an Unconventional Computing Resource

JONES, BENEDICT

How to cite:

JONES, BENEDICT (2023) *Harnessing Evolution in-Materio as an Unconventional Computing Resource*, Durham theses, Durham University. Available at Durham E-Theses Online:
<http://etheses.dur.ac.uk/15119/>

Use policy

The full-text may be used and/or reproduced, and given to third parties in any format or medium, without prior permission or charge, for personal research or study, educational, or not-for-profit purposes provided that:

- a full bibliographic reference is made to the original source
- a [link](#) is made to the metadata record in Durham E-Theses
- the full-text is not changed in any way

The full-text must not be sold in any format or medium without the formal permission of the copyright holders.

Please consult the [full Durham E-Theses policy](#) for further details.

Harnessing Evolution in-Materio as an Unconventional Computing Resource

BENEDICT A. H. JONES



University College
Durham University

A Thesis presented for the degree of
Doctor of Philosophy



Durham
University

Advanced Materials and Electronic Devices Research Node
Department of Engineering
University of Durham
England

March 2023

Dedicated to

My family & friends who have supported me.

‘Journey Before Destination’

– Brandon Sanderson, *The Way of Kings*

Harnessing Evolution in-Materio as an Unconventional Computing Resource

BENEDICT A. H. JONES

Submitted for the degree of Doctor of Philosophy

March 2023

Abstract

This thesis illustrates the use and development of physical conductive analogue systems for *unconventional computing* using the Evolution in-Materio (EiM) paradigm. EiM uses an Evolutionary Algorithm to configure and exploit a physical material (or medium) for computation. While EiM processors show promise, fundamental questions and scaling issues remain. Additionally, their development is hindered by slow manufacturing and physical experimentation. This work addressed these issues by implementing simulated models to speed up research efforts, followed by investigations of physically implemented novel *in-materio* devices.

Initial work leveraged simulated conductive networks as single substrate ‘monolithic’ EiM processors, performing classification by formulating the system as an optimisation problem, solved using Differential Evolution. Different material properties and algorithm parameters were isolated and investigated; which explained the capabilities of configurable parameters and showed ideal nanomaterial choice depended upon problem complexity. Subsequently, drawing from concepts in the wider Machine Learning field, several enhancements to monolithic EiM processors were proposed and investigated. These ensured more efficient use of training data, better classification decision boundary placement, an independently optimised read-out layer, and a smoother search space. Finally, scalability and performance issues were addressed by constructing in-Materio Neural Networks (iM-NNs), where several EiM processors were stacked in parallel and operated as physical realisations of Hidden Layer neurons. Greater flexibility in system implementation was achieved by re-using a single physical substrate recursively as several *virtual* neurons, but this sacrificed faster parallelised execution. These novel iM-NNs were first implemented using Simulated in-Materio neurons, and trained for classification as Extreme Learning Machines, which were found to outperform artificial networks of a similar size. Physical iM-NN were then implemented using a Raspberry Pi, custom Hardware Interface and Lambda Diode based Physical in-Materio neurons, which were trained successfully with neuroevolution. A more complex AutoEncoder structure was then proposed and implemented physically to perform dimensionality reduction on a handwritten digits dataset, outperforming both Principal Component Analysis and artificial AutoEncoders.

This work presents an approach to exploit systems with interesting physical dynamics, and leverage them as a computational resource. Such systems could become low power, high speed, *unconventional computing* assets in the future.

Declaration

The work in this thesis is based on research carried out at the Advanced Materials and Electronic Devices Research Node, the Department of Engineering, England. No part of this thesis has been submitted elsewhere for any other degree or qualification and it is all my own work unless referenced to the contrary in the text.

Copyright © 2023 by BENEDICT A. H. JONES.

“The copyright of this thesis rests with the author. No quotations from it should be published without the author’s prior written consent and information derived from it should be acknowledged”.

Acknowledgements


I would like to thank my Supervisors Chris Groves, Dagou Zeze and Noura Al Moubayed for their help and support. They have been outstanding mentors, making time and providing guidance throughout the PhD.

I would also like to thank the staff and students I have met and worked with during my time at the Department of Engineering. In particular, thank you to Eléonore, for your support and encouragement as I was setting out on my journey. Additionally, University College has been a home away from home, and provided a little microcosm which I could escape to.

To my friends, your support has been invaluable. Our coffees, discussions, games and more have helped ground me, particularly during the odd times that were the pandemic.

Finally, I wish to thank my family. My brothers, sister & nephews who mean the world to me, my partner Katy who has supported me & my sanity, and my parents whose encouragement has gotten me to both the start and the finish line of this PhD.

Contents

Abstract	iii
Declaration	iv
Acknowledgements	v
List of Figures	xi
List of Tables	xix
List of Publications 	xxi
Acronyms	xxiv
Glossary	xxv
Nomenclature	xxvii
1 Introduction	1
1.1 Chapter Overview	1
1.2 Historical Context	2
1.3 Unconventional Computing	4
1.4 Evolution in-Materio	5
1.5 Research Hypothesis	11
1.6 Thesis Structure	12
Bibliography	13

2	Theory	21
2.1	Chapter Overview	21
2.2	Evolution in-Materio	22
2.2.1	Traditional EiM Interpretation Schemes	22
2.3	Evolutionary Algorithms	24
2.3.1	Differential Evolution	24
2.3.2	OpenAI Evolutionary Strategy	28
2.3.3	Covariance Matrix Adaptation ES	29
2.4	Objective Functions	30
2.4.1	Classification Error	31
2.4.2	Mean Squared Error	31
2.4.3	Binary Cross Entropy	32
2.4.4	Cross Entropy	32
2.5	Feed Forward Artificial Neural Networks	33
2.5.1	Artificial Neurons & Activation Functions	34
2.5.2	Optimisation Methods	35
2.6	AutoEncoders	40
2.7	Physical Reservoir Computing	42
	Bibliography	45
3	Problem Formulation	52
3.1	Chapter Overview	52
3.2	Configurable Analogue Processor	53
3.3	EiM for Classification	55
3.4	Simulated CAP Model	57
3.5	Physical Experimentation	60
3.5.1	Test Platform	60
3.5.2	Physical CAP	64
3.6	Datasets	65
3.6.1	Classification Datasets	65
3.6.2	AutoEncoder Datasets	68
	Bibliography	70

4	Monolithic EiM Devices	74
4.1	Chapter Overview	74
4.2	Algorithm & Material Interaction	76
4.2.1	Material Properties and Stimuli Voltages	76
4.2.2	Electrode Reconfiguration and Weighting	79
4.3	Advanced EiM Processor	82
4.3.1	Electrode Allocation and Material Properties	84
4.3.2	Effect of Modifying the Decision Vector	86
4.3.3	Classification Performance	89
4.3.4	Discussion	91
4.4	Summary	92
	Bibliography	93
5	Enhancements to EiM Processors	96
5.1	Chapter Overview	96
5.2	Experimental System Configuration	97
5.3	Batching	99
5.3.1	Experimental Implementation	100
5.3.2	Performance	102
5.4	Binary Cross Entropy Objective Function	105
5.4.1	Experimental Implementation	105
5.4.2	Performance	107
5.5	Regressed Output Layer	110
5.5.1	Experimental Implementation	111
5.5.2	Performance	111
5.6	Fully Connected Input Layer	114
5.6.1	Experimental Implementation	115
5.6.2	Performance	116
5.7	Summary	118
	Bibliography	119

6	In-Materio Neural Networks	122
6.1	Chapter Overview	122
6.2	In-Materio Neural Network Structure	124
6.2.1	Directly Connected Input Layer	127
6.2.2	Virtual iM-NNs	127
6.3	In-Materio Extreme Learning Machines	128
6.3.1	Experimental Implementation	129
6.3.2	Performance	132
6.4	Neuroevolution of Physical iM-NNs	135
6.4.1	Physical LDN Neuron	137
6.4.2	Experimental Implementation	138
6.4.3	Performance	141
6.4.4	Further Discussion	145
6.5	In-Materio AutoEncoders	146
6.5.1	Experimental Implementation	147
6.5.2	Performance	149
6.6	Summary	152
	Bibliography	153
7	Conclusions	159
7.1	Hypothesis & Chapter Overview	159
7.2	Chapters and Contributions Summary	160
7.2.1	Chapter 4	160
7.2.2	Chapter 5	160
7.2.3	Chapter 6	161
7.3	Thesis Conclusion	162
7.4	Further Work	163
	Bibliography	166
A	Additional Theoretical Background	168
A.1	Complexity Measures	168
	Bibliography	170

B Physical System Validation	171
B.1 SWCNT-PMMA Experiments	171
B.2 RRN and DRN System Experiments	172
B.2.1 RRN	172
B.2.2 DRN	173
B.3 Lambda Diode Network System Experiments	174
C Additional Advanced Monolithic EiM Results	175
C.1 Repetition Reliability	176
C.2 con2DDS Results	178
C.3 2DDS Results	181
C.4 flipped2DDS Results	183
D Physical iM-NN Neuroevolution Hyperparameter Investigation	185
D.1 Differential Evolution	186
D.2 OpenAI ES	187
D.3 Covariance Matrix Adaptation ES	188

List of Figures

1.1	Plot illustrating Moore’s law by showing the increase in transistor count over time.	3
1.2	Visualisation of the training process for a Configurable Analogue Processor (CAP) as described by Miller and Downing	6
1.3	Diagram of a typical nanomaterial substrate based Evolution in-Materialio (EiM) processor.	8
2.1	Typical structure of a Feed Forward Neural Network.	34
2.2	Example of an artificial neuron generating an output y from input signals $x_1, x_2, x_3, \dots, x_J$ and a bias.	35
2.3	Some common activations functions for Artificial Neural Network (ANN) artificial neurons.	35
2.4	Basic structure of an artificial single hidden layer feedforward neural network (SLFN) used as Extreme Learning Machine (ELM).	38
2.5	Structure of a basic AutoEncoder (AE) constructed using an ANN. The encoder and decoder can consist of many Hidden Layers (HLs), but the network structure is typically symmetrical.	41
2.6	A conventional RC system with a Recurrent Neural Network (RNN)-based reservoir, where only the readout weights W^{out} are trained, and the input weights W^{in} & internal reservoir weights W are fixed. . . .	43
3.1	Representation of a configurable analogue or ‘material’ processor as a black box which transforms input data voltages \mathbf{V}^{in} and configurable stimuli \mathbf{V}^c signals into new output representations \mathbf{V}^{out}	54

3.2	Illustration of the proposed EiM processor structure, where output signals are collected using an output layer and combined to create an overall processor response Y used to make a classification prediction \hat{y} .	56
3.3	Example of a 5 node fully connected network used to model EiM material processors, with three input nodes (V_1^{in} , V_2^{in} and V_1^c) and two output nodes (V_1^{out} and V_2^{out}). Input nodes can be allocated as data-driven voltages V^{in} or configuration voltage stimuli V^c . Between every pair of nodes is a component (i.e., complex sub-circuit) modelling a particular material property.	58
3.4	Top and bottom side of the final Hardware Interface (HI) Printed Circuit Board (PCB).	62
3.5	Schematic of the Hardware Interface (HI).	63
3.6	The custom Lambda Diode Network (LDN) leveraged as a physical neuron, containing three inputs (V_1^{in} , V_2^{in} , V^c) and three outputs (V_1^{out} , V_2^{out} , V_3^{out}).	64
3.7	The lineally separable (a) 2DDS, the concentric (b) con2DDS, the half moon (c) hm2DDS, and the spiral (d) sp2DDS synthetic two-dimensional datasets.	66
3.8	Complexity plots as generated by <i>proplexity</i> for the classification problem datasets, further complexity metric details given in Appendix A.1. In general, “more colour” represents a higher degree of complexity for the specific complexity measure.	69
4.1	EiM processor structure as discussed in §3.3, where output signals are collected using an output layer and combined to create an overall processor response Y used to make a classification prediction \hat{y} .	75
4.2	Examples of the role that the material and configuration voltages have on untrained EiM processor responses. Surface plot of the network response, Y as a function of input attributes a_1 and a_2 for an untrained (a) Resistor Random Network (RRN), (b) Non-Linear Random Network (NLRN), and (c) Diode Random Network (DRN) processor, and the effect of varying their two configuration voltages V_1^c and V_2^c .	77

- 4.3 Example processor responses following training using the basic algorithm for 30 iterations. Surface plot of the network response, Y as a function of input attributes a_1 and a_2 for a (a) RRN, (b) NLRN, and (c) DRN processor, using the basic EiM algorithm on the 2DDS. The same colour scale as Fig. 4.2 is used. 79
- 4.4 Examples of the impact that electrode reconfiguration can have on untrained EiM processor responses. Surface plot of the network response Y as a function of input attributes a_1 and a_2 for four randomly selected shuffle genes (i.e., permutations of the input voltage order) of an unconfigured (a) RRN, (b) NLRN, and (c) DRN processor. The same colour scale as Fig. 4.2 is used. 81
- 4.5 Examples of the effect of varying the input and output weightings on untrained EiM processor responses. Surface plot of the network response Y as a function of inputs a_1 and a_2 for an unconfigured (a) RRN, (b) NLRN, and (c) DRN processor with various output or input weightings applied. The same colour scale as Fig. 4.2 is used. 82
- 4.6 Structure of the advanced EiM processor structure. Input data and configurable stimuli (V^c) are applied to the material as voltages. The output voltages are summed to generate an overall response (Y) which is used to predict the binary class (\hat{y}). If enabled input weights (w_r^{in}) and output weights (w_q^{out}) are applied. A shuffle ‘gene’ can re-arrange the applied location of the inputs (both input data and configuration nodes). 83
- 4.7 **Effect of varying which nodes are allocated as either configuration or output electrodes on the final test fitness after training.** Surface plot of the mean test fitness (from 15 material processors, each with 5 DE repetitions) after 50 iterations using the advanced EiM algorithm on the (a) RRN, (b) NLRN, and (c) DRN networks to classify the con2DDS. The materials all had a fixed size of ten nodes, but the number of nodes allocated as configuration or outputs was varied. Unallocated nodes were left floating. 85

4.8	Example processor responses following training using the advanced EiM algorithm for 50 iterations. Surface plot of the network response, Y as a function of input attributes a_1 and a_2 for a (a) RRN, (b) NLRN and (c) DRN processor, using the advanced EiM algorithm with all the additional configuration parameters, on the con2DDS. The same colour scale as Fig. 4.2 is used.	86
4.9	Evolution of training fitness and final test fitness for EiM processors using different materials and configuration parameters, classifying the con2DDS. The mean best training fitness & standard error (from 15 material processors, each with 5 DE repetitions) over 50 iterations, with different DE algorithm configuration parameters enabled, for (a) RRN, (c) NLRN, and (e) DRN processors using the con2DDS. These are paired with box & whisker plots of the final test fitness results for the (b) RRN, (d) NLRN, and (f) DRN processors.	88
5.1	Replication of Fig. 4.6 showing a monolithic EiM processor using the expanded set of decision vectors defined in §4.3, where output signals are collected using an output layer and combined to create an overall processor response Y used to make a classification prediction \hat{y}	97
5.2	Hyperparameter sweep of mutation factor (F) and crossover rate (CR) when using Differential Evolution (DE) to optimise monolithic EiM processors to classify the (a) con2DDS and (b) sp2DDS. Each result is the mean from five DRN conductive networks, each trained as an EiM classifier for 30 epochs, with performance averaged over three DE repetitions.	99
5.3	The (a) training and (b) test fitness convergence when optimising DRNs as EiM processors to classify the con2DDS dataset, using Algorithm 5.1 with $E = 1, 2, 5, 10, 20$	103
5.4	The (a) training and (b) test fitness convergence when optimising DRNs as EiM processors to classify the Banknote dataset, using Algorithm 5.1 with $E = 1, 2, 5, 10, 20$	104

5.5	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.	106
5.6	Outputs of the sigmoid $\sigma(k)$ and entropy $H(k)$ functions for the collected network response Y of the material processor for a particular data instance k . The entropy function is different depending on the true value of the class.	107
5.7	Histogram of the accumulated final test data outputs (i.e., material processor responses Y) for all material and algorithm repetitions trained on the (a) 2DDS, (b) con2DDS, and (c) Banknote datasets comparing the use of a classification error and Binary Cross Entropy (BCE) fitness metric. (Bins=0.2)	108
5.8	Receiver Operating Characteristic (ROC) for the results of all material and algorithm repetitions trained on the (a) 2DDS, (b) con2DDS, and (c) Banknote datasets comparing the use of a classification error and BCE fitness metric.	109
5.9	Plot showing the reduction in mean test accuracy of the trained EiM systems as increasing levels of Gaussian noise is introduced to the test data on the (a) 2DDS, (b) con2DDS, and (c) Banknote datasets, comparing the classification error and BCE evolved systems.	109
5.10	Evolution of the mean test fitness, with standard deviation, comparing the Standard evolved and Regressed EiM training methods on the (a) con2DDS, (b) sp2DDS, and (c) Banknote dataset.	112
5.11	Monolithic EiM processor configured for binary classification, using a <i>fully connected input layer</i> which combines input attributes to produce an input ‘data’ voltage value.	115
5.12	Convergence of the mean test fitness when using different styles of evolved input layers to optimise an EiM processor to classify the (a) con2DDS, (b) sp2DDS, and (c) Banknote datasets.	116

6.1	Basic structure of an artificial SLFN.	125
6.2	Structure of an in-Materio Neural Network (iM-NN) exploited for classification, using a fully connected linear input & output layer and configurable voltage stimuli V^c which alter a neuron's physical behaviour.	126
6.3	Basic structure of a physical neuron based SLFN exploited as an in-Materio Extreme Learning Machine (iM-ELM).	130
6.4	Median Mean Squared Error (mse) test fitness of all the (20 systems, each with 100 parameter initialisations) iM-ELMs for each HL size increment, used to classify the (a) sp2DDS, (b) diabetes, (c) wine, (d) aca and (e) wdbc datasets. Three different material neuron topologies are considered ($[P, S, Q]$), and these are compared to the mean accuracy of 2000 traditional artificial ELMs and RR-ELMs.	133
6.5	Median mse test fitness of all the (20 systems, each with 100 parameter initialisations) <i>Virtual</i> iM-ELMs for each HL size increment, used to classify the (a) sp2DDS, (b) diabetes, (c) wine, (d) aca and (e) wdbc datasets. Three different material neuron topologies are considered ($[P, S, Q]$), and these are compared to the mean accuracy of 2000 traditional artificial ELMs and RR-ELMs.	134
6.6	The (a) Lambda Diode Network (LDN) used in experimentation and leveraged as a Physical in-Materio (PiM) neuron, and (b) the surface plot of the Lambda Diode Network (LDN)'s physical outputs for a 2D sweep of V_1^{in} & V_2^{in} , and a selection of configuration voltages V^c . .	138
6.7	Surface plots of the (a) DE, (b) OpenAI Evolutionary Strategy (OpenAI-ES), and (c) Covariance Matrix Adaptation Evolution Strategy (CMA-ES) algorithms' final mean test Cross Entropy (CE) loss classifying the sp2DDS after a fixed budget of 800000 N_{comps} , for a sweep of different population sizes λ (with respect to (<i>w.r.t</i>) the iM-NN's number of parameters d) and batch sizes bs (<i>w.r.t</i> the total number of training instances K^{train}), using a three HL neuron iM-NN.	142

6.8	Comparison of the algorithms' mean test CE loss convergence, under a fixed budget, for the (a) sp2DDS, (b) banknote, (c) iris, (d) raisin, and (e) wine datasets.	144
6.9	Structure of a five layer ANN based AutoEncoder.	147
6.10	Structure of the implemented in-Materio AutoEncoder (iM-AE) using two physical neuron based Hidden Layers in a five-layer network. . . .	148
6.11	Convergence of (a) training and test fitnesses and (b) clustering metrics including Clustering Accuracy (CA) and Adjusted Rand Index (ARI).	150
6.12	Final result's (a) encoded test data features, where labels are distinguished by colour and marker type, and (b) examples of test data inputs with their corresponding reconstructed outputs.	150
6.13	The artificial AEs mean final (a) reconstruction loss, and encoded 2D data's clustering (b) Clustering Accuracy (CA), and (c) Adjusted Rand Index (ARI) for differently sized networks on the digits dataset. The final performance of the example in-Materio AutoEncoder (iM-AE) is marked for comparison, and the point at which the artificial AEs contains the same number of parameters is denoted by a vertical blue line.	151
B.1	Example inter-node IV characteristics between one node and 10 others on the fabricated microelectrode array with deposited 1.1wt% Single Walled Carbon Nanotubes (SWCNTs) suspended in Poly(butyl methacrylate) (PBMA).	171
B.2	Experimental and simulated response (Y) surface plots for an example RNN network.	172
B.3	Experimental and simulated response (Y) surface plots for an example DRN network.	173
B.4	Histogram of the output voltage residuals using a (a) linear and (b) log y-axis. Bin width = 0.002.	174

C.1	Surface plot of the test fitness standard deviation as we either train for longer or include more repetitions. These results used the advanced DE algorithm (with shuffle gene, input and output weights) to classify the con2DDS. It shows the effect of the cumulative standard deviation of the evolved final test fitness as more algorithm repetitions are introduced. This was done for two randomly generated DRN material processors and shows that the standard deviation is low and settles after four to five repetitions of the DE algorithm.	176
C.2	Box & whisker plots showing the effect of including more randomly generated ‘material processors’ into the final result. Each of the processor types (RRN, NLRN & DRN) were solved using the basic and advanced algorithms over 50 iterations. This was then repeated for a new material so a box plot of the cumulative test fitness’s could be plotted. This was then repeated for another newly randomly generated material (and so on) such that the effect of including more material processors could be visualised. Note that introducing multiple repetitions of the DE algorithm on each individual material also improves stability.	177
C.3	The mean test fitness convergence and final test fitness box and whisker plots for the RRN, NLRN, and DRN.	179
C.4	The mean test fitness convergence for the different conductive network-based EiM processors classifying the 2DDS.	181
C.5	The mean test fitness convergence for the different conductive network-based EiM processors classifying the flipped 2DDS.	183
D.1	DE hyperparameter investigation results solving the hm2DDS for a fixed budget of 500000 N_{comps} , with $\lambda = d$	186
D.2	OpenAI-ES hyperparameter investigation results solving the hm2DDS for a fixed budget of 500000 N_{comps} , with $\lambda = d$	187
D.3	CMA-ES hyperparameter investigation results solving the hm2DDS for a fixed budget of 500000 N_{comps}	188

List of Tables

3.1	Condensed Bill Of Materials (BOM) for the final Hardware Interface (HI), detailing the Integrated Circuit (IC) components used.	62
3.2	Synthetic dataset details.	66
3.3	Real dataset details.	67
4.1	The mean test fitness $\bar{\Phi}$, standard deviation, and best test fitness Φ^* after 30 iterations, for the different Evolution in-Materio (EiM) processors.	78
4.2	Classification performance of the discussed basic and advanced EiM algorithms final classification error test fitness, compared to other common algorithms and other work.	90
5.1	Final test results for the Standard evolved and Regressed EiM, using the classification error objective function.	112
5.2	Final mean \bar{A} and best A^* accuracy test results for the Evolved and Regressed EiM compared to some common <i>sklearn</i> classification methods.	114
5.3	Final test results for the different input scheme for the monolithic EiM processors.	117
6.1	Test results for the datasets when using several common classification methods. Best accuracy highlighted in bold.	131
6.2	Best accuracy achieved from the different systems, from across the different Hidden Layer (HL) sizes and neuron topologies ($[P, S, Q]$). The best accuracy for each dataset is highlighted in bold.	134

6.3	Test results for the datasets when using several common classification methods. The best accuracy for each dataset is highlighted in bold.	139
6.4	Final test results for the in-Materio Neural Network (iM-NN)'s neuroevolution, where $\bar{\Phi}$ is the mean final fitness, \bar{A} is the mean final accuracy and A^* is the accuracy of the run which achieved the smallest test fitness Φ^* . Best achieved loss for each dataset is highlighted in bold.	145
6.5	Final test results for different trained methods encoding the digits dataset as two-dimensional features.	151
B.1	Summary fitting parameters from experimental data for internode characteristics of a 1.1wt% Single Walled Carbon Nanotube (SWCNT) / Poly(butyl methacrylate) (PBMA) blend.	171
C.1	Classification error (Φ_{error}) results from varying utilising different configuration parameters when classifying the con2DDS on the corresponding type of material processor (averaged from 15 material networks, each with 5 DE repetitions).	180
C.2	Classification error (Φ_{error}) results from varying utilising different configuration parameters when classifying the 2DDS on the corresponding type of material processor (averaged from 15 material networks, each with 5 DE repetitions).	182
C.3	Classification error (Φ_{error}) results from varying utilising different configuration parameters when classifying the flipped2DDS on the corresponding type of material processor (averaged from 15 material networks, each with 5 DE repetitions).	184

List of Publications

- [1] Benedict. A. H. Jones, J. L. P. Chouard, B. C. C. Branco, E. G. B. Vissol-Gaudin, C. Pearson, M. C. Petty, N. Al Moubayed, D. A. Zeze, and C. Groves, “Towards Intelligently Designed Evolvable Processors,” *Evolutionary Computation*, pp. 1–23, Mar. 2022. [Online]. Available: https://doi.org/10.1162/evco_a_00309
- [2] Benedict. A. H. Jones, N. Al Moubayed, D. A. Zeze, and C. Groves, “Enhanced methods for Evolution in-Materio Processors,” in *2021 International Conference on Rebooting Computing (ICRC)*, Nov. 2021, pp. 109–118. [Online]. Available: <https://doi.org/10.1109/icrc53822.2021.00026>
- [3] B. A. H. Jones, N. Al Moubayed, D. A. Zeze, and C. Groves, “In-Materio Extreme Learning Machines,” in *Parallel Problem Solving from Nature – PPSN XVII*, ser. Lecture Notes in Computer Science, G. Rudolph, A. V. Kononova, H. Aguirre, P. Kerschke, G. Ochoa, and T. Tušar, Eds. Cham: Springer International Publishing, 2022, pp. 505–519. [Online]. Available: https://doi.org/10.1007/978-3-031-14714-2_35
- [4] Benedict. A. H. Jones, N. Al Moubayed, D. A. Zeze, and C. Groves, “Training In-Materio Neural Networks,” *Paper submitted for publication*, 2023.

Acronyms

<i>mse</i>	Mean Squared Error.
<i>nmse</i>	Normalised Mean Squared Error.
<i>w.r.t</i>	with respect to.
Adam	Adaptive Moment Estimation.
ADC	Analogue to Digital Converter.
AE	AutoEncoder.
ANN	Artificial Neural Network.
ARI	Adjusted Rand Index.
AUC	Area Under Curve.
BCE	Binary Cross Entropy.
BOM	Bill Of Materials.
CA	Clustering Accuracy.
CAP	Configurable Analogue Processor.
CE	Cross Entropy.
CMA-ES	Covariance Matrix Adaptation Evolution Strategy.
CMOS	Complementary Metal-Oxide-Semiconductor.
DAC	Digital to Analogue Converter.
DC	Direct Current.
DE	Differential Evolution.
DNA	Deoxyribonucleic Acid.
DRN	Diode Random Network.

EA	Evolutionary Algorithm.
EiM	Evolution in-Materio.
ELM	Extreme Learning Machine.
ES	Evolutionary Strategy.
ESN	Echo State Network.
FPGA	Field-Programmable Gate Array.
GD	Gradient Descent.
GPIO	General Purpose Input/Output.
HI	Hardware Interface.
HL	Hidden Layer.
IC	Integrated Circuit.
iM-AE	in-Materio AutoEncoder.
iM-ELM	in-Materio Extreme Learning Machine.
iM-NN	in-Materio Neural Network.
IoT	Internet-of-Things.
IV	Current-Voltage.
JFET	Junction-gate Field-Effect Transistor.
LC	Liquid Crystal.
LD	Lambda Diode.
LDN	Lambda Diode Network.
LSM	Liquid State Machine.
MGD	Multivariate Gaussian Distribution.
ML	Machine Learning.
MUX	Multiplexer.
Nº	Number of.
NC	Natural Computing.
NDR	Negative Differential Region.
NES	Natural Evolution Strategy.

NLRN	Non-Linear Random Network.
NN	Neural Network.
NS	Novelty Search.
Op-Amp	Operational Amplifier.
OpenAI-ES	OpenAI Evolutionary Strategy.
PBMA	Poly(butyl methacrylate).
PCA	Principal Component Analysis.
PCB	Printed Circuit Board.
PiM	Physical in-Materio.
PMMA	Poly(methyl methacrylate).
PSO	Particle Swarm Optimisation.
RC	Reservoir Computing.
RNN	Recurrent Neural Network.
ROC	Receiver Operating Characteristic.
RRN	Resistor Random Network.
SGD	Stochastic Gradient Descent.
SI	Software Interface.
SiM	Simulated in-Materio.
SLFN	single hidden layer feedforward neural network.
SNN	Spiking Neural Network.
SPI	Serial Peripheral Interface.
SPICE	Simulation Program with Integrated Circuit Emphasis.
SWCNT	Single Walled Carbon Nanotube.
UC	Unconventional Computing.

Glossary

<i>computation</i>	An act or the process of calculating something.
<i>dynamic material</i>	A material with variable IV characteristics.
<i>in-materio</i>	Processing or computation occurring in an exploited material or medium.
<i>in-simulo</i>	Processing or computation occurring in a simulation, also referred to as <i>in-silico</i> .
<i>neuromorphic computing</i>	Computing inspired by the structure and function of the brain.
<i>static material</i>	A material with fixed IV characteristics.
<i>unconventional computing</i>	Computing without standard digital computers.
black box	Systems within which it is difficult or impossible to understand how variables are being combined to make predictions.
decision boundary	A hyper plane (boundary or surface) which separates data points into specific classes.
neuroevolution	A type of machine learning where an Evolutionary Algorithm is used to optimise a Neural Network.

Nomenclature

Symbol	Description	Unit
A^*	Best Accuracy	—
B	A (mini-)batch of the training data	—
CR	Differential Evolution Crossover	—
F	Differential Evolution Mutation Factor	—
H	Entropy	—
K	Number of data instances	—
L	Total number of labels/classes	—
N_{comps}	Number of computations (a measurement of the number of data instance write/read cycles to a material)	—
P	Number of material input nodes/electrodes	—
Q	Number of material output nodes/electrodes	—
R	Number of data driven material nodes/electrodes	—
S	Number of evolvable stimuli driven material nodes/electrodes	—
V^c	Input ‘configuration’ voltage stimuli	V
V^{in}	Input ‘data’ voltage	V
V^{out}	Read output voltage	V
Y	Network Response	V
Φ^*	Best fitness or loss	—
Φ_{bce}	Binary Cross Entropy Loss	—
Φ_{ce}	Cross Entropy Loss	—
Φ_{error}	Classification Error Loss	—

Symbol	Description	Unit
Φ_{mse}	Mean Squared Error Loss	—
Φ_{nmse}	Normalised Mean Squared Error Loss	—
Φ	Objective Function used to calculate a potential solution's fitness or loss	—
α	Learning Rate	—
\bar{A}	Mean Accuracy	—
$\bar{\Phi}$	Mean fitness or loss	—
\mathbf{X}	Vector of decision variables (i.e., a particular solution, or genome)	—
θ	Best solution or population member	—
\mathbf{p}	Population of solutions	—
\mathbf{t}	Trial population of solutions	—
\hat{y}	Predicted output or output label	—
λ	Population size	—
σ	Standard Deviation (std)	—
a	Input attribute	—
bs	Batch size	—
d	The test data subset	—
k	A particular data instances within a dataset	—
l	Indexing a label or class	—
w^{in}	Input Weight	—
w^{out}	Output Weight	—
y	True label	—

Chapter 1

Introduction

1.1 Chapter Overview	1
1.2 Historical Context	2
1.3 Unconventional Computing	4
1.4 Evolution in-Materio	5
1.5 Research Hypothesis	11
1.6 Thesis Structure	12
Bibliography	13

1.1 Chapter Overview

Evolution in-Materio (EiM) is an unconventional computing paradigm which uses an Evolutionary Algorithm (EA) to leverage the inherent complex properties of a nanomaterial substrate or physical medium for computation. This bottom up exploitation of physical properties is in contrast with the traditional ‘top down’ approach typically used to develop modern silicon-based computers.

This chapter introduces EiM processors and provides context for a renewed interest in *unconventional computing*. This touches on related fields and discusses the literature. While EiM processors show promise, problems in their development exist, such as slow manufacturing and physical experimentation, and possible scaling issues. Here, the research hypothesis and thesis structure is outlined for the reader’s convenience.

1.2 Historical Context

The twenty-first century has heralded an unprecedented explosion in technological innovation. Many liken this new ‘digital age’ to that of the industrial revolution, bringing opportunity to the modern world. However, this new interconnected world requires one vital component – computers. These come in all shapes and sizes, from large and power hungry data centres, to small Internet-of-Things (IoT) edge devices [1].

The Cambridge Dictionary defines a processor as “the part of a computer that performs operations on the information that is put into it”. Therefore, devices or systems which can process information and perform *computation* pre-date modern electronic computers by centuries. Indeed, the oldest known computational device is the abacus [2] used in ancient civilisations around the world. The oldest known processor might therefore be the water clock, a device used to measure the passage of time by regulating the flow of a liquid from either into or out of a vessel [3, 4, 5]. Such devices exploit the vessel structure, and its physical dynamics, for the particular time ‘processing’ task.

The modern computer is generally thought to have been worked on by Babbage & Lovelace [6, 7], Zuse [8] and Turing [9]. Indeed, Ada Lovelace is often attributed as the ‘first programmer’ [10]. These early mechanical and analogue computers often used wheels, disks, shafts and gears to perform calculations [11]. The first digital computers used vacuum tubes to represent binary information, before being replaced with the transistor – now the fundamental building block for much of modern electronic systems.

Modern digital computers are general purpose machines which can run a variety of programs. These generally follow the Von Neumann architecture [12] where a single store (i.e., addressable memory) is used for both machine instructions and data. Therefore, there is a clear separation within the computer’s permanent structure (hardware) and its instructions (software). This enabled flexible computers that could be reconfigured (programmed) by entering new instructions into memory, rather than physically rewired. However, Von Neumann architecture limits a computer to sequential processing and requires a well-defined structure with a

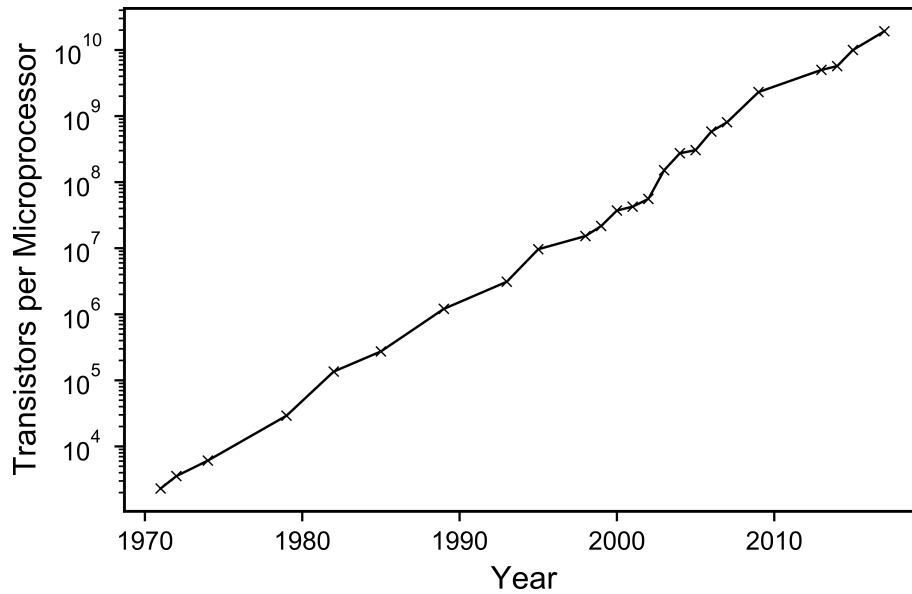


Figure 1.1: Plot illustrating Moore’s law by showing the increase in transistor count over time. Data taken from [19].

central processing unit.

The adoption of digital technology for personal use has reached new highs, with 8.27 billion worldwide mobile phone subscriptions, and approximately 60% of the world population having had internet access in 2020 [13]. A growing number of devices are being connected to the internet, helping realise the IoT – a network of physical objects which can communicate [14] and perform as smarter systems. IoT systems rely on a number of underlying technologies, such as sensors, communication systems, internet protocols and embedded computational devices. Beyond personal use, the application of computing technology is expanding, from smart cities [15] and autonomous vehicles [16], to data centres [17] and large Machine Learning (ML) models [18].

These advances have been made possible by the continued progress in electronic device development. In 1965 Gordon E. Moore observed that manufacturers had been doubling the density of components per Integrated Circuit (IC) at regular intervals (roughly every two years) [20], a trend seen in Fig. 1.1 which became known as Moore’s law. Indeed, the success of Moore’s law has helped drive the silicon manufacturing industry. However, manufactures are now facing serious challenges, with fundamental physical limits suggesting field-effect transistor gate length is unlikely

to go below 5 nm [21].

The effort to continue scaling conventional silicon Complementary Metal-Oxide-Semiconductor (CMOS) devices has overwhelmingly dominated intellectual & financial capital investments from industry, government and academia [21]. However, the challenges to typical silicon device development has led to a growing interest in new *unconventional computing* methods [22]. Such devices may provide power and speed efficiency gains, or even offer entirely new computational paradigms.

1.3 Unconventional Computing

Unconventional Computing (UC), sometimes referred to as unconventional computation [23] or alternative computing, is a wide area of study with varied content and many related fields. While whether something is ‘unconventional’ can be subjective [23], UC methods are broadly defined as *computing without standard digital computers* [24]. Examples can include leveraging Field-Programmable Gate Arrays (FPGAs) [25], Deoxyribonucleic Acid (DNA), quantum properties, mechanical devices, water [26], nano-technologies and more.

Natural Computing (NC) has a strong relationship with UC; it is a large field that contains techniques inspired from nature or the use of natural materials to perform computation [24, 27]. This can include algorithms using concepts such as reproduction, mutation, recombination, and natural selection [28], or physical systems performing computation such as with physarum (slime mould), DNA or more [29, 24]. Many argue that there are lessons which can be drawn from nature, since natural evolution has produced “biological machines” which still maintain a level of complexity far above what conventional computers have achieved [30]. Indeed, research such as Artificial Neural Networks (ANNs) was a landmark piece of work in the branch of nature inspired computing [27]. The realisation of such biologically, and particularly brain, inspired systems is often called *neuromorphic computing* [31], and has resulted in a large body of research [32, 33], from leaky integrate-and-fire spike-driven hardware [34, 35] to memristive synapses [36] and crossbar arrays [37, 38, 39].

The growing curiosity in UC has also revived interest in analogue computing [29]. This coincides with a desire to produce efficient but powerful computing and ML at “the edge” [1] and physical neuromorphic hardware [40]. For this reason, the idea of using physical analogue systems has remained an attractive option, due to many analogue devices’ high theoretical throughput and low-energy consumption [41]. Additionally, constraints associated with digital computing could be sidestepped, such as avoiding analogue-to-digital conversion (i.e., discretisation) [42]. However, such emerging UC devices face challenges such as device variability, stochastic behaviour and scalability [43, 44].

While ANNs have exploded in popularity, some consider them to be over parameterised [45, 46]. Conversely, the complexity engineering approach [44] states that one should attempt to minimise external control of a complex system being leveraged for computation. In doing so, a system could instead be self-organising with emergent functionality – contrary to a classical engineering approach which is often top down and well-defined. Such approaches prompts us to re-think how computing methods can be adapted for new UC devices, or perhaps search for entirely new computational paradigms [47].

1.4 Evolution in-Materio

EiM is an UC method which seeks to exploit a physical substrate’s inherent complex properties to perform a computational task. Initially proposed by Miller & Downing in 2002 [30], EiM was inspired by the remarkably complex and varied functions that simple nucleotides can perform when configured by evolution into a genome. They envisaged a type of evolutionary exploitable device operated as a Configurable Analogue Processor (CAP), whose configuration (and therefore operation) could be selected by some discrete set of parameters such as voltages, fields or other physical signals. They argued that the evolvable hardware research community was too focused on transistor technology, and that many other types of reconfigurable systems may exist, suggesting that materials with rich physical properties might be ideal. They also suggested that numerous CAP configurations might need to be

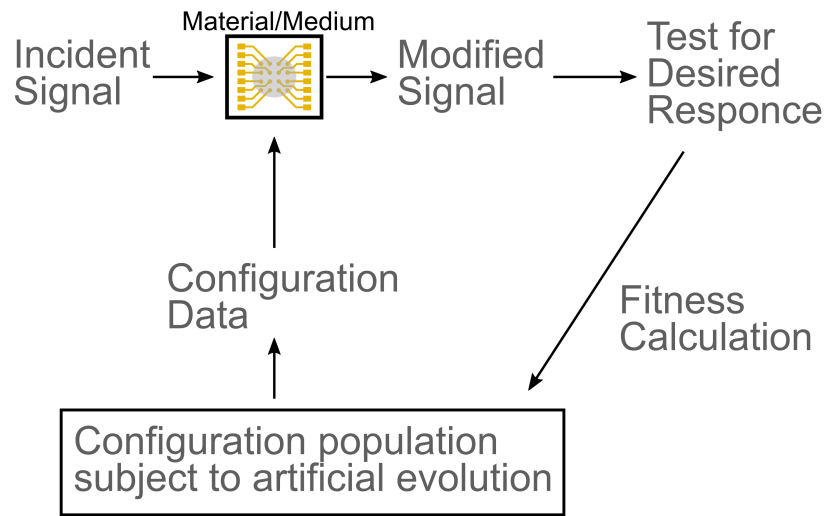


Figure 1.2: Visualisation of the training process for a Configurable Analogue Processor (CAP) as described by Miller and Downing [30].

tested before one is found that transforms the incident signal in the desired manner. The resulting ‘fitness landscape’ was likely to have many local optima, to which an optimising EA [28] might be best suited for. The EA performs an iterative search, optimising the CAP’s input/output relationship until a certain performance level is achieved or a particular number of iterations has elapsed. Such a training process is illustrated in Fig. 1.2. Therefore, EiM uses a ‘bottom up’ approach where the material is leveraged for computation without explicit knowledge of its internal properties.

In the past decade, significant progress has been made advancing the EiM paradigm. The term EiM processor is used here to describe a system which exploits a configurable analogue or ‘*in-materio*’ processor using an EA. Examples of such devices include the use of a nanomaterial substrate such as metal-nanoparticles [48, 49], Single Walled Carbon Nanotube (SWCNT) composites [50, 51], Liquid Crystals (LCs) [52, 53], LCs/SWCNT mixtures [54], and dopant-atom networks [55] – all of which were configured with the application of static ‘configuration’ voltages. However, it is highlighted that any material or medium which is interfaceable and contains interesting physical properties might be used in an EiM processor. This could include using light [56], radio waves [57], acoustics [58], or potentially turning an entire building into a computational resource by exploiting conductive concrete [59].

Therefore, at the simplest level, perhaps even the humble water clock (mentioned in §1.2) could be described as an *in-materio* processor, exploiting the dynamics of a chosen vessel to perform a time keeping process.

So, EiM devices consist of a system within which a material or medium's physical properties are exploited and leveraged towards the desired computational task. Thus, EiM devices generally comprised of three constituent parts:

- a material whose characteristics can be altered via external stimuli,
- a Hardware Interface which can apply input and read output signals from the material,
- a device which can host and execute an EA to optimise the material.

Within this work, an exploited nanomaterial substrate will be referred to interchangeably as a configurable analogue, material or *in-materio* processor.

Research has often focused on conductive nanomaterial substrates since they can be easily accessed and manipulated via the application and reading 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. An example of such an EiM processor device is depicted in Fig. 1.3. Nanomaterial based EiM processors have used a range of microelectrode array sizes to contact the material: such as sixty-four [60], sixteen [61, 50] or often fewer electrodes [62, 55, 48]. Smaller networks using only eight electrodes have shown promising results as physical realisations of high-capacity neurons [41].

The electronic functionality of these EiM processors is not designed by the assembling of discrete components, rather an optimal material configuration is sought via evolution through a supervised learning process. The human element of EiM processor design is the selection of an appropriate configurable material/medium, selecting the physical stimuli, formulating the computational problem, and choosing an algorithm to efficiently optimise the system [63, 64, 65, 66]. While EAs are traditionally used to produce EiM processors, other algorithms or methods could be used to produce novel *in-materio* devices, such as Particle Swarm Optimisation [63, 64].

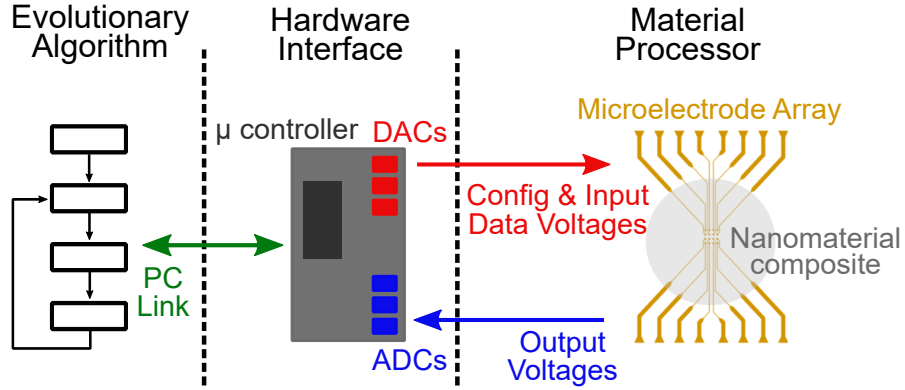


Figure 1.3: Diagram of a typical nanomaterial substrate based EiM processor.

So far, EiM processors have been well positioned to operate as basic classifiers for non-temporal (static) data, performing complex multivariate machine-learning problems [54, 67] or as logic gates [48, 62, 51]. However, the literature is predominantly made up of case studies, using a range of techniques and nanomaterial substrates. It has been demonstrated that optimising different nanomaterials with the same EA for the same function has both varying degrees of success and training time [66, 51, 68, 67]. Since these nanomaterial processors are analogue, they often have underlying physical properties that are difficult to model. This leads them to be treated as black boxes, making investigation into which nanomaterial properties are beneficial difficult. Even EiM processors fabricated from nominally the same nanomaterial and optimised by the same EA for the same computational problem vary in quality of solution due to the inherent randomness of nanomaterial morphology [67] and EA convergence. Collating data and conducting experimentation to further investigate these issues is challenging due to the slow fabrication and training processes that are required for each EiM processor [49]. Additionally, while the literature presents many implementations of EiM devices, a lack of a unified method or common approach makes reliable, repeatable investigation difficult. As such, the fundamental question of which configurable material properties lead to better performing EiM processors, and what algorithm properties will lead to better exploited performance, remains largely unanswered.

Other computational frameworks closely overlap with EiM. For example, Ex-

Extreme Learning Machines (ELMs) and Reservoir Computings (RCs) present a good analogy for *in-materio* processors since both involve the exploitation of random networks. These systems depend on the underlying assumption that the randomised network/reservoir will produce useful and often higher dimensional output states that are used to process the data more successfully. Notably, within these fields of research, it is generally assumed that the network/reservoir remains fixed after its inception. However, previous work has shown that some stochastic optimisation can improve a system's performance [68, 69, 70]. ELMs were developed from single hidden layer feedforward neural networks and are generally employed to process non-temporal data [71]. Examples of physical implementations of ELM remain sparse, but include memristor based networks [72] and photonic systems [73, 74]. RC was developed from Recurrent Neural Networks and are generally employed to process temporal data. Like EiM, Physical RCs [75] could lead to low power, efficient and fast systems which can operate at 'the edge'. Examples include the use of circuit (anti-parallel diode) based non-linear neuron [76], memristive network [72, 77], FPGAs [25], and magnetic spintronic [78] based reservoirs. There remains significant opportunity to develop both classical and quantum substrates [79] for both ELM and RC. Drawing elements from such successful ML methods and leveraging physical substrates using EiM could unlock efficient but powerful *unconventional computing* resources.

Work combining physical RC and EiM was carried out by Matthew Dale et al. at York University [68, 66] in which they constructed Reservoir Computing in-Materio (RCiM) devices. These deviate from traditional physical RC by introducing aspects of EiM; specifically, utilising an EA to tune a reservoir by evolving some stimuli (equivalent to optimising the internal reservoir parameters). Earlier discussion from Dale et al. considered the advantages of the RC readout layer (i.e weights for the output signals) and how it allows the system to selectively choose and separate/combine interesting output signal patterns [68]. Within the work carried out on RCiM, evolvable configuration parameters included static (configuration) voltages, connection location, a variable number of inputs/outputs and input weights [68, 66, 80]. They hypothesised that an observed increase in performance might be the result of: (i) in-

put weightings allowing variations of the input which cause interesting interactions, (ii) a conductive network might not be present across all electrodes, instead several may exist, so additional inputs-outputs may grant access to these smaller networks, (iii) combining and weighting several outputs allows training to exploit the whole material, not just a single area around a particular electrode. This conjecture requires further exploration and represents the general lack of established knowledge about how nanomaterial processors can be best exploited. The importance of developing this foundational understanding cannot be understated, and is required to provide actionable guidance on the future design of EiM devices.

The scalability of EiM devices is an incredibly important, but as yet unanswered question. EiM processors have typically used a single substrate ‘monolithic’ structure. With larger and more complex ML problems, a monolithic EiM processor would require a physically larger nanomaterial substrate with a larger microelectrode array. However, such an approach is unlikely to perform well, as such larger devices will have weaker interactions between distant electrodes. Therefore, to scale *in-materio* systems to process larger, more complex datasets, devices will have to move beyond the typical monolithic structure. The introduction of novel devices or intra-substrate structures represents an exciting avenue for research.

In summary, while EiM processors show promise as an *unconventional computing* resource, problems in their development exist. To isolate and investigate fundamental questions about optimal material and algorithms properties, a standard approach is required. However, investigations are often slow, due to lengthy physical manufacturing and physical experimentation. Finally, to advance the paradigm to larger, more complex computational problems, new and novel EiM device structures, and accompanying algorithms, will be required.

1.5 Research Hypothesis

Nanomaterial substrates can be used to produce EiM devices, but physical development and experimentation can be slow. Instead, a simulation can be used to model a material and use it as a proxy for experimentation; this will allow fast and efficient investigation into what material and algorithm properties are most beneficial to EiM processors.

Further to this, new device structures will be considered, drawing from concepts in the wider Machine Learning field, such as feed forward Neural Networks and AutoEncoders. These devices will address scaling issues found in typical monolithic (single substrate) EiM processors.

1.6 Thesis Structure

The thesis structure is as follows:

Chapter 2 presents background information about EiM processors, and related topics including ANNs and RC. Detailed descriptions about relevant EA and objective fitness functions are given.

Chapter 3 details how a conductive nanomaterial can be leveraged as a CAP and used in EiM for classification. The simulated model and physical testing platform are presented. Finally, the datasets used for classification and dimensionality reduction tasks are introduced.

Chapter 4 is the first results chapter which examines fundamental algorithm and material interaction, establishing a better understanding of EiM and typical single substrate ‘monolithic’ devices using fast and efficient simulations.

Chapter 5 reports on several enhancements to EiM devices and their exploiting algorithms. This includes making better use of the available training data, the benefits of a cross entropy fitness metric, the use of a regressed output layer and fully connected input layer.

Chapter 6 proposes a novel in-Materio Neural Network (iM-NN) device structure, enabling a scalable system by stacking several *in-materio* processing units in parallel and drawing on ML concepts. These iM-NNs are first simulated and trained as Extreme Learning Machines. Then using a Raspberry Pi and Hardware Interface, physical realisations of iM-NNs are investigated in the lab, performing classification and then constructing an AutoEncoder which are trained via neuroevolution.

Chapter 7 outlines the main conclusions of this thesis, and suggests several possible directions for future research.

Bibliography

- [1] M. G. S. Murshed, C. Murphy, D. Hou, N. Khan, G. Ananthanarayanan, and F. Hussain, “Machine Learning at the Network Edge: A Survey,” *ACM Computing Surveys*, vol. 54, no. 8, pp. 170:1–170:37, Oct. 2021. [Online]. Available: <http://doi.org/10.1145/3469029>
- [2] “Ancient Computation Devices,” in *Ancient Engineers’ Inventions*, ser. History of Mechanism and Machine Science, C. Rossi, F. Russo, and F. Russo, Eds. Dordrecht: Springer Netherlands, 2009, pp. 41–59. [Online]. Available: https://doi.org/10.1007/978-90-481-2253-0_4
- [3] D. Allen, “A Schedule of Boundaries: An Exploration, Launched From the Water-Clock, of Athenian Time,” *Greece & Rome*, vol. 43, no. 2, pp. 157–168, Oct. 1996. [Online]. Available: <https://doi.org/10.1093/gr/43.2.157>
- [4] S. Remijsen, “Living by the Clock. The Introduction of Clock Time in the Greek World,” *Klio*, vol. 103, no. 1, pp. 1–29, Jun. 2021. [Online]. Available: <https://doi.org/10.1515/klio-2020-0311>
- [5] J. F. Fleet, “XII. The Ancient Indian Water-clock,” *Journal of the Royal Asiatic Society*, vol. 47, no. 2, pp. 213–230, Apr. 1915. [Online]. Available: <https://doi.org/10.1017/S0035869X00048139>
- [6] L. F. Menabrea and A. Lovelace, “Sketch of the analytical engine invented by charles babbage,” 1842.
- [7] S. Charman-Anderson, “Ada lovelace: Victorian computing visionary,” vol. 36, pp. 35–41, Mar. 2015.
- [8] R. Rojas, “Babbage Meets Zuse: A Minimal Mechanical Computer,” in *Unconventional Computation and Natural Computation*, ser. Lecture Notes in Computer Science, M. Amos and A. CONDON, Eds. Cham: Springer International Publishing, 2016, pp. 25–34.
- [9] A. M. Turing, “Computing Machinery and Intelligence,” in *Parsing the Turing Test: Philosophical and Methodological Issues in the Quest for the Thinking Computer*, R. Epstein, G. Roberts, and G. Beber, Eds. Dordrecht: Springer Netherlands, 2009, pp. 23–65. [Online]. Available: https://doi.org/10.1007/978-1-4020-6710-5_3
- [10] T. Haigh and M. Priestley, “Innovators assemble: Ada Lovelace, Walter Isaacson, and the superheroines of computing,” *Communications of the ACM*, vol. 58, no. 9, pp. 20–27, Aug. 2015. [Online]. Available: <https://dl.acm.org/doi/10.1145/2804228>
- [11] G. O’Regan, “What Is a Computer?” in *A Brief History of Computing*, G. O’Regan, Ed. London: Springer, 2012, pp. 23–34. [Online]. Available: https://doi.org/10.1007/978-1-4471-2359-0_2

- [12] J. von Neumann, "First draft of a report on the EDVAC," *IEEE Annals of the History of Computing*, vol. 15, no. 4, pp. 27–75, 1993.
- [13] H. Ritchie and M. Roser, "Technology Adoption," *Our World in Data*, Oct. 2017. [Online]. Available: <https://ourworldindata.org/technology-adoption>
- [14] A. Al-Fuqaha, M. Guizani, M. Mohammadi, M. Aledhari, and M. Ayyash, "Internet of Things: A Survey on Enabling Technologies, Protocols, and Applications," *IEEE Communications Surveys & Tutorials*, vol. 17, no. 4, pp. 2347–2376, 2015.
- [15] M. Batty, K. W. Axhausen, F. Giannotti, A. Pozdnoukhov, A. Bazzani, M. Wachowicz, G. Ouzounis, and Y. Portugali, "Smart cities of the future," *The European Physical Journal Special Topics*, vol. 214, no. 1, pp. 481–518, Nov. 2012. [Online]. Available: <https://doi.org/10.1140/epjst/e2012-01703-3>
- [16] S. Campbell, N. O'Mahony, L. Krpalcova, D. Riordan, J. Walsh, A. Murphy, and C. Ryan, "Sensor Technology in Autonomous Vehicles : A review," in *2018 29th Irish Signals and Systems Conference (ISSC)*, Jun. 2018, pp. 1–4.
- [17] K. Kant, "Data center evolution: A tutorial on state of the art, issues, and challenges," *Computer Networks*, vol. 53, no. 17, pp. 2939–2965, Dec. 2009. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S1389128609003090>
- [18] R. Dale, "GPT-3: What's it good for?" *Natural Language Engineering*, vol. 27, no. 1, pp. 113–118, Jan. 2021. [Online]. Available: <https://www.cambridge.org/core/journals/natural-language-engineering/article/gpt3-whats-it-good-for/0E05CFE68A7AC8BF794C8ECBE28AA990>
- [19] M. Roser, H. Ritchie, and E. Mathieu, "Technological Change," *Our World in Data*, May 2013. [Online]. Available: <https://ourworldindata.org/technological-change>
- [20] R. Schaller, "Moore's law: Past, present and future," *IEEE Spectrum*, vol. 34, no. 6, pp. 52–59, Jun. 1997.
- [21] R. S. Williams, "What's Next? [The end of Moore's law]," *Computing in Science & Engineering*, vol. 19, no. 2, pp. 7–13, Mar. 2017.
- [22] 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. [Online]. Available: <https://doi.org/10.1109/MC.2017.8>
- [23] C. S. Calude, "Unconventional Computing: A Brief Subjective History," in *Advances in Unconventional Computing: Volume 1: Theory*, ser. Emergence, Complexity and Computation, A. Adamatzky, Ed. Cham: Springer International Publishing, 2017, pp. 855–864. [Online]. Available: https://doi.org/10.1007/978-3-319-33924-5_31

- [24] M. Oltean, “Unconventional Computing: A Short Introduction,” *Studia Universitatis Babes-Bolyai : Series Informatica*, vol. 54, Jul. 2009.
- [25] I. Shani, L. Shaughnessy, J. Rzasa, A. Restelli, B. R. Hunt, H. Komkov, and D. P. Lathrop, “Dynamics of analog logic-gate networks for machine learning,” *Chaos: An Interdisciplinary Journal of Nonlinear Science*, vol. 29, no. 12, p. 123130, Dec. 2019. [Online]. Available: <https://aip.scitation.org/doi/10.1063/1.5123753>
- [26] N. Taberlet, Q. Marsal, J. Ferrand, and N. Plihon, “Hydraulic logic gates: Building a digital water computer,” *European Journal of Physics*, vol. 39, no. 2, p. 025801, Jan. 2018. [Online]. Available: <https://dx.doi.org/10.1088/1361-6404/aa97fc>
- [27] L. N. de Castro, “Fundamentals of natural computing: An overview,” *Physics of Life Reviews*, vol. 4, no. 1, pp. 1–36, Mar. 2007. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S1571064506000315>
- [28] A. N. Sloss and S. Gustafson, “2019 Evolutionary Algorithms Review,” *arXiv:1906.08870 [cs]*, Jun. 2019. [Online]. Available: <http://arxiv.org/abs/1906.08870>
- [29] O. Bournez and A. Pouly, “A Survey on Analog Models of Computation,” May 2018. [Online]. Available: <http://arxiv.org/abs/1805.05729>
- [30] 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. [Online]. Available: <http://ieeexplore.ieee.org/document/1029882/>
- [31] M. Ziegler, “Novel hardware and concepts for unconventional computing,” *Scientific Reports*, vol. 10, no. 1, p. 11843, Jul. 2020. [Online]. Available: <https://www.nature.com/articles/s41598-020-68834-1>
- [32] J. Hasler and H. Marr, “Finding a roadmap to achieve large neuromorphic hardware systems,” *Frontiers in Neuroscience*, vol. 7, 2013. [Online]. Available: <https://www.frontiersin.org/articles/10.3389/fnins.2013.00118>
- [33] D. Marković, A. Mizrahi, D. Querlioz, and J. Grollier, “Physics for neuromorphic computing,” *Nature Reviews Physics*, vol. 2, no. 9, pp. 499–510, Sep. 2020. [Online]. Available: <https://www.nature.com/articles/s42254-020-0208-2>
- [34] S. Furber, “Large-scale neuromorphic computing systems,” *Journal of Neural Engineering*, vol. 13, no. 5, p. 051001, Aug. 2016. [Online]. Available: <https://dx.doi.org/10.1088/1741-2560/13/5/051001>
- [35] M. Davies, N. Srinivasa, T.-H. Lin, G. Chinya, Y. Cao, S. H. Choday, G. Dimou, P. Joshi, N. Imam, S. Jain, Y. Liao, C.-K. Lin, A. Lines, R. Liu, D. Mathaikutty, S. McCoy, A. Paul, J. Tse, G. Venkataramanan, Y.-H. Weng, A. Wild,

- Y. Yang, and H. Wang, “Loihi: A Neuromorphic Manycore Processor with On-Chip Learning,” *IEEE Micro*, vol. 38, no. 1, pp. 82–99, Jan. 2018.
- [36] B. Sueoka and F. Zhao, “Memristive synaptic device based on a natural organic material—honey for spiking neural network in biodegradable neuromorphic systems,” *Journal of Physics D: Applied Physics*, vol. 55, no. 22, p. 225105, Mar. 2022. [Online]. Available: <https://doi.org/10.1088/1361-6463/ac585b>
- [37] S. Agarwal, S. J. Plimpton, D. R. Hughart, A. H. Hsia, I. Richter, J. A. Cox, C. D. James, and M. J. Marinella, “Resistive memory device requirements for a neural algorithm accelerator,” in *2016 International Joint Conference on Neural Networks (IJCNN)*, Jul. 2016, pp. 929–938.
- [38] D. S. Jeong, K. M. Kim, S. Kim, B. J. Choi, and C. S. Hwang, “Memristors for Energy-Efficient New Computing Paradigms,” *Advanced Electronic Materials*, vol. 2, no. 9, p. 1600090, 2016. [Online]. Available: <https://onlinelibrary.wiley.com/doi/abs/10.1002/aelm.201600090>
- [39] A. J. Pérez-Ávila, E. Pérez, J. B. Roldán, C. Wenger, and F. Jiménez-Molinos, “Multilevel memristor based matrix-vector multiplication: Influence of the discretization method,” in *2021 13th Spanish Conference on Electron Devices (CDE)*, Jun. 2021, pp. 66–69.
- [40] S. Bains, “The business of building brains,” *Nature Electronics*, vol. 3, no. 7, pp. 348–351, Jul. 2020. [Online]. Available: <https://www.nature.com/articles/s41928-020-0449-1>
- [41] 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. [Online]. Available: <http://arxiv.org/abs/2007.12371>
- [42] F. Zangeneh-Nejad, D. L. Sounas, A. Alù, and R. Fleury, “Analogue computing with metamaterials,” *Nature Reviews Materials*, vol. 6, no. 3, pp. 207–225, Mar. 2021. [Online]. Available: <https://www.nature.com/articles/s41578-020-00243-2>
- [43] M. Verhelst and B. Murmann, “Machine Learning at the Edge,” in *NANO-CHIPS 2030: On-Chip AI for an Efficient Data-Driven World*, ser. The Frontiers Collection, B. Murmann and B. Hoefflinger, Eds. Cham: Springer International Publishing, 2020, pp. 293–322. [Online]. Available: https://doi.org/10.1007/978-3-030-18338-7_18
- [44] N. Ganesh, “Rebooting Neuromorphic Hardware Design – A Complexity Engineering Approach,” *arXiv:2005.00522 [cs]*, Sep. 2020. [Online]. Available: <http://arxiv.org/abs/2005.00522>
- [45] S. Goldt, M. S. Advani, A. M. Saxe, F. Krzakala, and L. Zdeborová, “Generalisation dynamics of online learning in over-parameterised neural networks,” Jan. 2019. [Online]. Available: <http://arxiv.org/abs/1901.09085>

- [46] K. A. Sankararaman, S. De, Z. Xu, W. R. Huang, and T. Goldstein, "The Impact of Neural Network Overparameterization on Gradient Confusion and Stochastic Gradient Descent," Jul. 2020. [Online]. Available: <http://arxiv.org/abs/1904.06963>
- [47] V. G. Cerf, "Unconventional computing," *Communications of the ACM*, vol. 57, no. 10, p. 7, Sep. 2014. [Online]. Available: <https://doi.org/10.1145/2666093>
- [48] 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. [Online]. Available: <http://www.nature.com/articles/nnano.2015.207>
- [49] K. Greff, R. M. J. van Damme, J. Koutnik, H. J. Broersma, J. O. Mihal, 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. [Online]. Available: <https://research.utwente.nl/en/publications/using-neural-networks-to-predict-the-functionality-of-reconfigura>
- [50] 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. [Online]. Available: <http://www.nature.com/articles/srep32197>
- [51] 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. [Online]. Available: <http://aip.scitation.org/doi/10.1063/1.4915343>
- [52] 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.
- [53] 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.
- [54] 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. [Online]. Available: <http://ieeexplore.ieee.org/document/8123677/>
- [55] 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. [Online]. Available: <https://www.nature.com/articles/s41586-019-1901-0>
- [56] Y. Viero, D. Guérin, A. Vladyka, F. Alibart, S. Lenfant, M. Calame, and D. Vuillaume, "Light-Stimulatable Molecules/Nanoparticles Networks for Switchable Logical Functions and Reservoir Computing," *Advanced Functional Materials*, vol. 28, no. 39, p. 1801506, 2018. [Online]. Available: <https://onlinelibrary.wiley.com/doi/abs/10.1002/adfm.201801506>
- [57] D. Linden and E. Altshuler, "Evolving wire antennas using genetic algorithms: A review," in *Proceedings of the First NASA/DoD Workshop on Evolvable Hardware*, Jul. 1999, pp. 225–232.
- [58] A. Parsa, D. Wang, C. S. O'Hern, M. D. Shattuck, R. Kramer-Bottiglio, and J. Bongard, "Evolving Programmable Computational Metamaterials," in *Proceedings of the Genetic and Evolutionary Computation Conference*, Jul. 2022, pp. 122–129. [Online]. Available: <http://arxiv.org/abs/2204.08651>
- [59] D. Przyczyna, M. Suchecki, A. Adamatzky, and K. Szaciłowski, "Computing with bricks and mortar: Classification of waveforms with a doped concrete blocks," *arXiv:2005.03498 [cs]*, May 2020. [Online]. Available: <http://arxiv.org/abs/2005.03498>
- [60] J. F. Miller, S. L. Harding, and G. Tufte, "Evolution-in-materio: Evolving computation in materials," *Evolutionary Intelligence*, vol. 7, no. 1, pp. 49–67, Apr. 2014. [Online]. Available: <https://doi.org/10.1007/s12065-014-0106-6>
- [61] 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.
- [62] 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. [Online]. Available: <http://www.oldcitypublishing.com/journals/ijuc-home/ijuc-issue-contents/ijuc-volume-10-numbers-5-6/ijuc-10-5-6-p-473-497/>
- [63] 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.
- [64] —, "Training a Carbon-Nanotube/Liquid Crystal Data Classifier Using Evolutionary Algorithms," in *Unconventional Computation and Natural Computa-*

- tion, ser. Lecture Notes in Computer Science, M. Amos and A. CONDON, Eds. Cham: Springer International Publishing, 2016, pp. 130–141.
- [65] J. W. Lawson and D. H. Wolpert, “Adaptive Programming of Unconventional Nano-Architectures,” *Journal of Computational and Theoretical Nanoscience*, vol. 3, no. 2, pp. 272–279, Apr. 2006.
- [66] M. Dale, J. F. Miller, S. Stepney, and M. A. Trefzer, “Evolving Carbon Nanotube Reservoir Computers,” in *Unconventional Computation and Natural Computation*, ser. Lecture Notes in Computer Science, M. Amos and A. CONDON, Eds. Cham: Springer International Publishing, 2016, pp. 49–61.
- [67] 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.
- [68] M. Dale, S. Stepney, J. Miller, and M. Trefzer, “Reservoir computing in materio: An evaluation of configuration through evolution,” *2016 IEEE Symposium Series on Computational Intelligence (SSCI)*, 2016.
- [69] Q.-Y. Zhu, A. K. Qin, P. N. Suganthan, and G.-B. Huang, “Evolutionary extreme learning machine,” *Pattern Recognition*, vol. 38, no. 10, pp. 1759–1763, Oct. 2005. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S0031320305001809>
- [70] M. Eshtay, H. Faris, and N. Obeid, “Metaheuristic-based extreme learning machines: A review of design formulations and applications,” *International Journal of Machine Learning and Cybernetics*, vol. 10, no. 6, pp. 1543–1561, Jun. 2019. [Online]. Available: <https://doi.org/10.1007/s13042-018-0833-6>
- [71] G.-B. Huang, Q.-Y. Zhu, and C.-K. Siew, “Extreme learning machine: Theory and applications,” *Neurocomputing*, vol. 70, no. 1, pp. 489–501, Dec. 2006. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S0925231206000385>
- [72] C. Bennett, D. Querlioz, and J.-O. Klein, “Spatio-Temporal learning with arrays of analog nanosynapses,” in *Proceedings of the IEEE/ACM International Symposium on Nanoscale Architectures, NANOARCH 2017*, 2017, pp. 125–130.
- [73] S. Ortín, M. C. Soriano, L. Pesquera, D. Brunner, D. San-Martín, I. Fischer, C. R. Mirasso, and J. M. Gutiérrez, “A Unified Framework for Reservoir Computing and Extreme Learning Machines based on a Single Time-delayed Neuron,” *Scientific Reports*, vol. 5, no. 1, p. 14945, Oct. 2015. [Online]. Available: <http://www.nature.com/articles/srep14945>
- [74] A. Lupo, L. Butschek, and S. Massar, “Photonic Extreme Learning Machine based on frequency multiplexing,” *Optics Express*, vol. 29, no. 18, p. 28257, Aug. 2021. [Online]. Available: <http://arxiv.org/abs/2107.04585>

- [75] 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. [Online]. Available: <http://www.sciencedirect.com/science/article/pii/S0893608019300784>
- [76] S. Kan, K. Nakajima, Y. Takeshima, T. Asai, Y. Kuwahara, and M. Akai-Kasaya, “Simple Reservoir Computing Capitalizing on the Nonlinear Response of Materials: Theory and Physical Implementations,” *Physical Review Applied*, vol. 15, no. 2, p. 024030, Feb. 2021. [Online]. Available: <https://link.aps.org/doi/10.1103/PhysRevApplied.15.024030>
- [77] C. Du, F. Cai, M. Zidan, W. Ma, S. Lee, and W. Lu, “Reservoir computing using dynamic memristors for temporal information processing,” *Nature Communications*, vol. 8, no. 1, 2017.
- [78] M. Dale, R. F. L. Evans, S. Jenkins, S. O’Keefe, A. Sebald, S. Stepney, F. Torre, and M. Trefzer, “Reservoir Computing with Thin-film Ferromagnetic Devices,” *arXiv:2101.12700 [cond-mat]*, Jan. 2021. [Online]. Available: <http://arxiv.org/abs/2101.12700>
- [79] P. Mujal, R. Martínez-Peña, J. Nokkala, J. García-Beni, G. L. Giorgi, M. C. Soriano, and R. Zambrini, “Opportunities in Quantum Reservoir Computing and Extreme Learning Machines,” *Advanced Quantum Technologies*, vol. 4, no. 8, p. 2100027, 2021. [Online]. Available: <http://onlinelibrary.wiley.com/doi/abs/10.1002/qute.202100027>
- [80] 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.

Chapter 2

Theory

2.1 Chapter Overview	21
2.2 Evolution in-Materio	22
2.3 Evolutionary Algorithms	24
2.4 Objective Functions	30
2.5 Feed Forward Artificial Neural Networks	33
2.6 AutoEncoders	40
2.7 Physical Reservoir Computing	42
Bibliography	45

2.1 Chapter Overview

This chapter introduces background knowledge and theory required to understand, formulate and develop Evolution in-Materio (EiM) processors. This includes defining an EiM system and representing its configuration as a mathematical solution which can be optimised. Three types of Evolutionary Algorithm (EA) are introduced, including Differential Evolution, OpenAI Evolutionary Strategy and Covariance Matrix Adaptation Evolution Strategy; all of which could be used to optimise EiM processors. Various objective functions are discussed, used to calculate a fitness or loss score, which an EA will attempt to improve.

Finally, Machine Learning (ML) methods such as Artificial Neural Networks (ANNs) and their training methods are also considered. This includes typical sin-

gle hidden layer feedforward neural networks (SLFNs), AutoEncoders (AEs), and Reservoir Computing (RC); concepts which are drawn from when developing more advanced and novel EiM processing devices.

2.2 Evolution in-Materio

As discussed in §1.4, EiM attempts to leverage a material’s (or medium’s) complex internal physical properties for computation. To achieve this, the selected material substrate must be operated as an ‘*in-materio*’ or ‘configurable analogue’ processor, where signals can be applied and read from the system, and the input-output transformation can be tuned using external stimuli.

Therefore, an *in-materio* processor’s behaviour is defined by a number of configurable parameters (e.g., voltage stimuli), which can be altered to improve the device’s performance. These configurable parameters can be grouped into a decision vector \mathbf{X} which defines the system’s configuration and represents a possible solution to a target task. An EiM processor uses an EA [1] to optimise the system by increasing the quality and performance of a solution \mathbf{X} , determined by an objective function Φ . EAs have been traditionally used to optimise *in-materio* systems, since the material substrates commonly used for computation are often hard to model and are therefore treated as a black box.

In this work, conductive nanomaterials and networks are considered to construct EiM processors performing classification, further details on the methods used are found in Chapter 3. In this case, input data signals are represented as input voltages \mathbf{V}^{in} . Similarly, output voltage states \mathbf{V}^{out} can be read from the *in-materio* processor and interpreted for the task at hand; such decision-making is referred to as the EiM processor’s interpretation scheme.

2.2.1 Traditional EiM Interpretation Schemes

When EiM is being used for classification applications, binary or otherwise, there must be some interpretation scheme. This allows the outputs to be assigned to a class or grouping. Often a decision is taken by using a classification rule [2],

commonly used in supervised learning. However, other methods of grouping data exist, such as a clustering algorithm [3] which is generally used for unsupervised learning. Here, some common classification rules which have been used for EiM systems are considered.

Classification Rules

A classification rule is used to assign a particular input data instance to a set of predefined classes. There are many ways to assign a class from a material's output states V^{out} or collected response. The most common classification rules include:

- i) **Output-Threshold Comparison.** This is when an output is gathered from the material and compared to a threshold, e.g., a single output voltage V^{out} :

$$\hat{y} = \begin{cases} 0, & \text{if } V^{out} < \tau \\ 1, & \text{otherwise} \end{cases}, \quad (2.2.1)$$

where \hat{y} is the predicted class and τ is a threshold which is either fixed or sometimes evolvable/can be optimised.

- ii) **Output-Output Comparison.** This is when a class is assigned by comparing two (or more) outputs from the material. For example, consider a system with two output voltages V_1^{out} & V_2^{out} :

$$\hat{y} = \begin{cases} 0, & \text{if } V_1^{out} < V_2^{out} \\ 1, & \text{otherwise} \end{cases}. \quad (2.2.2)$$

- iii) **kNN Algorithm.** The k Nearest Neighbour (kNN) algorithm tries to classify an unknown sample based on the known classification of its k number of neighbours [4]. This method requires forming a queue (i.e., storing) the training instances with their class labels. The nearest neighbours of new unlabelled data is determined by calculating the distance between it and all the instances in the queue. Commonly, the Euclidean distance is used as the distance metric. Some more advanced versions of kNN have been used in Unconventional

Computing (UC) situations, such as the kNN ball tree Algorithm [2] which was found to work well in classification problems using Single Walled Carbon Nanotube (SWCNT)/Poly(methyl methacrylate) (PMMA) EiM processors.

2.3 Evolutionary Algorithms

EAs are a subset of evolutionary computing, consisting of population-based meta-heuristics search algorithms, which take advantage of biologically inspired methods such as reproduction, mutation, recombination, and natural selection [1]. This makes them ideal when exploiting nanomaterials as *in-materio* devices, since they are analogue and generally lack an analytical model so have historically been treated as black boxes. Many types of EAs have been used for EiM such as Evolutionary Strategies [5], Genetic Algorithms [6] or Differential Evolution [7, 8, 9].

The purpose of the EA is to discover a vector of system parameters that achieves the best possible solution θ for a selected problem. The quality of a particular solution \mathbf{X} is determined by an objective function Φ , used to calculate a fitness score, as discussed in §2.4. Similarly, a population of solutions $\mathbf{p} = [\mathbf{X}_1, \mathbf{X}_2, \dots, \mathbf{X}_p]$ could each be evaluated. In summary, the EA optimises the system's available parameters to achieve the best possible fitness (sometimes referred to as a loss) score. In this section, three EAs used within the thesis are detailed.

2.3.1 Differential Evolution

Differential Evolution (DE) is an easily implemented and effective optimisation algorithm for exploiting real-world parameters [10]. DE is a derivative-free, stochastic, population-based, heuristic direct search method [11, 1] which only requires a few robust control variables [12]; these hyperparameters include a mutation factor F , a crossover CR and population size λ . A trial (i.e., child) population is formed via mutation and recombination of the parent population. These children are then evaluated and directly replace their parents if found to have a better 'fitness'.

The following describes how the DE algorithm is implemented, taken from [11] where further details are available. Each member of the population i is defined

by a d -dimensional vector of decision variables/parameters (sometimes known as a genome):

$$\mathbf{X}_{i,G} = [x_{1,i,G}, x_{2,i,G}, \dots, x_{d,i,G}] \quad \text{for } i = 1, 2, \dots, \lambda \quad (2.3.3)$$

where G is the particular generation, i is a member within a generation, and $x_{j,i,G}$ is a decision variable where $j \in 1, 2, \dots, d$. Population size λ does not change during the optimisation process. An objective function Φ is used to evaluate the fitness of any particular member of the population $\mathbf{X}_{i,G}$. The basic procedure to carry out DE is illustrated as follows:

Step 1: Produce initial population. The initial population, generation 0, is chosen randomly, with uniform probability, from the entire parameter space.

Step 2: Evaluate Population. Using the objective function, the fitness of every population member is evaluated. If any member of the population satisfies the problem (i.e., if termination criteria is met), then the algorithm terminates. Otherwise, the process continues.

Step 3: Mutation. For each target vector $\mathbf{X}_{i,G}$ (i.e., the current generation of ‘parents’) a mutant vector $\mathbf{V}_{i,G+1}$ is generated. This is achieved using mutation functions such as the random-1 mutation method (*rand1*):

$$\mathbf{V}_{i,G+1} = \mathbf{X}_{a,G} + F \times (\mathbf{X}_{b,G} - \mathbf{X}_{c,G}) , \quad (2.3.4)$$

where a , b and c are integer, mutually different, random indices from the range $\{1, 2, \dots, \lambda\}$, and F (also known as *mut*) is the differential weight or step size.

Other methods exist, such as the best-1 mutation method (*best1*):

$$\mathbf{V}_{i,G+1} = \mathbf{X}_{best,G} + F \times (\mathbf{X}_{a,G} - \mathbf{X}_{b,G}) , \quad (2.3.5)$$

where $\mathbf{X}_{best,G}$ is the best member of the target (parent) population, and a , b are random indices from the range $\{1, 2, \dots, \lambda\}$.

Once the mutant population has been created, the boundary constraints

must be considered to prevent any violations. Often, values are just *clipped* to the boundary limits $[l_j, m_j]$ for the j^{th} parameter variable, also known as projection [13, 14] :

$$v_{j,i,G+1} = \begin{cases} m_j, & \text{if } v_{j,i,G+1} > m_j \\ l_j, & \text{if } v_{j,i,G+1} < l_j \\ v_{j,i,G+1}, & \text{otherwise} \end{cases} . \quad (2.3.6)$$

Similarly, other methods exist [14] such as the *reflection* bounds constraint:

$$v_{j,i,G+1} = \begin{cases} 2m_j - v_{j,i,G+1}, & \text{if } v_{j,i,G+1} > m_j \\ 2l_j - v_{j,i,G+1}, & \text{if } v_{j,i,G+1} < l_j \\ v_{j,i,G+1}, & \text{otherwise} \end{cases} . \quad (2.3.7)$$

Step 4: **Crossover**. In order to increase the diversity of the perturbed parameter vectors, crossover is introduced. This acts as the recombination between the existing target (parent) population and the generated mutated population to produce a trial (child) population. Commonly, binomial (or uniform) crossover is utilised, where a trial vector:

$$\mathbf{U}_{i,G+1} = [u_{1,i,G+1}, u_{2,i,G+1}, \dots, u_{d,i,G+1}] , \quad (2.3.8)$$

is formed, where:

$$u_{j,i,G+1} = \begin{cases} v_{j,i,G+1}, & \text{if } r_j \leq CR \\ x_{j,i,G}, & \text{if } r_j > CR \end{cases} , \quad (2.3.9)$$

and r_j is the j^{th} evaluation of a random uniform number generator with outcome $\in [0, 1]$. CR is the Crossover Probability, also known as *crossp*, and is a constant $\in [0, 1]$ selected by the user. Often, a scheme is used such

that $\mathbf{U}_{i,G+1}$ always gets at least one parameter from $\mathbf{V}_{i,G+1}$, such as:

$$u_{j,i,G+1} = \begin{cases} v_{j,i,G+1}, & \text{if } r_j \leq CR \text{ or } j = q \\ x_{j,i,G}, & \text{if } r_j > CR \text{ and } j \neq q \end{cases}, \quad (2.3.10)$$

where q is a randomly chosen index $\in 0, 1, \dots, d$.

Step 5: Selection. Now, the trial vector $\mathbf{U}_{i,G+1}$ is compared to the target vector $\mathbf{X}_{i,G}$ to see whether it should become a member of generation $G + 1$. Whichever has the best fitness is retained and assigned to $\mathbf{X}_{i,G+1}$, this is repeated for every position i in the population.

Step 6: Repeat from step 2.

For a simpler explanation of the DE algorithm, a Pseudocode ¹ summary can be seen in Algorithm 2.1. Here, the algorithm is performed for I iterations before terminating. The data is generally split into two subsets (described further in §3.6.1), (i) *training* data which is used to update and train the population, and (ii) *test* data used to provide an unbiased evaluation of the model's performance. The best member of the population $\boldsymbol{\theta}$ is tracked, allowing a convergence plot of the population's best member's test and/or training fitness. Finally, the type of DE algorithm is often written in shorthand [10] such as stating the use of a 'DE/best/1/bin' algorithm (i.e., using DE, with a *best1* mutation scheme, and binomial crossover).

Algorithm 2.1: Pseudocode for basic DE.

```

Initialise a random population  $\mathbf{p}$ ;
Evaluate initial population fitnesses;
Assign the best member of  $\mathbf{p}$  as  $\boldsymbol{\theta}_0$ ;
for  $i = 0, 1, 2, \dots, I$  do
    Generate trial population  $\mathbf{t}$ ;
    Evaluate trial population fitnesses;
    Update population  $\mathbf{p}$  with respect to  $\mathbf{t}$ ;
    Update the best member  $\boldsymbol{\theta}_i$ ;

```

¹The algorithm pseudocode is meant to provide a less detailed but more understandable explanation of the algorithm.

2.3.2 OpenAI Evolutionary Strategy

Evolutionary Strategies (ESs) involve the evaluation of a population of real valued genotypes, after which the best members are kept, and others discarded [15, 1, 16]. Natural Evolution Strategy (NES) are a family of ESs which iteratively update a search distribution by using an estimated gradient on its distribution parameters [15]. NES calculate the fitness of a batch of search points, allowing the algorithm to capture the local structure of the fitness function. Using this the NES estimates a search gradient on the parameters towards a higher expected fitness. Notably, NES performs gradient ascent along the *natural gradient* [17, 15, 18] helping prevent oscillations, premature convergence, and undesired effects; unlike the plain gradient (detailed in §2.5.2).

In this section, the OpenAI Evolutionary Strategy (OpenAI-ES) algorithm [19] is outlined, which is a type of NES. The OpenAI-ES can be thought of as maintaining a single parent θ . During each iteration the parent is perturbed with Gaussian noise $\epsilon \sim \mathcal{N}(0, \sigma^2)$ to create a pseudo-population. Once evaluated, this pseudo-population is used to estimate the gradient:

$$\hat{g}_i = \frac{1}{\lambda\sigma} \sum_{j=1}^{\lambda} F_j \epsilon_j, \quad (2.3.11)$$

where σ is the standard deviation of Gaussian noise, λ is the size of the pseudo-population, ϵ_j is the noise used to create the j^{th} pseudo-population member, and F_j is the fitness of that member. The estimated gradient \hat{g} is then used to update the parent member [19]:

$$\theta_{i+1} = \theta_i + \alpha \hat{g}_i, \quad (2.3.12)$$

where α is the learning rate. The entire process is described in Algorithm 2.2. Here, the typical OpenAI-ES is altered to assign a lower fitness starting θ from an initial random population. In summary, the OpenAI-ES contains only three hyper-parameters: α , σ and λ .

The OpenAI-ES can rival Stochastic Gradient Descent (SGD) (introduced in §2.5.2) for deep reinforcement learning methods with large neural networks [19].

Algorithm 2.2: Pseudocode for OpenAI-ES.

Generate λ random starting solutions \mathbf{p} ;
 Evaluate initial starting solution fitnesses;
 Assign the best member of \mathbf{p} as $\boldsymbol{\theta}_0$;
for $i = 0, 1, 2, \dots, I$ **do**
 Sample $\boldsymbol{\epsilon}_1, \dots, \boldsymbol{\epsilon}_\lambda \sim \mathcal{N}(0, 1)$;
 Calculate fitness F_j of $(\boldsymbol{\theta}_i + \sigma \boldsymbol{\epsilon}_j)$ for $j = 0, 1, \dots, \lambda$;
 Set $\boldsymbol{\theta}_{i+1} = \boldsymbol{\theta}_i + \frac{\alpha}{\lambda \sigma} \sum_{j=1}^{\lambda} F_j \boldsymbol{\epsilon}_j$;

Further work has shown OpenAI-ES can offer significant speed up [20] compared to SGD, and highlights that the OpenAI-ES might be most useful in domains without perfect gradient information available – as in *in-materio* systems. Since the OpenAI-ES algorithm implements a complete version of SGD, improvements can be borrowed from traditional ANNs such as implementing Adaptive Moment Estimation (Adam) optimiser [21], a method further discussed in §2.5.2.

2.3.3 Covariance Matrix Adaptation ES

The Covariance Matrix Adaptation Evolution Strategy (CMA-ES) algorithm is a stochastic, or randomised, method for real-parameter (continuous domain) optimisation [22]. In the CMA-ES, a population of new search points is generated by sampling a Multivariate Gaussian Distribution (MGD). This MGD is adapted using the fitness results from a given generation. A sample solution \mathbf{X} is taken from the MGD [22] using:

$$\mathbf{X} = \mathbf{m}^{(i)} + \sigma^{(i)} \mathcal{N}(0, \mathbf{C}^{(i)}) , \quad (2.3.13)$$

where $\mathbf{m}^{(i)}$ is the distribution mean, $\mathbf{C}^{(i)}$ is the distribution’s covariance matrix, and $\sigma^{(i)}$ is the standard deviation or ‘step-size’ at generation i . The initial $\mathbf{C}^{(0)}$ is set as the identity, and the initial $\mathbf{m}^{(0)}$ is set to the centre of the boundary values. Therefore, the CMA-ES algorithm contains two hyperparameters: the initial standard deviation σ and population size λ . However, the population size is commonly automatically assigned as $\lambda = 4 + 3 \log(d)$.

An update during the CMA-ES is strongly related to the natural gradient descent [23, 24]. Indeed, CMA-ES and NES have been found to be special cases of one

Algorithm 2.3: Pseudocode for CMA-ES.

```

Initialise Multivariate Gaussian Distribution  $\mathcal{N}(\mathbf{m}, \sigma^2 \mathbf{C})$ ;
for  $i = 0, 1, 2, \dots, I$  do
    Sample population  $\mathbf{t}$  from  $\mathcal{N}(\mathbf{m}, \sigma^2 \mathbf{C})$ ;
    Evaluate trial population fitnesses;
    Update the best member  $\theta_{i+1}$ ;
    Update  $\sigma, \mathbf{m}, \mathbf{C}$  with respect to  $\mathbf{t}$ ;

```

another [25]. While full details can be found elsewhere [26], an overview of the method can be seen in the Pseudocode of Algorithm 2.3. In this work, the *cmaes* python package [27] is utilised, exploiting its ‘ask’ and ‘tell’ functionality to integrate it with existing code.

CMA-ES is a popular algorithm and overcomes some typical problems often associated with EAs such as the need to use large population sizes or premature convergence [28]. However, CMA-ES is generally applied to problems with dimensions of up to $d \leq 100$, beyond which it starts to slow down [29]. Methods exist to produce a diagonal and/or low-rank model of the covariance matrix, allowing it to be successfully applied to 500,000-dimensional (noise-free) problems [30].

2.4 Objective Functions

An objective function Φ is used to determine a fitness value for a particular set of system parameters \mathbf{X} , which represents a possible solution. For classification, this is achieved by considering dataset D containing K data instances, and performing some comparison between the dataset’s real labels and predicted outcomes. The datasets are normalised and scaled before being used to train the models, further described in §3.6.1. Therefore, an objective function allows an EA (or optimising algorithm) to evaluate its population members and make informed choices about how to improve its solutions. Commonly, the objective of the EA is thus to minimise the objective function. In this work, fitness is predominantly used to denote the quality of solution produced by a EA, whereas loss can denote the quality of solution produced by any algorithm; therefore, in this work fitness & loss are nominally interchangeable.

A wide variety of objective or ‘loss’ functions for classification problems exist. The objective function can have a large effect on the final solution of a classifier, something explored more in §5.4. In the following, some common objective functions used for classification are outlined.

2.4.1 Classification Error

Classification error is one of the simplest loss functions for classification problems. The classification error of a dataset D is simply:

$$\Phi_{error} = 1 - accuracy . \quad (2.4.14)$$

which can be found using:

$$\Phi_{error} = \frac{1}{K} \sum_{k=1}^K e(k) , \quad (2.4.15)$$

where k is an instance within the dataset which contains K data instances, and each data input instance produces an error value $e(k)$ of 0 or 1 for correct or incorrect classification respectively.

2.4.2 Mean Squared Error

A very popular loss function is the Mean Squared Error (mse) loss. The mse can be calculated using:

$$\Phi_{mse} = \frac{1}{K} \sum_{k=1}^K (y(k) - \hat{y}(k))^2 , \quad (2.4.16)$$

where $y(k)$ is the true output label/class and $\hat{y}(k)$ is a model’s predicted output for a particular data instance k .

Notably, ANNs systems deal with values and data which are often normalised or bound between $\in [0, 1]$. Physical systems, such as *in-materio* conductive substrates, are likely to operate with a larger range of real valued physical output voltage signals. To enable comparisons between different EiM devices, or even between EiM devices and an ANN equivalent (as done in §6.5) a Normalised Mean Squared Error ($nmse$) metric was developed in Eq.6.5.10.

2.4.3 Binary Cross Entropy

Binary Cross Entropy (BCE) or log loss is an established loss function for ML 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/classifier must be first constricted to $\in [0, 1]$. To accomplish this, a sigmoid function $\sigma(k)$ is used such that for a particular input data instance:

$$\sigma(k) = \frac{1}{1 + e^{-\hat{y}(k)}} . \quad (2.4.17)$$

The entropy or log loss $H(k)$ is defined as:

$$H(k) = \begin{cases} -\ln(1 - \sigma(k)), & \text{if } y(k) = 1 \\ -\ln(\sigma(k)), & \text{if } y(k) = 2 \end{cases} , \quad (2.4.18)$$

where $\ln()$ is the natural logarithm and we assume $y(k) = \{1, 2\}$ are the true labels associated with the data. Therefore, the adapted BCE objective function that the system attempts to minimise is defined as:

$$\Phi_{bce} = \frac{1}{K} \sum_{k=1}^K H(k) . \quad (2.4.19)$$

2.4.4 Cross Entropy

Cross Entropy (CE) can be applied to multi-class problems to predict a loss or fitness. In binary problems, each data instances results in a single predicted value \hat{y} . In multi-class problems, each data instance k results in a vector $\hat{\mathbf{y}}(k) = [\hat{y}_1, \hat{y}_2, \dots, \hat{y}_L]$ containing a predicted value $\hat{y}_l = \{0, 1\}$ for each of the L labels/classes. The multi-class log loss or ‘categorical’ cross-entropy loss can be calculated using:

$$\Phi = -\frac{1}{K} \sum_k \sum_l y_l(k) \ln(\hat{y}_l(k)) , \quad (2.4.20)$$

where l is a class $\in [1, 2, \dots, L]$.

As mentioned in the binary case above, it is often useful to consider the output predictions \hat{y} as a probability with values $\in [0, 1]$. To achieve this, the output predictions of a classification model can be ‘masked’ with a Softmax. The Softmax function behaves similarly to the sigmoid in §2.4.3, but is more applicable in multi-class problems, ensuring that the sum of the output predictions totals to a ‘probability’ of 1. The Softmax function is defined as:

$$\hat{s}_l(k) = \text{Softmax}(\hat{y}_l(k)) = \frac{\exp(\hat{y}_l(k))}{\sum_j \exp(\hat{y}_j(k))} , \quad (2.4.21)$$

where $\hat{s}_l(k)$ is the predicted probability for l^{th} class. These can then be applied to the log-loss function and used to calculate the CE as follows:

$$\Phi_{ce} = -\frac{1}{K} \sum_k \sum_l y_l(k) \ln(\hat{s}_l(k)) . \quad (2.4.22)$$

2.5 Feed Forward Artificial Neural Networks

Feed-forward Neural Networks (NNs) are one of the simplest forms of ANN used to evaluate non-temporal (static) data. They are made up of three types of layers: an input and an output layer, which are separated by one or more Hidden Layers (HLs). These layers consist of groups of neurons. The input layer contains special neurons that act as ‘sensor units’ which often only ‘detect’ the input features [31]. A HL consists of artificial neurons which accumulates signals from the input layer (or the outputs from the previous HL) and applies an activation, discussed further in §2.5.1. There can be any number of HLs, each containing one or more neurons. The outputs from the final HL are evaluated in the output layer which again contains artificial neuron(s). Generally, for binary classification only a single output is required, so only a single output neuron is needed; a binary classification decision is made based upon whether that signal exceeds a certain threshold or not. For multi-class classification, each class requires an individual output neuron; a classification decision is often made depending on which output neuron generates the largest output signal for the particular data instance. A diagram of a basic feed-forward NN structure is shown in Fig. 2.1.

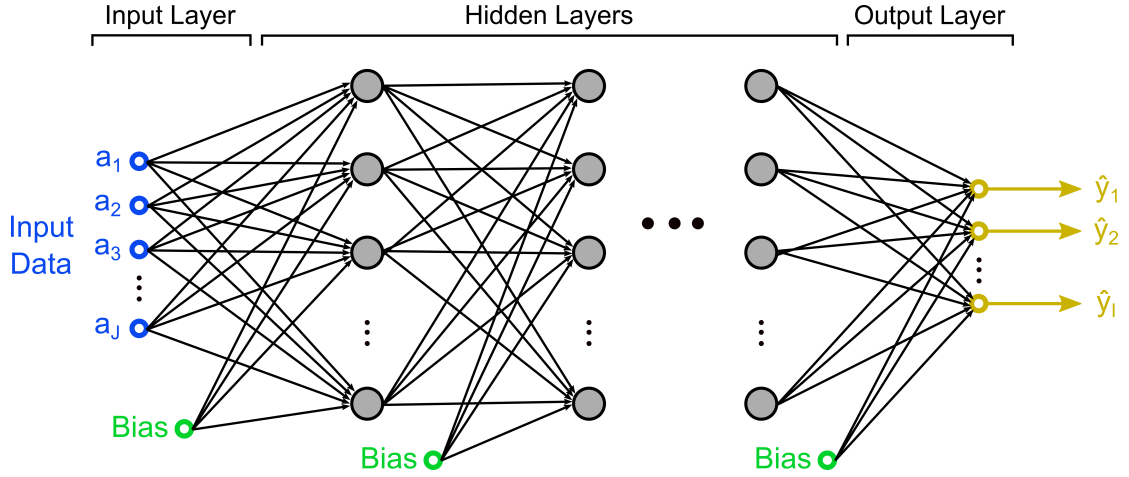


Figure 2.1: Typical structure of a Feed Forward Neural Network.

It is key to note that in feed forward networks there are no loops in the network - information is always fed forward, and never fed back [32]. Feed forward structures have been extensively studied for non-temporal problems, generally designed for multi-dimensional datasets where attributes are largely independent of one another. They are ideal for approximating nonlinear input-output functions. Feed forward ANNs are often referred to as *shallow* if they contain only a single HL and *deep* if they contain more than two HLs.

2.5.1 Artificial Neurons & Activation Functions

An artificial neuron usually consists of two parts. Firstly, a weighted summation of the previous layer's outputs and (sometimes) a bias, followed by an activation function. Therefore, for an input $\mathbf{x} = [x_1, x_2, \dots, x_J]$, the output of such a neuron is expressed as follows:

$$y = f(\mathbf{w} \cdot \mathbf{x} + b) = f\left(\sum_{j=1}^J w_j x_j + b\right) \quad (2.5.23)$$

where $\mathbf{w} = [w_1, w_2, \dots, w_J]^T$ is the weight vector, b is the bias, $f(x)$ is the activation function of the artificial neuron. The structure of such a typical artificial neuron is shown in Fig. 2.2.

The activation function is used to introduce non-linearity into the NN, which is essential to solve complex functions and datasets. Common activation functions in-

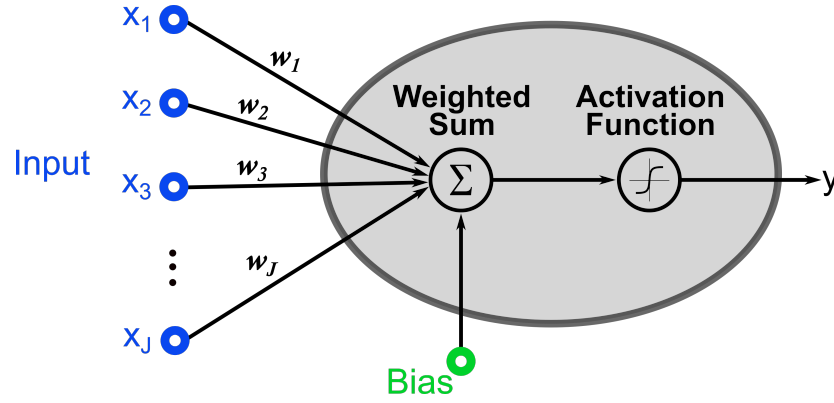


Figure 2.2: Example of an artificial neuron generating an output y from input signals $x_1, x_2, x_3, \dots, x_J$ and a bias.

clude unity (i.e., linear), ReLU, sigmoid and tanh, which are shown in Fig 2.3. The most common optimisation method for ANN is gradient descent with back propagation. Therefore, ANN research has often focused on neuron activation functions that are easily differentiable.

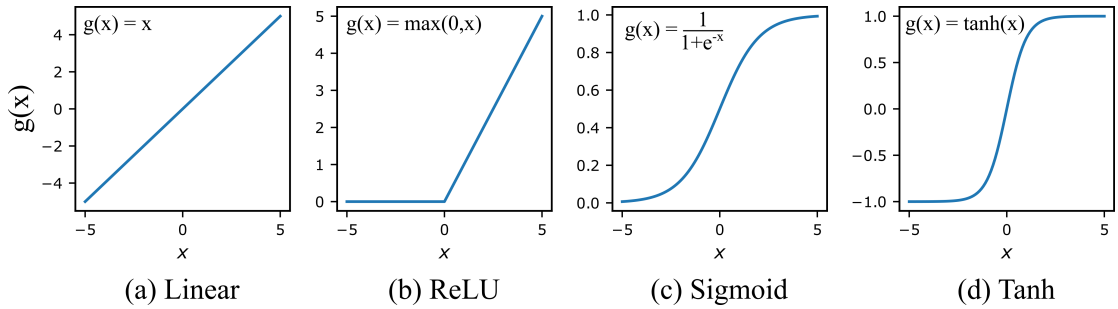


Figure 2.3: Some common activations functions for ANN artificial neurons.

2.5.2 Optimisation Methods

Traditional feed forward ANNs can be trained in a multitude of ways and with a variety of algorithms, from neuroevolution to back propagation. In the following, a few common ANNs training methods are briefly discussed.

Gradient Descent, mini-batching and Back Propagation

Gradient Descent (GD) is a method of updating a system's (e.g., an ANN) parameters θ by computing (or estimating) a gradient \mathbf{g} of the system's loss function Φ with respect to its parameters, and taking a step in the direction negative to that

gradient:

$$\boldsymbol{\theta}_{i+1} = \boldsymbol{\theta}_i - \alpha \mathbf{g}_i. \quad (2.5.24)$$

where α is the learning rate, a positive scalar used to select the step size. As such, GD is a first order optimisation method, only considering the first order derivatives of the loss function.

Originally, gradient descent algorithms used the entire training set, and its accumulation of gradient computations, before making an update to the system parameters [33]. The term “batch” or “mini-batch” gradient descent often refers to more modern methods where the training data is split into small groups (i.e., batches or minibatches), each of which are executed and used to update the system parameters. Originally, Stochastic Gradient Descent (SGD), sometimes called ‘online’, referred to methods where the parameters are updated based on a single training example (i.e., a single data instance) [34]. However, most algorithms now use more than one but fewer than all the training samples - these days simply referred to as stochastic methods [33]. Stochastic learning has been found to usually be much faster, especially when large datasets are being used. The act of sampling a smaller group of data (i.e., performing batching) introduces noise, often beneficially introducing some regularization [35].

Forward propagation is the process of applying inputs to a system and generating corresponding outputs. Back propagation (sometimes called *backprop*) allows information from the output’s ‘cost’ or loss to flow back through the system in order to compute the gradient [33, 36]. In other words, back propagation specifically refers to the efficient method of directly computing the gradient of the system’s loss function to enable GD. Other methods to calculate or estimate a gradient exist, such as the local perturbation method discussed in §2.3.2.

Momentum and Adam

Vanilla mini-batch GD, as described in Eq.(2.5.24), does not guarantee good convergence [37]. For example, selecting proper hyperparameters can be difficult, applying the same learning rate to all parameters can be inappropriate, and systems can often get stuck in local minimum - especially when surrounding gradients are close to zero

in all dimensions. The last of these is caused by typical GD being only a first order optimisation method, where the algorithm has no knowledge of the curvature of the loss function or ‘solution space’.

Momentum is a simple method to help mitigate some of these problems [38]. Essentially, momentum accumulates gradients of the past steps, helping accelerate the SGD in the relevant direction and dampen oscillations. This is achieved by adding a fraction γ of the previous update vector \mathbf{m}_{i-1} to the current step’s calculated \mathbf{g}_i :

$$\begin{aligned}\mathbf{m}_i &= \gamma \mathbf{m}_{i-1} + \alpha \mathbf{g}_i , \\ \boldsymbol{\theta}_{i+1} &= \boldsymbol{\theta}_i - \mathbf{m}_i ,\end{aligned}\tag{2.5.25}$$

where the initial value of the update vector \mathbf{m}_0 is set to zero. This effectively maintains an exponential moving average of the first order gradient.

Adam is a very popular ANN optimiser [21] which computes adaptive learning rates for each parameter. It maintains an exponential decaying average of past gradients \mathbf{m} (similar to momentum) and of the past squared gradients \mathbf{v} :

$$\begin{aligned}\mathbf{m}_t &= \beta_1 \mathbf{m}_{t-1} + (1 - \beta_1) \mathbf{g}_t , \\ \mathbf{v}_t &= \beta_2 \mathbf{v}_{t-1} + (1 - \beta_2) \mathbf{g}_t^2 ,\end{aligned}\tag{2.5.26}$$

where \mathbf{m}_t and \mathbf{v}_t are the estimates of the first moment (the mean) and the second moment (the uncentered variance) of the gradients respectively [37] and initialised as zero. The authors [21] observed that the first and second moment are biased towards zero, especially during the initial time steps and when the decay rates β_1 & β_2 small (i.e., they are close to 1). Therefore, a ‘bias corrected’ first $\hat{\mathbf{m}}$ and second $\hat{\mathbf{v}}$ moments are estimated to counteract this bias:

$$\begin{aligned}\hat{\mathbf{m}}_t &= \frac{\mathbf{m}_t}{1 - \beta_1^t} , \\ \hat{\mathbf{v}}_t &= \frac{\mathbf{v}_t}{1 - \beta_2^t} .\end{aligned}\tag{2.5.27}$$

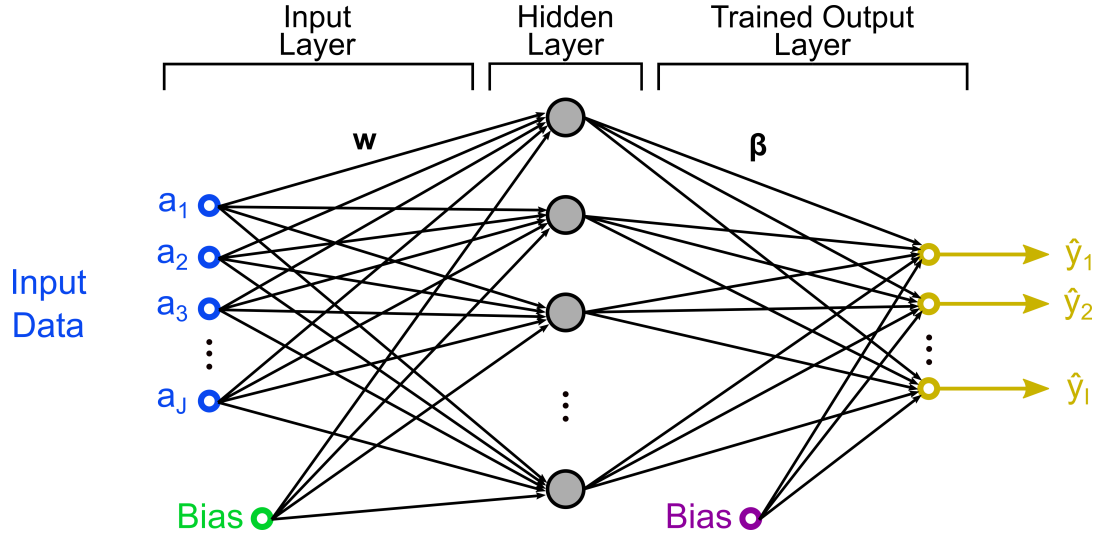


Figure 2.4: Basic structure of an artificial SLFN used as ELM.

These then are used to update the parameters using the Adam update rule:

$$\theta_{t+1} = \theta_t - \frac{\alpha}{\epsilon + \sqrt{\hat{v}_t}} \hat{m}_t . \quad (2.5.28)$$

The authors propose default values of $\beta_1 = 0.9$, $\beta_2 = 0.999$, and $\epsilon = 10^{-8}$. They show that Adam works well in practice and compares favourable to other adaptive learning-method algorithms.

Extreme Learning Machines

Extreme Learning Machines (ELMs) generally consist of a single hidden layer feed-forward neural network (SLFN), as seen in Fig. 2.4. They operate by assigning random weights and biases to the input and HIs respectively [39]. These parameters are fixed and remain unchanged during training. The only parameters learned are the weights (and sometimes biases) associated with the output layer, done during the training phase. Therefore, ELMs converge significantly faster than traditional ANN algorithms, such as back propagation. ELMs have been shown to perform well and are more likely to reach a global optimum than systems with networks which have all parameters trained [40]. Specifically, ELM systems achieve fast training speeds with good generalisation capability.

Consider a dataset with K data instances, where a particular data instance k is

defined by its inputs \mathbf{a}_k and its target outputs \mathbf{y}_k . A particular instance is defined by R attributes $\mathbf{a}_k = [a_1, a_2, \dots, a_R]^T$ used as the ELM's inputs, and its corresponding target containing Q outputs as $\mathbf{y}_k = [y_1, y_2, \dots, y_Q]^T$. The predicted outputs $\hat{\mathbf{y}}$ from an ELM with N hidden neurons can be expressed as:

$$\hat{\mathbf{y}}_k = \sum_{n=1}^N \beta_n f(\mathbf{w}_n \cdot \mathbf{a}_k + b_n) = \sum_{n=1}^N \beta_n h_{nk}, \quad k = 1, \dots, K \quad (2.5.29)$$

where $\mathbf{w}_n = [w_{n1}, w_{n2}, \dots, w_{nR}]^T$ is the weight vector connecting the n^{th} hidden neuron and the input neurons, $\beta_n = [\beta_{n1}, \beta_{n2}, \dots, \beta_{nQ}]^T$ is the weight vector connecting the n^{th} hidden neuron and the output neurons, b_n is the bias of the n^{th} hidden neuron, $f(x)$ is the activation function of the HL neurons, and h_{nk} is a HL neuron's output. A SLFN with enough hidden neurons can approximate these K samples such that $\sum_{n=1}^N \|\hat{\mathbf{y}}_n - \mathbf{y}_k\| = 0$ (universal approximation capability) [39], so there must be a set of β_n , \mathbf{w}_n and b_n such that [40]:

$$\sum_{n=1}^N \beta_n g(\mathbf{w}_n \cdot \mathbf{a}_k + b_n) = \sum_{n=1}^N \beta_n h_{kn} = \mathbf{y}_k, \quad k = 1, \dots, K \quad (2.5.30)$$

which can be rewritten more compactly as:

$$\mathbf{H}\beta = \mathbf{Y}, \quad (2.5.31)$$

where $\mathbf{H} = \{h_{nk}\}$ ($k = 1, \dots, K$ and $n = 1, \dots, N$) is the HL output matrix, $\mathbf{Y} = [\mathbf{y}_1, \mathbf{y}_2, \dots, \mathbf{y}_K]^T$ is the matrix of target outputs, and $\beta = [\beta_1, \beta_2, \dots, \beta_N]^T$ is the matrix of output weights.

Having randomised and fixed the input layer, the output layer is then learnt during training using training data (data sets discussed in §3.6.1). The output weights β are traditionally obtained by the Moore-Penrose inverse. Therefore, the smallest norm least-squares solution is:

$$\hat{\beta} = \mathbf{H}^\dagger \mathbf{Y}, \quad (2.5.32)$$

where \mathbf{H}^\dagger is the Moore-Penrose inverse of matrix \mathbf{H} . The final solution is then

tested on the test set to provide an unbiased evaluation of the system.

Often, many randomly initialised networks are considered, and the network size is incrementally increased. Various methods of optimising the output layer, adjusting network structure, and increasing convergence speed have been proposed [40]. Specifically, work producing an RR-ELM (Ridge Regressed Extreme Learning Machine) algorithm [41] is highlighted, which optimises the output layer using ridge regression rather than the Moore-Penrose method described above. The RR-ELM algorithm is shown to have good generalisation and stability, while also reducing adverse effects caused by perturbation or multicollinearity - properties likely to be useful in physical systems.

Neuroevolution

The term neuroevolution [42, 43, 44] is used to describe the training of a ANN using an EA. This can be the sole optimisation method applied to the system, or can be used in combination with other algorithms, such as the input weights [45] or activation function [46] of a NN being tuned by an EA while the output layer is trained as an ELM.

2.6 AutoEncoders

An AutoEncoder (AE) is typically created using an ANN and are trained in an unsupervised manner. AEs are formed of two parts, an encoder which transforms the input data into a new set of features (i.e., the latent space representation), and a decoder which attempts to reconstruct the original input data using the encoded data. The most basic type of AE, depicted in Fig. 2.5, consists of three layers: an input layer, a single HL, and an output layer. An AE's output layer always contains the same number of neurons as the input layer so that it can properly reconstruct the input data. Larger AEs typically contain odd numbers of HLs, to ensure a symmetrical structure.

AEs can have several uses depending on the network structure. If a constriction or "bottleneck" is introduced to the network, then the latent space will produce a new

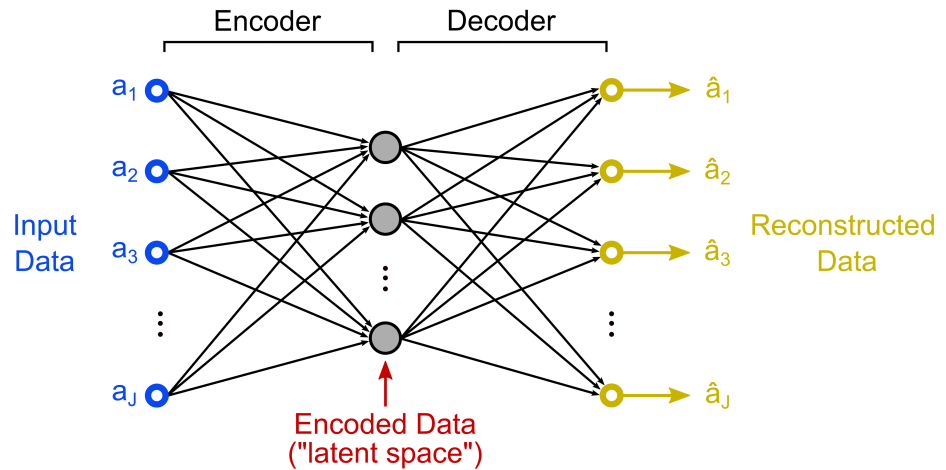


Figure 2.5: Structure of a basic AE constructed using an ANN. The encoder and decoder can consist of many HLs, but the network structure is typically symmetrical.

‘compressed’ set of features from the original data, known as dimensionality reduction. The ability to reduce the number of input variables has a wide range of applications, including simplifying classification datasets and better data visualisation, such as representing the MNIST (handwritten numbers) dataset in two dimensions ($\mathbb{R}^{784} \Rightarrow \mathbb{R}^2$) [47]. AEs are also commonly used for unsupervised pre-training of deep NNs [48, 49]. Other applications include data (e.g., image) de-noising and anomaly detection.

Generally, an optimising algorithm will attempt to minimise the reconstruction error, or similar loss function, such as Mean Squared Error. Commonly, AEs are trained using gradient descent and back propagation [48, 49, 50], but neuroevolution is an alternative method which has been shown to be successful [51, 52, 53]. Indeed, it can have several advantages over back propagation, such as EAs performing global searches that help avoid getting stuck at local minima [53], or by allowing the evolution of hyperparameters and even network structure [54, 52]. Perhaps most interestingly is the speculation that good performance achieved by a Genetic Algorithm based AEs might arise from a mutation randomly disabling some of the weights during training [51] similar to dropout [55] (randomly disabling some neurons during training), dropconnect [56] (randomly disabling some weights during training) or de-noising AEs [49] (which attempt to reconstruct corrupted input data). A third, less common method for training AEs is achieved by considering them as ELMs [57].

Evaluating the encoded features generated by an AEs is not always simple. As a

minimum, when performing dimensionality reduction, the encoded features should outperform Principal Component Analysis (PCA), a well-known dimensionality-reduction technique [58]. To measure performance, an unsupervised clustering algorithm [59] can be used to cluster the AE or PCA generated features. Then, the quality of the clusters can be analysed in a supervised manner using the following metrics:

- i) Clustering Accuracy (CA), which is the accuracy of the best match between the class labels and the cluster labels [60]. This can be defined as:

$$accuracy(y, \hat{y}) = \max_{perm \in P} \frac{1}{K} \sum_k^K (perm(\hat{y}_i) = y_i) , \quad (2.6.33)$$

where P is the set of all permutations when assigning labels to the clusters.

- ii) Adjusted Rand Index (ARI), which is a metric that computes a similarity between two clustering results by considering all pairs of samples and counting pairs that are assigned in the same or different clusters in the predicted and ground truth clustering results [59, 61], which is adjusted against chance. This similarity score can be calculated using sklearn [61] and takes a value $\in [-0.5, 1.0]$; random labellings have an ARI close to 0, and a value of 1 represents a perfect match. It has been shown that ARI should be used in cases where clusters are similarly sized [62].

2.7 Physical Reservoir Computing

Reservoir Computing (RC) is a computational framework suited for temporal/sequential data processing. RC was first used as a method to train Recurrent Neural Networks (RNNs), but it presents a method by which any high dimensional dynamic system with the right dynamic properties can be used as a temporal ‘kernel’, ‘basis’ or ‘code’ to pre-process the data such that it can be easily processed using linear techniques [63]. Indeed, as discussed by Tanaka et al [64], physical mediums can be exploited as reservoirs within the RC paradigm. Examples of reservoirs used in physical RC include analogue circuits, Field-Programmable Gate Arrays (FPGAs),

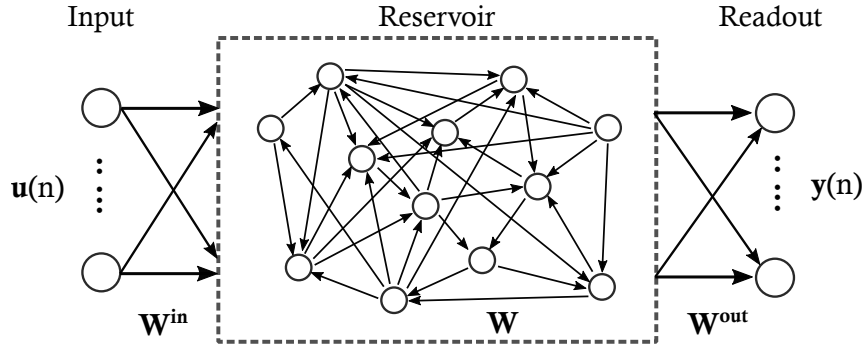


Figure 2.6: A conventional RC system with a RNN-based reservoir, where only the readout weights W^{out} are trained, and the input weights W^{in} & internal reservoir weights W are fixed.

memristive devices, optical node arrays and even brain regions [64]. It is important to consider how RC operates and how methods for physical RC implementations might overlap with EiM processors.

The main characteristic of the original, ANN based RC method is that the input weights and the weights inside the reservoir are *not trained*, only the weights of a readout layer are trained, usually with a simple learning algorithm such as linear regression [64]. The purpose of the reservoir is to nonlinearly transform sequential inputs into a new, often high dimensional space such that features from the input data can be efficiently read out by the simple learning algorithm. Consider the generic reservoir shown in Fig. 2.6, at any discrete time n there will be a number of inputs defined by the input vector $\mathbf{u}(n)$, and also a corresponding output state vector of the reservoir units $\mathbf{x}(n)$ which are read out from the reservoir. Any nonlinear dynamic system could be used instead of RNNs.

Reservoir computing models can be derived from the sub-fields of Echo State Networks (ESNs) and Liquid State Machines (LSMs), each of which details a separate approach to perform RC, and which are briefly outlined here. The ESN model was proposed by Jaeger [65] and defines the states of a reservoir as:

$$\mathbf{x}(n) = f(W^{in}\mathbf{u}(n) + W\mathbf{x}(n-1) + W^{fb}\mathbf{y}(n-1)) \quad , \quad (2.7.34)$$

where W^{in} is the weight matrix for the reservoir-input connections, W is the weight matrix for the recurrent (internal) reservoir connections. If feedback from the out-

puts of the reservoir $\mathbf{y}(n)$ are used, then Eq.(2.7.34) also includes a weight matrix for the feedback W^{fb} . The function f represents the behaviour of the reservoir. The trained output is commonly given by a linear combination of the reservoir states with the trained output weight matrix W^{out} :

$$\mathbf{y}(n) = W^{out} \mathbf{x}(n) . \quad (2.7.35)$$

The performance of the ESN depends on the reservoir, which must have the echo state property.

The LSM model was proposed by Maass et al. [66]. Its purpose is to develop biologically relevant learning models using Spiking Neural Networks (SNNs) with recurrent connectivity. It uses *liquid* reservoirs, where the probability that two neurons are connected depends on the distance between them. In order for a machine M to map input functions of time $u(\cdot)$ to output functions $y(\cdot)$, we assume that it contains some "liquid" that generates, at every time t some internal states $x^M(t)$:

$$x^M(t) = (L^M u)(t), \quad (2.7.36)$$

where L^M is the filter for transforming the input into the reservoir state. The output $y(t)$ is given by:

$$y(t) = f^M(x^M(t)), \quad (2.7.37)$$

where f^M is a memory-less readout map (i.e., not required to retain any memory of previous states). A simple ML algorithm can be used to train the readout map such that a target output function/sequence $y(\cdot)$ can be achieved.

One driving force behind conventional RC is that a particular RNN reservoir might perform well without the need for extensive conventional training. This led to a technique which could be successfully applied to physical reservoirs that are often difficult to model and treated as black boxes. As discussed in §1.3 & 1.4, using a physical analogue system might have benefits, such as possible low-power and efficient computing. Notably, the RC paradigm often limits itself to a fixed reservoir

– this being one of the major benefits when training artificial RNNs. However, physical reservoirs might in fact benefit from tuning similarly to EiM processors.

Bibliography

- [1] A. N. Sloss and S. Gustafson, “2019 Evolutionary Algorithms Review,” *arXiv:1906.08870 [cs]*, Jun. 2019. [Online]. Available: <http://arxiv.org/abs/1906.08870>
- [2] F. QAISER, “Training Single Walled Carbon Nanotube based Materials to perform computation,” Doctoral, Durham University, 2018. [Online]. Available: <http://etheses.dur.ac.uk/12893/>
- [3] “Data Mining (Third Edition),” in *Data Mining (Third Edition)*, ser. The Morgan Kaufmann Series in Data Management Systems, J. Han, M. Kamber, and J. Pei, Eds. Boston: Morgan Kaufmann, Jan. 2012. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/B9780123814791000162>
- [4] A. Mucherino, P. J. Papajorgji, and P. M. Pardalos, “K-Nearest Neighbor Classification,” in *Data Mining in Agriculture*, ser. Springer Optimization and Its Applications, A. Mucherino, P. J. Papajorgji, and P. M. Pardalos, Eds. New York, NY: Springer, 2009, pp. 83–106. [Online]. Available: https://doi.org/10.1007/978-0-387-88615-2_4
- [5] 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.
- [6] 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. [Online]. Available: <http://www.nature.com/articles/nnano.2015.207>
- [7] 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. [Online]. Available: <http://ieeexplore.ieee.org/document/8123677/>
- [8] 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. [Online]. Available: <http://www.nature.com/articles/srep32197>

- [9] B. A. H. Jones, J. L. P. Chouard, B. C. C. Branco, E. G. B. Vissol-Gaudin, C. Pearson, M. C. Petty, N. Al Moubayed, D. A. Zeze, and C. Groves, “Towards Intelligently Designed Evolvable Processors,” *Evolutionary Computation*, pp. 1–23, Mar. 2022. [Online]. Available: https://doi.org/10.1162/evco_a_00309
- [10] 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.
- [11] 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. [Online]. Available: <https://doi.org/10.1023/A:1008202821328>
- [12] M. E. H. Pedersen, “Good Parameters for Differential Evolution,” 2010.
- [13] J. Brest, S. Greiner, B. Boskovic, M. Mernik, and V. Zumer, “Self-Adapting Control Parameters in Differential Evolution: A Comparative Study on Numerical Benchmark Problems,” *IEEE Transactions on Evolutionary Computation*, vol. 10, no. 6, pp. 646–657, Dec. 2006.
- [14] V. Kreischer, T. Magalhães, H. Barbosa, and E. Krempser, “Evaluation of Bound Constraints Handling Methods in Differential Evolution using the CEC2017 Benchmark,” 2018.
- [15] D. Wierstra, T. Schaul, J. Peters, and J. Schmidhuber, “Natural Evolution Strategies,” in *2008 IEEE Congress on Evolutionary Computation (IEEE World Congress on Computational Intelligence)*. Hong Kong, China: IEEE, Jun. 2008, pp. 3381–3387. [Online]. Available: <http://ieeexplore.ieee.org/document/4631255/>
- [16] P. A. Vikhar, “Evolutionary algorithms: A critical review and its future prospects,” in *2016 International Conference on Global Trends in Signal Processing, Information Computing and Communication (ICGTSPICC)*, Dec. 2016, pp. 261–265.
- [17] S.-i. Amari, “Natural Gradient Works Efficiently in Learning,” *Neural Computation*, vol. 10, no. 2, pp. 251–276, Feb. 1998. [Online]. Available: <https://doi.org/10.1162/089976698300017746>
- [18] J. Martens, “New insights and perspectives on the natural gradient method,” Sep. 2020. [Online]. Available: <http://arxiv.org/abs/1412.1193>
- [19] T. Salimans, J. Ho, X. Chen, S. Sidor, and I. Sutskever, “Evolution Strategies as a Scalable Alternative to Reinforcement Learning,” Sep. 2017. [Online]. Available: <http://arxiv.org/abs/1703.03864>
- [20] X. Zhang, J. Clune, and K. O. Stanley, “On the Relationship Between the OpenAI Evolution Strategy and Stochastic Gradient Descent,” Dec. 2017. [Online]. Available: <http://arxiv.org/abs/1712.06564>

- [21] D. P. Kingma and J. Ba, “Adam: A Method for Stochastic Optimization,” *arXiv*, Jan. 2017. [Online]. Available: <http://arxiv.org/abs/1412.6980>
- [22] N. Hansen, “The CMA Evolution Strategy: A Tutorial,” *CoRR*, vol. abs/1604.00772, 2016. [Online]. Available: <http://arxiv.org/abs/1604.00772>
- [23] M. Nomura, S. Watanabe, Y. Akimoto, Y. Ozaki, and M. Onishi, “Warm Starting CMA-ES for Hyperparameter Optimization,” Dec. 2020. [Online]. Available: <http://arxiv.org/abs/2012.06932>
- [24] Y. Ollivier, L. Arnold, A. Auger, and N. Hansen, “Information-Geometric Optimization Algorithms: A Unifying Picture via Invariance Principles,” Apr. 2017. [Online]. Available: <http://arxiv.org/abs/1106.3708>
- [25] Y. Akimoto, Y. Nagata, I. Ono, and S. Kobayashi, “Bidirectional Relation between CMA Evolution Strategies and Natural Evolution Strategies,” in *Parallel Problem Solving from Nature, PPSN XI*, ser. Lecture Notes in Computer Science, R. Schaefer, C. Cotta, J. Kołodziej, and G. Rudolph, Eds. Berlin, Heidelberg: Springer, 2010, pp. 154–163.
- [26] V.-H. Dang, N. A. Vien, and T. Chung, “A covariance matrix adaptation evolution strategy in reproducing kernel Hilbert space,” *Genetic Programming and Evolvable Machines*, vol. 20, no. 4, pp. 479–501, Dec. 2019. [Online]. Available: <https://doi.org/10.1007/s10710-019-09357-1>
- [27] M. Shibata, “Cmaes: Lightweight Covariance Matrix Adaptation Evolution Strategy (CMA-ES) implementation for Python 3.” [Online]. Available: <https://github.com/CyberAgentAILab/cmaes>
- [28] N. Hansen, “The CMA Evolution Strategy: A Tutorial,” Apr. 2016. [Online]. Available: <http://arxiv.org/abs/1604.00772>
- [29] N. Müller and T. Glasmachers, “Challenges in High-Dimensional Reinforcement Learning with Evolution Strategies,” in *Parallel Problem Solving from Nature – PPSN XV*, ser. Lecture Notes in Computer Science, A. Auger, C. M. Fonseca, N. Lourenço, P. Machado, L. Paquete, and D. Whitley, Eds. Cham: Springer International Publishing, 2018, pp. 411–423.
- [30] I. Loshchilov, “A computationally efficient limited memory CMA-ES for large scale optimization,” in *Proceedings of the 2014 Annual Conference on Genetic and Evolutionary Computation*, ser. GECCO ’14. New York, NY, USA: Association for Computing Machinery, Jul. 2014, pp. 397–404. [Online]. Available: <https://doi.org/10.1145/2576768.2598294>
- [31] R. M. Golden, “Artificial Neural Networks: Neurocomputation,” in *International Encyclopedia of the Social & Behavioral Sciences*, N. J. Smelser and P. B. Baltes, Eds. Oxford: Pergamon, Jan. 2001, pp. 806–811. [Online]. Available: <http://www.sciencedirect.com/science/article/pii/B0080430767005635>

- [32] M. A. Nielsen, “Neural Networks and Deep Learning,” 2015. [Online]. Available: <http://neuralnetworksanddeeplearning.com>
- [33] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016. [Online]. Available: <https://www.deeplearningbook.org/>
- [34] D. Masters and C. Lusch, “Revisiting Small Batch Training for Deep Neural Networks,” *arXiv:1804.07612 [cs, stat]*, Apr. 2018. [Online]. Available: <http://arxiv.org/abs/1804.07612>
- [35] Y. Bengio, “Practical Recommendations for Gradient-Based Training of Deep Architectures,” in *Neural Networks: Tricks of the Trade: Second Edition*, ser. Lecture Notes in Computer Science, G. Montavon, G. B. Orr, and K.-R. Müller, Eds. Berlin, Heidelberg: Springer, 2012, pp. 437–478. [Online]. Available: https://doi.org/10.1007/978-3-642-35289-8_26
- [36] Y. A. LeCun, L. Bottou, G. B. Orr, and K. R. Müller, “Efficient backprop,” in *Neural Networks*, ser. Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics). Springer Verlag, 2012, pp. 9–48. [Online]. Available: <http://www.scopus.com/inward/record.url?scp=84872543023&partnerID=8YFLogxK>
- [37] S. Ruder, “An overview of gradient descent optimization algorithms,” Jun. 2017. [Online]. Available: <http://arxiv.org/abs/1609.04747>
- [38] N. Qian, “On the momentum term in gradient descent learning algorithms,” *Neural Networks*, vol. 12, no. 1, pp. 145–151, Jan. 1999. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S0893608098001166>
- [39] G.-B. Huang, Q.-Y. Zhu, and C.-K. Siew, “Extreme learning machine: Theory and applications,” *Neurocomputing*, vol. 70, no. 1, pp. 489–501, Dec. 2006. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S0925231206000385>
- [40] J. Wang, S. Lu, S.-H. Wang, and Y.-D. Zhang, “A review on extreme learning machine,” *Multimedia Tools and Applications*, May 2021. [Online]. Available: <https://doi.org/10.1007/s11042-021-11007-7>
- [41] G. Li and P. Niu, “An enhanced extreme learning machine based on ridge regression for regression,” *Neural Computing and Applications*, vol. 22, no. 3, pp. 803–810, Mar. 2013. [Online]. Available: <https://doi.org/10.1007/s00521-011-0771-7>
- [42] K. O. Stanley, J. Clune, J. Lehman, and R. Miikkulainen, “Designing neural networks through neuroevolution,” *Nature Machine Intelligence*, vol. 1, no. 1, pp. 24–35, Jan. 2019. [Online]. Available: <http://www.nature.com/articles/s42256-018-0006-z>
- [43] E. Galván and P. Mooney, “Neuroevolution in Deep Neural Networks: Current Trends and Future Challenges,” *IEEE Transactions on Artificial*

- Intelligence*, vol. 2, no. 6, pp. 476–493, Dec. 2021. [Online]. Available: <http://arxiv.org/abs/2006.05415>
- [44] M. Baiocchi, G. Di Bari, A. Milani, and V. Poggioni, “Differential Evolution for Neural Networks Optimization,” *Mathematics*, vol. 8, no. 1, p. 69, Jan. 2020. [Online]. Available: <https://www.mdpi.com/2227-7390/8/1/69>
- [45] T. Matias, R. Araújo, C. H. Antunes, and D. Gabriel, “Genetically optimized extreme learning machine,” in *2013 IEEE 18th Conference on Emerging Technologies Factory Automation (ETFA)*, Sep. 2013, pp. 1–8.
- [46] B. Li, Y. Li, and X. Rong, “The extreme learning machine learning algorithm with tunable activation function,” *Neural Computing and Applications*, vol. 22, no. 3, pp. 531–539, Mar. 2013. [Online]. Available: <https://doi.org/10.1007/s00521-012-0858-9>
- [47] Y. Wang, H. Yao, and S. Zhao, “Auto-encoder based dimensionality reduction,” *Neurocomputing*, vol. 184, pp. 232–242, Apr. 2016. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S0925231215017671>
- [48] Y. Bengio, P. Lamblin, D. Popovici, H. Larochelle, and U. Montreal, “Greedy layer-wise training of deep networks,” in *Advances in Neural Information Processing Systems*, vol. 19, Jan. 2007.
- [49] P. Vincent, H. Larochelle, I. Lajoie, Y. Bengio, and P.-A. Manzagol, “Stacked Denoising Autoencoders: Learning Useful Representations in a Deep Network with a Local Denoising Criterion,” *The Journal of Machine Learning Research*, vol. 11, pp. 3371–3408, Dec. 2010.
- [50] H. Wang, K. Ren, and J. Song, “A closer look at batch size in mini-batch training of deep auto-encoders,” in *2017 3rd IEEE International Conference on Computer and Communications (ICCC)*, Dec. 2017, pp. 2756–2761.
- [51] E. David and I. Greental, “Genetic Algorithms for Evolving Deep Neural Networks,” *Proceedings of the Companion Publication of the 2014 Annual Conference on Genetic and Evolutionary Computation*, pp. 1451–1452, Jul. 2014. [Online]. Available: <http://arxiv.org/abs/1711.07655>
- [52] Q. S. U. Khan, J. Li, and S. Zhao, “Training Deep Autoencoder via VLC-Genetic Algorithm,” *Neural Information Processing*, pp. 13–22, Nov. 2017. [Online]. Available: https://doi.org/10.1007/978-3-319-70096-0_2
- [53] H. Okada, “Neuroevolution of Autoencoders by Genetic Algorithm,” *International Journal of Science and Engineering Investigations*, vol. 6, no. 6, pp. 127–131, 2017. [Online]. Available: <https://arastirmax.com/en/publication/international-journal-science-and-engineering-investigations/6/6/127-131-neuroevolution-autoencoders-genetic-algorithm/arid/a1bf7da7-9381-4772-8087-231ad22df6df>

- [54] S. Lander and Y. Shang, “EvoAE – A New Evolutionary Method for Training Autoencoders for Deep Learning Networks,” in *2015 IEEE 39th Annual Computer Software and Applications Conference*, vol. 2, Jul. 2015, pp. 790–795.
- [55] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. R. Salakhutdinov, “Improving neural networks by preventing co-adaptation of feature detectors,” Jul. 2012. [Online]. Available: <http://arxiv.org/abs/1207.0580>
- [56] L. Wan, M. Zeiler, S. Zhang, Y. L. Cun, and R. Fergus, “Regularization of Neural Networks using DropConnect,” in *Proceedings of the 30th International Conference on Machine Learning*, vol. 28. PMLR, May 2013, pp. 1058–1066. [Online]. Available: <https://proceedings.mlr.press/v28/wan13.html>
- [57] K. Sun, J. Zhang, C. Zhang, and J. Hu, “Generalized extreme learning machine autoencoder and a new deep neural network,” *Neurocomputing*, vol. 230, pp. 374–381, Mar. 2017. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S092523121631503X>
- [58] E. Plaut, “From Principal Subspaces to Principal Components with Linear Autoencoders,” *arXiv:1804.10253 [cs, stat]*, Dec. 2018. [Online]. Available: <http://arxiv.org/abs/1804.10253>
- [59] S. Lu and R. Li, “DAC: Deep Autoencoder-based Clustering, a General Deep Learning Framework of Representation Learning,” *arXiv:2102.07472 [cs]*, Feb. 2021. [Online]. Available: <http://arxiv.org/abs/2102.07472>
- [60] “Coclust.evaluation.external — Coclust 0.2.1 documentation.” [Online]. Available: https://coclust.readthedocs.io/en/v0.2.1/_modules/coclust/evaluation/external.html?highlight=accuracy#
- [61] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and É. Duchesnay, “Scikit-learn: Machine Learning in Python,” *Journal of Machine Learning Research*, vol. 12, no. 85, pp. 2825–2830, 2011. [Online]. Available: <http://jmlr.org/papers/v12/pedregosa11a.html>
- [62] S. Romano, N. X. Vinh, J. Bailey, and K. Verspoor, “Adjusting for Chance Clustering Comparison Measures,” *Journal of Machine Learning Research*, vol. 17, no. 134, pp. 1–32, 2016. [Online]. Available: <http://jmlr.org/papers/v17/15-627.html>
- [63] B. Schrauwen, D. Verstraeten, and J. Van Campenhout, “An overview of reservoir computing: Theory, applications and implementations,” *Proceedings of the 15th European Symposium on Artificial Neural Networks. p. 471-482 2007*, pp. 471–482, 2007. [Online]. Available: <http://hdl.handle.net/1854/LU-416607>
- [64] 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. [Online]. Available: <http://www.sciencedirect.com/science/article/pii/S0893608019300784>
- [65] H. Jaeger, “The “Echo State” Approach to Analysing and Training Recurrent Neural Networks,” *GMD-Report 148*, German National Research Institute for Computer Science, Jan. 2001.
- [66] W. Maass, T. Natschläger, and H. Markram, “Real-time computing without stable states: A new framework for neural computation based on perturbations,” *Neural Computation*, vol. 14, no. 11, pp. 2531–2560, 2002.

Chapter 3

Problem Formulation

3.1 Chapter Overview	52
3.2 Configurable Analogue Processor	53
3.3 EiM for Classification	55
3.4 Simulated CAP Model	57
3.5 Physical Experimentation	60
3.6 Datasets	65
Bibliography	70

3.1 Chapter Overview

Having established background knowledge on Evolutionary Algorithms (EAs), objective functions, and other wider theory, this chapter considers how a material (or medium) can be leveraged as an Evolution in-Materio (EiM) processor. Specifically, this chapter focuses on how a conductive nanomaterial substrate can be exploited as a Configurable Analogue Processor (CAP) (also referred to as a material or *in-materio* processor) and be formulated as a EiM device used for classification. Previously, the literature has used a wide range of methods and system setups. The developed EiM processor in this work presents a standardised framework, allowing for consistent and reliable investigation throughout this thesis and for future researchers.

An obstacle for the development of EiM devices has typically been the slow

fabrication and experimentation of physical devices. To overcome this, an electrical model combining python and Simulation Program with Integrated Circuit Emphasis (SPICE) was produced¹, enabling conductive networks to be simulated and leveraged as a proxy for physical CAPs. This allowed for fast, efficient *in-simulo* experimentation into the EiM computing paradigm.

Lab-based experimentation is also performed in this thesis, which required the production of a Hardware Interface (HI) to apply and read voltage signals to a conductive network or nanomaterial substrate. The system used a Raspberry Pi to host an optimising EA and execute the experimental procedures. A custom Software Interface (SI) was produced¹ to convert higher level python functions into lower level hardware commands. This physical test platform was used to exploit a complex Lambda Diode Network as a computational resource, as seen in chapter 6.4. These Lambda Diodes are similar to tunnel diodes, with a non-monotonically increasing Current-Voltage (IV) characteristic, presenting interesting properties which can be leveraged, similar to dopant atom networks [1, 2].

Finally, some synthetic and real datasets are introduced. These were used to benchmark EiM processors, new novel *in-materio* device structures, and other classification methods throughout the thesis.

3.2 Configurable Analogue Processor

A CAP, also referred to interchangeably as a material or *in-materio* processor, operates by some material (or medium) projecting input signals to a new, often higher dimensional, representation. This transformation can be configured by tuning stimuli signals or other system parameters. This work focuses on conductive networks or nanomaterials which operate as the material processor within an EiM system. While systems could be selected to be configured using a variety of different external stimuli, such as light [3], vibrations/acoustics [4] or radio waves [5], research has often focused on substrates which can be interacted with via the application and reading of voltages, due to its simple implementation.

¹ <https://github.com/benedictjones>

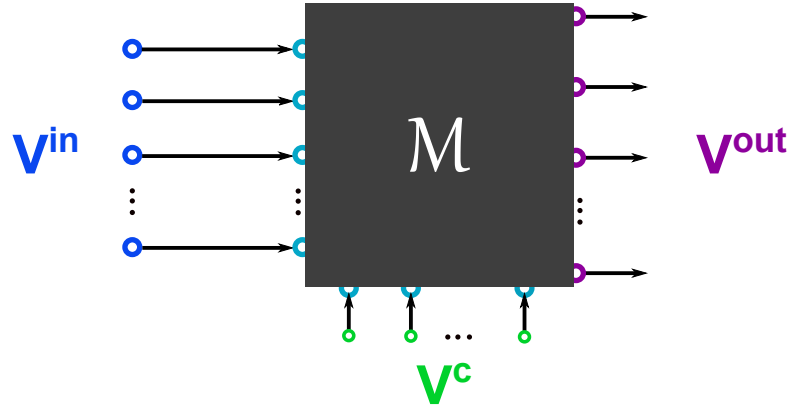


Figure 3.1: Representation of a configurable analogue or ‘material’ processor as a black box which transforms input data voltages V^{in} and configurable stimuli V^c signals into new output representations V^{out} .

Consider a material processor which contains P -inputs and Q -outputs. Input signals are broken into input configuration ‘stimuli’ voltages $V^c = [V_1^c, V_2^c, \dots, V_S^c]$ used to alter how the material processor behaves, and input ‘data’ voltages $V^{in} = [V_1^{in}, V_2^{in}, \dots, V_R^{in}]$ used to apply information to the material processor, where S is the number of stimuli, R is the number of data voltage inputs, and $P = S + R$. Once the input signals have been applied, the output signal can be read from the material processor. The outputs generated can be considered to be the result of a black box transformation, defined by the function \mathcal{M} as follows:

$$V^{out} = \mathcal{M}(V^{in}, V^c) , \quad (3.2.1)$$

where $V^{out} = [V_1^{out}, V_2^{out}, \dots, V_Q^{out}]$ is the vector of output voltages read from the CAP. A visualisation of such a conductive material based processor is presented in Fig. 3.1. To utilise such a system as a fully fledged EiM processor, the challenge becomes how best to interpret the output signals and optimise the system’s configurable parameters for the task at hand.

3.3 EiM for Classification

As previously discussed in §2.2.1, traditional EiM devices have typically used either single output-threshold [6, 1, 2] or output-output comparisons [7, 8], to predict a particular label or class. However, it is hypothesised that using only one or two fixed outputs is likely to lead to inflexible systems and poor performance, where an apparently badly performing material cannot be appropriately reconfigured for maximum exploitation. For example, while one output electrode/node might lead to poor performance, another might produce useful output signals. Indeed, some initial work using multiple outputs [9] and insights from Physical Reservoir Computing (RC) [10, 11] supports this.

For this reason, an alternative EiM processor structure was developed. Considering the basic structure of a material processor presented in §3.2, a generic framework for its computational exploitation is produced.

Consider a classification problem as detailed in §3.6.1, containing K data instances to be classified. Each data instance k is defined by its J attributes $\mathbf{a}(k) = [a_1(k), a_2(k), \dots, a_J(k)]$. These input attributes are converted to input data voltages $V_r^{in}(k)$ which are applied to one of the input electrodes as follows:

$$V_r^{in}(k) = a_r(k) , \quad (3.3.2)$$

where r is a data-driven input electrode corresponding to an input attribute, assuming a 1:1 conversion between the attribute unit and Volts, and the total number of data-driven input electrodes R is equal to the total number of data attributes J .

A basic EiM processor might just be configured using voltage stimuli [12] that will alter the characteristics of the material, as mentioned in §3.2. The system's configurable parameters are gathered together into a vector of decision variables \mathbf{X} . Therefore, a potential 'configured' solution for this basic EiM processor would be defined by a set of static configuration voltages applied to the material [6, 13], and the decision vector, sometimes known as a genome, would be defined as:

$$\mathbf{X} = [V_1^c, V_2^c, \dots, V_S^c]^T , \quad (3.3.3)$$

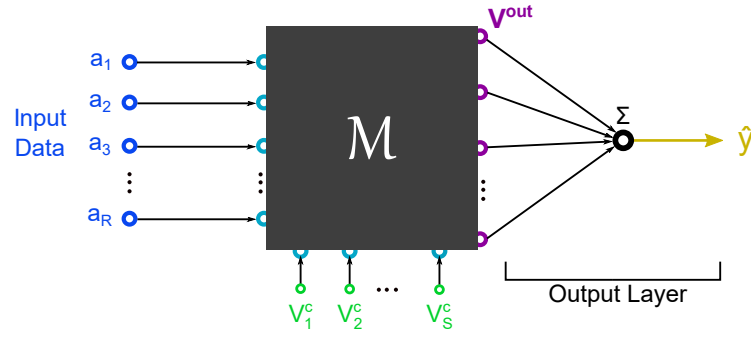


Figure 3.2: Illustration of the proposed EiM processor structure, where output signals are collected using an output layer and combined to create an overall processor response Y used to make a classification prediction \hat{y} .

where T is the transpose, V_s^c is a stimuli voltage applied to a configuration electrode s , and the total number of configuration electrodes is S .

Once the input data and stimuli signals have been applied, the outputs can be read. However, these material outputs require some interpretation scheme in order to classify a processed input data instance. Here, an output layer is introduced, which collects the output voltage signals and produces an overall network response Y , defined as the sum of the voltages $V_q^{out}(k)$ at all the output nodes q :

$$Y(k) = \sum_{q=1}^Q V_q^{out}(k) , \quad (3.3.4)$$

where Q is the total number of output nodes/electrodes.

The collected response Y , is compared with a simple threshold to make a prediction. For binary classification, a data instance is designated a predicted class label using:

$$\hat{y}(k) = \begin{cases} 2, & \text{if } Y(k) \geq 0 \\ 1, & \text{if } Y(k) < 0 \end{cases} , \quad (3.3.5)$$

where \hat{y} is the predicted class or label of the processed data instance. The structure of the proposed EiM processor is shown in Fig 3.2.

Finally, the fitness for a particular configuration of the EiM can be computed using an Objective or Loss Function Φ , as discussed in §2.4. In order to improve the system, an EA is used to optimise the system's decision parameters \mathbf{X} , as discussed

in §2.3.

The lack of consistency of EiM system structure and interpretation scheme in the literature makes it difficult to compare techniques and isolate which changes lead to more (or less) successful EiM processors. The interpretation scheme described above, defined by Eq.(3.3.4) and Eq.(3.3.5), was chosen to be conceptually simple, with the aim of clarifying the role of material properties upon performance. The choice of interpretation scheme can impact the performance of an EiM processor, this is discussed in further detail in §4.2. It is noted, however, that the framework developed and used here could be modified to examine other arbitrary network topologies or interpretation schemes.

3.4 Simulated CAP Model

In order to quickly and efficiently analyse different material and algorithm properties, a model was developed to simulate a conductive network which could act as a proxy for a physical nanomaterial. The material model was based on circuit networks which could be solved by a SPICE simulator. PySpice ² was used to interface a python program (which can run the optimising EA etc.) with Ngspice ³, an established open source mixed level circuit simulator.

Many EiM processors utilise nanomaterials that provide complex morphology with random interconnections between electrodes, such as Single Walled Carbon Nanotubes (SWCNTs). Thus, within the model, conductive networks are generated in which all nodes are interconnected via an equivalent circuit ' r_{mn} ' which have IV characteristics selected from a distribution similar to that of a real nanomaterial. Each node n represents an electrode that is either an:

- Input driven voltage nodes, which can be purposed as data-driven voltages V^{in} or configuration voltage stimuli V^c ,
- Read output voltages V^{out} , computed via Direct Current (DC) operating point analyses.

²Developed by Fabrice Salvaire <https://github.com/PySpice-org/PySpice>

³<https://ngspice.sourceforge.io/>

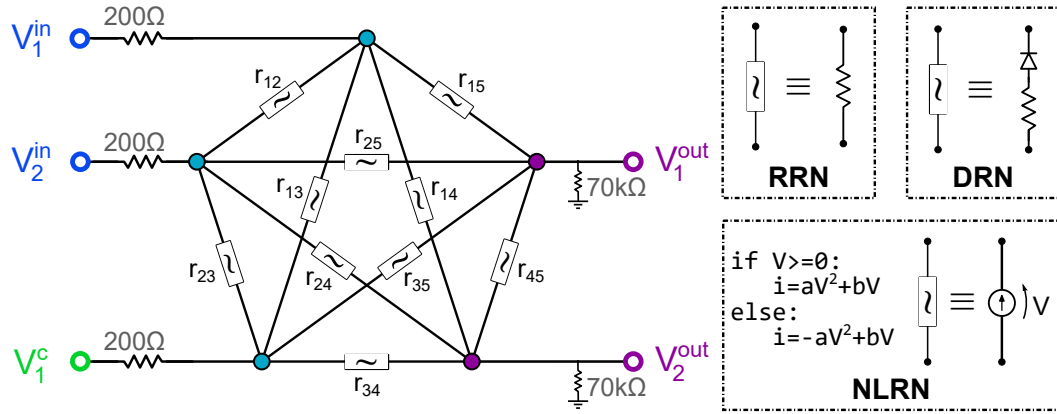


Figure 3.3: Example of a 5 node fully connected network used to model EiM material processors, with three input nodes (V_1^{in} , V_2^{in} and V_1^c) and two output nodes (V_1^{out} and V_2^{out}). Input nodes can be allocated as data-driven voltages V^{in} or configuration voltage stimuli V^c . Between every pair of nodes is a component (i.e., complex sub-circuit) modelling a particular material property.

An individual processor is thus characterised by the equivalent circuits between nodes, and the number of nodes (which represent electrode connections) required for the inputs and outputs. Figure 3.3 shows an example of a fully connected network, containing three input and two output nodes, which could be used as a material processor within an EiM system. The equivalent circuits between nodes are selected to replicate different functional forms of conductivity and randomness therein.

Thus, this model describes the IV characteristics of an ensemble of nanoparticles between electrodes of the EiM processor, rather than individual elements [14] or junctions [15]. The motivation for this approach was that individual nanoparticles are usually at least an order of magnitude smaller than the electrode array upon which they are deposited [7, 6, 16] and so it is generally only possible to experimentally characterise a network of nanoparticles, rather than the nanoparticles themselves. In this work, three material models are considered:

- Resistor Random Network (RRN) in which a randomly selected resistor is between every node pair,
- Non-Linear Random Network (NLRN) in which a current source of non-linear characteristic is between every node pair,
- Diode Random Network (DRN) in which a diode of random orientation is in

series with a randomly selected resistor between every node pair.

Note that since this model's focus is to investigate the foundational issues of randomness and conductivity, the impact of negative differential resistance offered by memristive materials [17], and physically reconfigurable materials such as nanomaterials suspended in solution [8], are outside its scope.

The non-linearity of the conduction within the materials increases from the RRN, to NLRN, to DRN models. These models were chosen to represent a range of conduction mechanisms that could be realised with nanomaterials. RRN networks have been reported in the literature [18, 19]. The NLRNs are modelled after the behaviour of on SWCNT/Poly(butyl methacrylate) (PBMA) composites; assuming the IV characteristic was symmetrical, experiments (Appendix B.1) found they could be fitted well using current source equations defined as:

$$I = \begin{cases} aV^2 + bV, & \text{if } V \geq 0 \\ -aV^2 + bV, & \text{if } V < 0 \end{cases}, \quad (3.4.6)$$

where a and b are material properties, I is the current and V is the voltage between nodes/electrodes. Lab-based tests were carried out to determine that $a \in [33\text{n}, 170\text{n}]$ and $b \in [280\text{n}, 960\text{n}]$. These limits were used to generate uniformly distributed random a and b values for the simulated NLRN materials. Finally, the DRN is a natural extension to the modelled NLRN, representing a highly non-linear nanomaterial with a non-symmetrical IV characteristic. The RRN and DRN use resistor values uniformly randomly selected from between $\in [10, 100]\text{k}\Omega$, the DRN uses 1N4148PH diodes of random direction, and all the simulated systems use voltages $\in [-5, 5]\text{V}$ unless otherwise stated. A shunt resistor of $70\text{k}\Omega$ is connected to each output node, and an input resistance of 200Ω applied to each input node, as shown in Fig 3.3.

When a new material is generated, a specific random seed is used, which can be recalled to re-use the same network. Additionally, these SPICE netlists can be saved and used to re-create the RRNs and DRNs physically. Indeed, physical manifestations of example RRN and DRN were constructed using discrete circuit

components and tested using a custom test bench to verify the model’s behaviour (Appendix B.2). However, the NLRNs are theoretical, representing the expected behaviour of the SWCNT composites described above.

3.5 Physical Experimentation

3.5.1 Test Platform

A test platform was needed to interface with conductive networks (or nanomaterials), enabling lab-based experiments to implement EiM processors. To achieve this, some hardware, sometime known as an ‘evolvable motherboard’ [8], is required to translate signals from the computer into physical inputs and stimuli – which in this work is done via application and interpretation of voltages.

The test platform produced for physical experiments in this thesis consists of two parts. Firstly, a HI which is the physical interface used to set and read voltages. Secondly, the SI which consists of the software used to communicate and control the physical hardware. These two halves of the test platform are explored in further detail below.

Hardware Interface

In preparation for physical experiments on real nanomaterials (or conductive networks) a HI was produced. Previous designs at Durham by Eléonore Vissol-Gaudin [8] used an Mbed microcontroller connected to a PC. However, this design lacked some features such as the ability to apply positive and negative voltages, or change the allocated purpose of a pin (i.e., changing a particular pin to an output or input). Other such similar test platforms include the versatile Mecobo board [20, 21], which allowed more flexibility in assigning pin functionality and producing static or some temporal (e.g., PWM) signals. Mecobo did allow voltage signals $\in [-5, 5]V$; however, these were encoded using a $[0, 255]$ (i.e., 8 bit) range.

For this work, a new system was developed which used a Raspberry Pi to both host the optimising EA and control the HI. The HI was realised using a Printed Circuit Board (PCB) whose main components consisted of higher resolution 12 bit

Digital to Analogue Converters (DACs) to apply voltages, 12 bit Analogue to Digital Converters (ADCs) to read voltages, an analogue Multiplexer (MUX) used to toggle connections, Operational Amplifiers (Op-Amps) to amplify signals, and the Raspberry Pi which could communicate and control components over the Serial Peripheral Interface (SPI) interface.

The design process allowed for rapid prototyping and hot swapping of components when required. Three versions of the HI were developed as the design was iterated upon. The introduced features during this process were as follows:

- **HIv1** Initial design, where voltages $\in [0, 10]\text{V}$ could be applied to any of sixteen pins, which connect to a substrate's micro-electrode array. An analogue MUX was used to toggle four of these pins to perform as outputs (connected to an ADC) or inputs (connected to a DAC).
- **HIv2** An amplification layer was added to scale the DACs outputs to a $\in [-10, 10]\text{V}$ range, allowing both positive and negative voltages to be applied. Similarly, the read voltages are compressed to a range $\in [0, 5]\text{V}$ which can be interpreted by the precision ADC. The ADCs now uses a dedicated chip enable pin to achieve faster read speeds.
- **HIv3** Updated design using more precise Op-Amps and a dedicated 5 V reference chip to reduce system noise.

The final HI design (i.e., version 3) is used for the physical experimentation found elsewhere in this Thesis. A schematic of the final HI can be seen in Fig. 3.5 and images of the PCB can be seen in Fig. 3.4. A condensed Bill Of Materials (BOM) containing the main components for the final design is provided in Table 3.1.

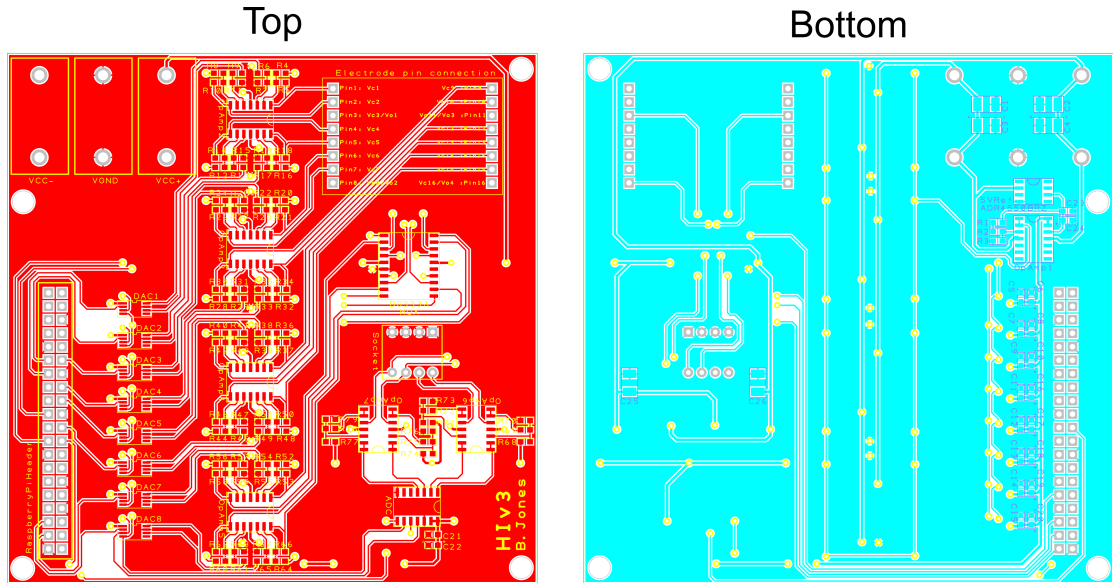


Figure 3.4: Top and bottom side of the final HI PCB.

Table 3.1: Condensed BOM for the final HI, detailing the Integrated Circuit (IC) components used.

Component	Description	Number
MCP3204	Four channel 12 bit ADC	1
MCP4822	Two channel 12 bit DAC	8
DG333A	Precision Quad SPDT Analogue Switch (i.e., Multiplexer)	1
OPA4134	High precision op-amp	6
LF347DT	Op-amp	1
ADR4550BRZ	5 V Reference Chip	1

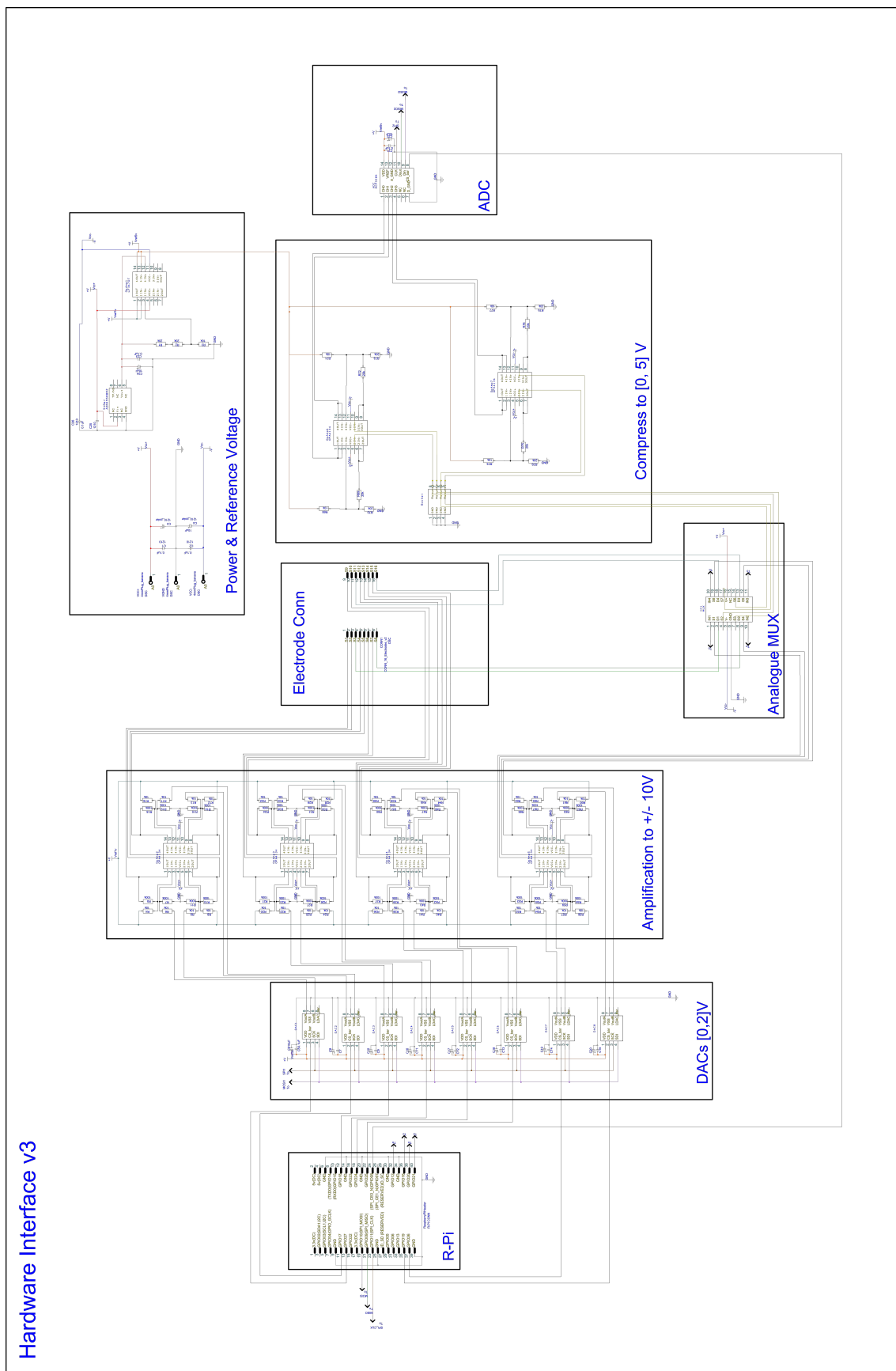


Figure 3.5: Schematic of the Hardware Interface (HI).

Software Interface

The SI encompass the code, written in python, which the Raspberry Pi can run to communicate with the different ICs in the HI. These ICs can be controlled either by the SPI interface or by using the Raspberry Pi's General Purpose Input/Output (GPIO) pins as chip enables. The SI provides a layer of abstraction where higher level commands (e.g., capture and average several voltage readings) can be executed using the lower level code.

3.5.2 Physical CAP

Work in chapter 6 conducts lab-based experiments which implement an *in-materio* based Neural Network (NN) using Lambda Diode Networks (LDNs) as physical neurons. These physical LDNs are exploited as configurable analogue processing units within this novel system.

Lambda Diodes (LDs) consist of a pair of Junction-gate Field-Effect Transistors (JFETs) configured as a two-terminal device [22] and produce an IV curve that contains negative resistance similar to a tunnel diode [23], i.e., they have a non-monotonically increasing IV curve. When several LDs are connected in a network, this negative resistance can produce interesting interactions. Reliable simulation of such non-monotonic IV devices has been of interest [24, 23]. However, when

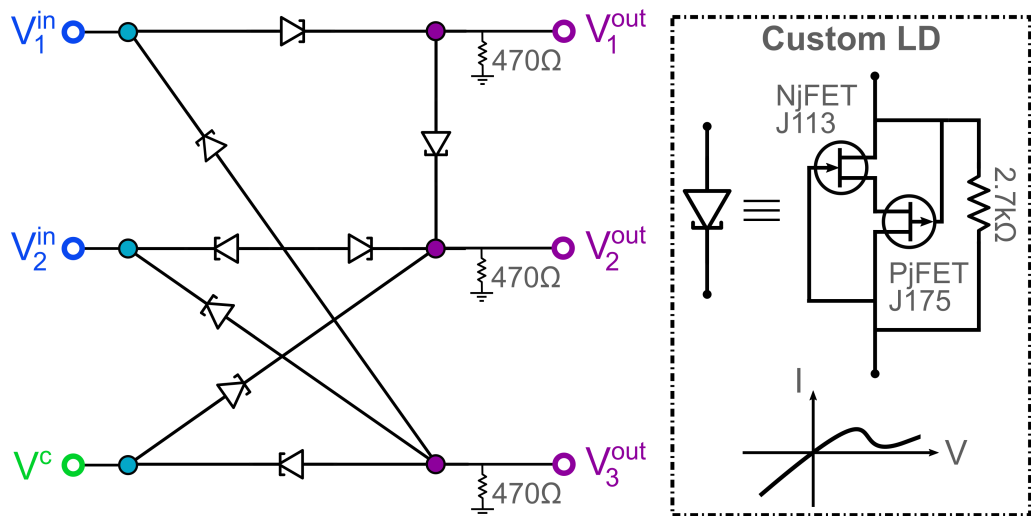


Figure 3.6: The custom Lambda Diode Network (LDN) leveraged as a physical neuron, containing three inputs (V_1^{in} , V_2^{in} , V^c) and three outputs (V_1^{out} , V_2^{out} , V_3^{out}).

wired in large multi-node networks, they can be difficult to simulate with reliable convergence. Conversely, a physical LDN represents a good proxy for a nanomaterial substrate, which is easily constructed and replicates properties found due to (e.g.,) hopping conduction [1], without the need for simulation.

The LDN constructed and leveraged as a physical neuron in this work can be seen in Fig. 3.6. The network contains six nodes: three inputs and three outputs. Of the inputs, two are assigned as ‘data inputs’ and one as a ‘configurable stimuli’. Selecting such a small network allows for better visualisation and interpretation of the physical neuron’s behaviour, as performed in §6.4.1.

3.6 Datasets

3.6.1 Classification Datasets

A classification dataset is a dataset used primarily to solve classification problems, which is the process of assigning a label (i.e., class) to a data instance based on its properties. Here, multi-class datasets are considered where each data instance is assigned to only one label. Consider a dataset D which contains K data instances, each of which belongs to one class $\in [y_1, y_2, \dots, y_L]$; note that binary classification is when data is assigned to one of only two classes. Each data instance k contains J attributes $\mathbf{a}(k) = [a_1, a_2, \dots, a_J]$ which are fed into a model and used to make an informed prediction of the data instance’s label.

In order to both train and evaluate a classification model, a dataset is normally split into two subsets: a training set D^{train} used to optimise the system parameters, and an unseen test set D^{test} used to give an unbiased evaluation of the model’s performance. In this work, the dataset is generally normalised and then scaled to the selected maximum and minimum system voltages. The fitness of the training and test subset can be tracked during evolution, to analyse performance and ensure that an optimising algorithm is successfully converging to an adequate solution.

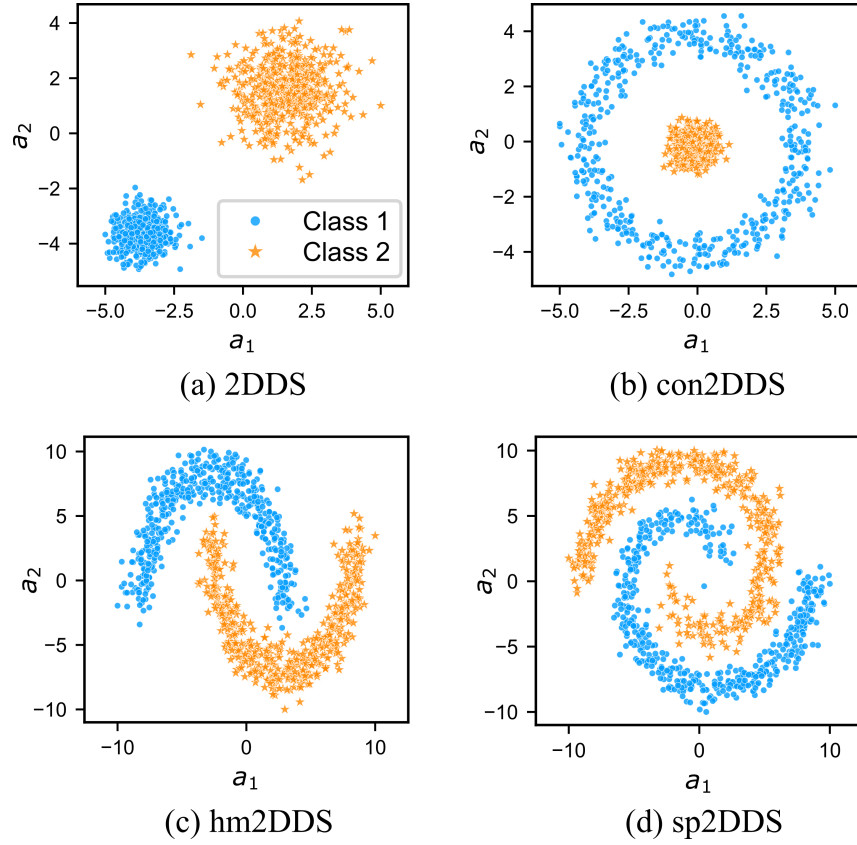


Figure 3.7: The linearly separable (a) 2DDS, the concentric (b) con2DDS, the half moon (c) hm2DDS, and the spiral (d) sp2DDS synthetic two-dimensional datasets.

Synthetic Datasets

To perform some initial investigation of both EiM processors and their exploiting algorithms, some synthetic datasets were generated. These consist of a linearly separable two-dimensional dataset (2DDS), a concentric two-dimensional dataset (con2DDS), a half moon two-dimensional dataset (hm2DDS), and a spiral two-dimensional dataset (sp2DDS). These datasets have been plotted in Fig. 3.7, with further details provided in Table 3.2.

Table 3.2: Synthetic dataset details.

Dataset	Details			
	Nº Attributes	Nº Instances (K)	Nº Classes	Class Ratios
2DDS	2	1000	2	0.50/0.50
con2DDS	2	1000	2	0.50/0.50
hm2DDS	2	1000	2	0.50/0.50
sp2DDS	2	1000	2	0.50/0.50

It is noted that these datasets can also have their classes *flipped* with respect to their original data labels. In this case, the dataset would be denoted by adding a ‘flipped’ prefix; for example: flipped2DDS.

Real Datasets

The remaining classification problems considered in this thesis consist of several common machine learning datasets found on the UCI repository [25]. These include the Banknote Authentication (banknote) dataset [26], the Mammographic Mass dataset (MMDS) [27], the Iris dataset (iris) [28], the Raisin dataset (raisin) [29], the Pima Indians Diabetes Database dataset (diabetes), the Wine (wine) dataset [30], the Australian Credit Approval dataset (aca) [31], and the Wisconsin Diagnostic Breast Cancer dataset (wdbc) [32]. The properties of all these datasets are detailed in Table 3.3.

Table 3.3: Real dataset details.

Dataset	Details			
	Nº Attributes	Nº Instances (K)	Nº Classes	Class Ratios
banknote	4	1372	2	0.55/0.45
MMDS	4	831	2	0.52/0.49
iris	4	150	3	0.33/0.33/0.33
raisin	7	900	2	0.50/0.50
diabetes	8	768	2	0.65/0.35
wine	13	178	3	0.33/0.40/0.27
aca	14	690	2	0.55/0.45
wdbc	30	569	2	0.63/0.37
digits	64	1797	10	$\sim 10\%$ per class

Dataset Complexity

The complexity of a dataset will affect how easily it can be classified [33]. Typical measures include Fisher’s Discrimination Ratio ($f1$) or the Volume of Overlap Region ($f2$). These are both indicators about the overlap of individual feature values. However, there are in fact several classes of complexity measures [34], which can be grouped as follows:

- **Feature Based** measures that characterize how informative the available features are to separate the classes,
- **Linearity** measures that try to quantify whether the classes can be linearly separated,
- **Neighbourhood** measures that characterize the presence and density of same or different classes in local neighbourhoods,
- **Network** measures that extract structural information from the dataset by modelling it as a graph,
- **Dimensionality** measures that evaluate data sparsity based on the number of samples relative to the data dimensionality,
- **Class Imbalance** measures that consider the ratio of the numbers of examples between classes.

A visualisation of the complexity of the discussed classification datasets is shown in Fig. 3.8, generated using the python *proplexity* package [35]. While further details about the specific measures used are given in Appendix A.1 and the literature [33, 34, 35], in general “more colour” relates to a more complex classification dataset.

3.6.2 AutoEncoder Datasets

Finally, the Optical Recognition of Handwritten Digits Data Set (digits) [36] is included, as taken from sklearn [37]. This is a smaller collection of handwritten numerical digits from 0 to 9, each made of 8×8 pixels; further details are provided in Table 3.3. This dataset is used to train an in-Materio AutoEncoder in §6.5.

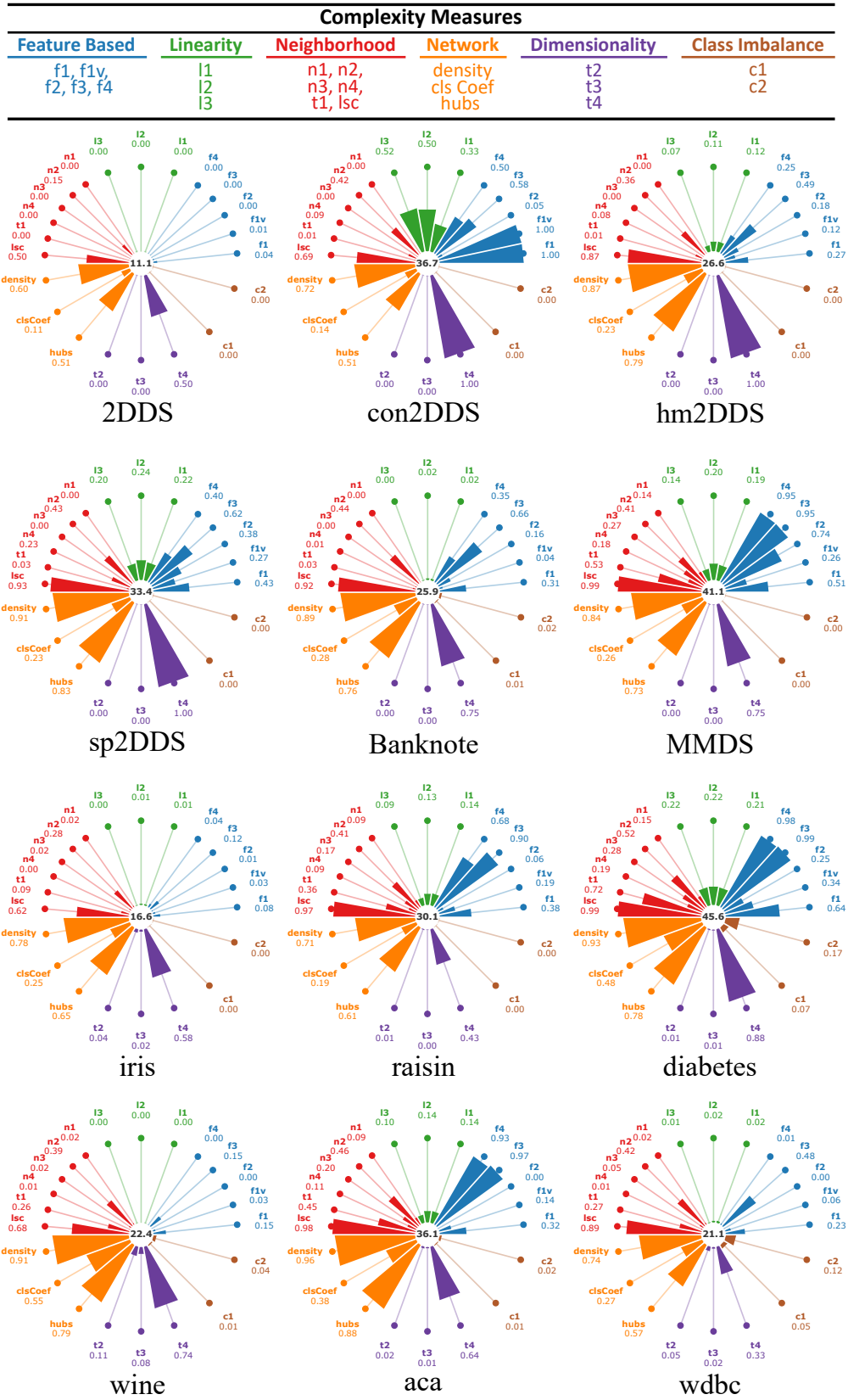


Figure 3.8: Complexity plots as generated by *proplexity* [35] for the classification problem datasets, further metric details given in Appendix A.1. In general, “more colour” represents a higher degree of complexity for the specific complexity measure.

Bibliography

- [1] 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. [Online]. Available: <https://www.nature.com/articles/s41586-019-1901-0>
- [2] 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. [Online]. Available: <http://arxiv.org/abs/2007.12371>
- [3] Y. Viero, D. Guérin, A. Vladyka, F. Alibart, S. Lenfant, M. Calame, and D. Vuillaume, "Light-Stimulatable Molecules/Nanoparticles Networks for Switchable Logical Functions and Reservoir Computing," *Advanced Functional Materials*, vol. 28, no. 39, p. 1801506, 2018. [Online]. Available: <https://onlinelibrary.wiley.com/doi/abs/10.1002/adfm.201801506>
- [4] A. Parsa, D. Wang, C. S. O'Hern, M. D. Shattuck, R. Kramer-Bottiglio, and J. Bongard, "Evolving Programmable Computational Metamaterials," in *Proceedings of the Genetic and Evolutionary Computation Conference*, Jul. 2022, pp. 122–129. [Online]. Available: <http://arxiv.org/abs/2204.08651>
- [5] D. Linden, "A system for evolving antennas in-situ," in *Proceedings Third NASA/DoD Workshop on Evolvable Hardware. EH-2001*, Jul. 2001, pp. 249–255.
- [6] 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. [Online]. Available: <http://www.nature.com/articles/nnano.2015.207>
- [7] 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. [Online]. Available: <http://aip.scitation.org/doi/10.1063/1.4915343>
- [8] 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. [Online]. Available: <http://ieeexplore.ieee.org/document/8123677/>
- [9] M. Dale, S. Stepney, J. F. Miller, and M. Trefzer, "Reservoir computing in materio: A computational framework for in materio computing," in *2017 Inter-*

- national Joint Conference on Neural Networks (IJCNN)*, May 2017, pp. 2178–2185.
- [10] M. Dale, S. Stepney, J. Miller, and M. Trefzer, “Reservoir computing in materio: An evaluation of configuration through evolution,” *2016 IEEE Symposium Series on Computational Intelligence (SSCI)*, 2016.
- [11] 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. [Online]. Available: <http://www.sciencedirect.com/science/article/pii/S0893608019300784>
- [12] B. A. H. Jones, J. L. P. Chouard, B. C. C. Branco, E. G. B. Vissol-Gaudin, C. Pearson, M. C. Petty, N. Al Moubayed, D. A. Zeze, and C. Groves, “Towards Intelligently Designed Evolvable Processors,” *Evolutionary Computation*, pp. 1–23, Mar. 2022. [Online]. Available: https://doi.org/10.1162/evco_a_00309
- [13] 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. [Online]. Available: <http://www.nature.com/articles/srep32197>
- [14] J. W. Lawson and D. H. Wolpert, “Adaptive Programming of Unconventional Nano-Architectures,” *Journal of Computational and Theoretical Nanoscience*, vol. 3, no. 2, pp. 272–279, Apr. 2006.
- [15] K. Greff, R. M. J. van Damme, J. Koutnik, H. J. Broersma, J. O. Mikhail, 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. [Online]. Available: <https://research.utwente.nl/en/publications/using-neural-networks-to-predict-the-functionality-of-reconfigura>
- [16] M. Massey, D. Volpati, F. Qaiser, A. Kotsialos, C. Pearson, D. Zeze, and M. Petty, “Alignment of liquid crystal/carbon nanotube dispersions for application in unconventional computing,” *AIP Conference Proceedings*, vol. 1648, no. 1, p. 280009, Mar. 2015. [Online]. Available: <http://aip.scitation.org/doi/abs/10.1063/1.4912538>
- [17] D. B. Strukov, G. S. Snider, D. R. Stewart, and R. S. Williams, “The missing memristor found,” *Nature*, vol. 453, no. 7191, pp. 80–83, May 2008. [Online]. Available: <http://www.nature.com/articles/nature06932>
- [18] M. Dale, J. F. Miller, S. Stepney, and M. A. Trefzer, “Evolving Carbon Nanotube Reservoir Computers,” in *Unconventional Computation and Natural Computation*, ser. Lecture Notes in Computer Science, M. Amos and ANNE. CONDON, Eds. Cham: Springer International Publishing, 2016, pp. 49–61.

- [19] O. R. Lykkebø, S. Nichele, and G. Tufte, “An Investigation of Square Waves for Evolution in Carbon Nanotubes Material,” *Artificial Life Conference Proceedings*, vol. 27, pp. 503–510, Jul. 2015. [Online]. Available: <https://www.mitpressjournals.org/doi/abs/10.1162/978-0-262-33027-5-ch088>
- [20] FAWADA. QAISER, “Training Single Walled Carbon Nanotube based Materials to perform computation,” Doctoral, Durham University, 2018. [Online]. Available: <http://etheses.dur.ac.uk/12893/>
- [21] 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.
- [22] A. Ipri, “Lambda diodes utilizing an enhancement-depletion CMOS/SOS process,” *IEEE Transactions on Electron Devices*, vol. 24, no. 6, pp. 751–756, Jun. 1977.
- [23] K.-J. Gan, Y.-K. Su, and R.-L. Wang, “Simulation and analysis of negative differential resistance devices and circuits by load-line method and PSpice,” *Solid-State Electronics*, vol. 42, no. 1, pp. 176–180, Jan. 1998. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S0038110197002165>
- [24] N. M. Kriplani, S. Bowyer, J. Huckaby, and M. B. Steer, “Modelling of an Esaki Tunnel Diode in a Circuit Simulator,” p. e830182, Apr. 2011. [Online]. Available: <https://www.hindawi.com/journals/apec/2011/830182/>
- [25] E. K. T. Dheeru Dua, “UCI Machine Learning Repository,” 2017. [Online]. Available: <http://archive.ics.uci.edu/ml>
- [26] Lohweg, “UCI Machine Learning Repository: Banknote authentication Data Set,” Apr. 2013. [Online]. Available: <https://archive.ics.uci.edu/ml/datasets/banknote%20authentication>
- [27] M. Elter, “UCI Machine Learning Repository: Mammographic Mass Data Set,” Oct. 2007. [Online]. Available: <https://archive.ics.uci.edu/ml/datasets/mammographic+mass>
- [28] R. A. Fisher, “UCI Machine Learning Repository: Iris Data Set,” Jul. 1988. [Online]. Available: <https://archive.ics.uci.edu/ml/datasets/iris>
- [29] U. Ilhan, A. Ilhan, K. Uyar, and E. I. Iseri, “Classification of Osmancik and Cammeo Rice Varieties using Deep Neural Networks,” in *2021 5th International Symposium on Multidisciplinary Studies and Innovative Technologies (ISM-SIT)*, Oct. 2021, pp. 587–590.
- [30] M. Forina, “UCI Machine Learning Repository: Wine Data Set,” Jul. 1991. [Online]. Available: <https://archive.ics.uci.edu/ml/datasets/wine>

- [31] K. Quinlan, “UCI Machine Learning Repository: Statlog (Australian Credit Approval) Data Set,” 1987. [Online]. Available: [https://archive.ics.uci.edu/ml/datasets/statlog%20\(australian%20credit%20approval\)](https://archive.ics.uci.edu/ml/datasets/statlog%20(australian%20credit%20approval))
- [32] W. H. Wolberg, “UCI Machine Learning Repository: Breast Cancer Wisconsin (Diagnostic) Data Set,” Nov. 1995. [Online]. Available: [https://archive.ics.uci.edu/ml/datasets/Breast+Cancer+Wisconsin+\(Diagnostic\)](https://archive.ics.uci.edu/ml/datasets/Breast+Cancer+Wisconsin+(Diagnostic))
- [33] T. K. Ho and M. Basu, “Complexity measures of supervised classification problems,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 24, no. 3, pp. 289–300, Mar. 2002.
- [34] A. C. Lorena, L. P. F. Garcia, J. Lehmann, M. C. P. Souto, and T. K. Ho, “How Complex is your classification problem? A survey on measuring classification complexity,” Dec. 2020. [Online]. Available: <http://arxiv.org/abs/1808.03591>
- [35] J. Komorniczak and P. Ksieniewicz, “Problexity—An open-source Python library for supervised learning problem complexity assessment,” *Neurocomputing*, vol. 521, pp. 126–136, Feb. 2023. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S0925231222014461>
- [36] E. Alpaydin and C. Kaynak, “UCI Machine Learning Repository: Optical Recognition of Handwritten Digits Data Set,” Jul. 1998. [Online]. Available: <https://archive.ics.uci.edu/ml/datasets/Optical+Recognition+of+Handwritten+Digits>
- [37] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and É. Duchesnay, “Scikit-learn: Machine Learning in Python,” *Journal of Machine Learning Research*, vol. 12, no. 85, pp. 2825–2830, 2011. [Online]. Available: <http://jmlr.org/papers/v12/pedregosa11a.html>

Chapter 4

Monolithic EiM Devices

4.1 Chapter Overview	74
4.2 Algorithm & Material Interaction	76
4.3 Advanced EiM Processor	82
4.4 Summary	92
Bibliography	93

4.1 Chapter Overview

Monolithic Evolution in-Materio (EiM) devices are systems which attempt to exploit a nanomaterial substrate as a computational resource. While any interfaceable material or medium could be used, conductive nanomaterials are typically utilised since they can be easily interacted with by voltage signals. Therefore, EiM systems are often built from a nanomaterial which is operated as a Configurable Analogue Processor (CAP), often referred to here as a ‘material processor’, whose complex physical properties are exploited and leveraged towards a desired computational task. EiM processors are thus comprised of a material whose characteristics are determined by its configuration, and programming (or re-programming) is achieved by an Evolutionary Algorithm (EA) which optimises the material’s configuration for a target application.

The electronic functionality of EiM processors is not designed by the assembling of discrete components, rather an optimal material configuration is evolved through

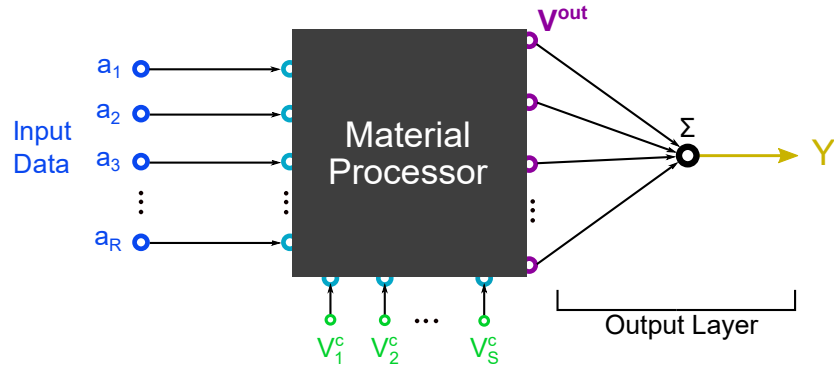


Figure 4.1: EiM processor structure as discussed in §3.3, where output signals are collected using an output layer and combined to create an overall processor response Y used to make a classification prediction \hat{y} .

a supervised learning process. The human element of EiM processor design is the selection of a configurable material, and an appropriate algorithm to efficiently exploit its properties for the target application [1, 2, 3, 4]. The importance of material selection in this process is underscored by the significant differences in computing function observed when different configurable materials are optimised by the same algorithms to solve the same problems [4, 5, 6].

This chapter seeks to address the fundamental issue of what material properties lead to better performing EiM processors, and which configurable parameters can ensure they are best exploited. These largely unanswered questions have historically been complicated by the many and varied styles of EiM processor design, along with slow development and implementation times.

Here, the standardised EiM structure (Fig. 4.1) and simulation model developed in §3.3 and §3.4, allows us to directly relate the choice of nanomaterial used, via the Current-Voltage (IV) characteristic, and algorithm configuration parameters to the classification performance. The framework developed enables reliable and repeatable results, as well as significantly faster investigation into EiM systems than would otherwise be possible with physical construction and experimentation. It was demonstrated that problem complexity influences the most beneficial nanomaterial choice, and furthermore, that design of the algorithm could lead to better exploitation of intrinsic properties. Together, these findings show how EiM processors can be designed to give better performance for classification problems. The results dis-

cussed in this chapter contributed to work presented in the MIT Press Evolutionary Computation (ECJ)¹ Journal [7].

4.2 Algorithm & Material Interaction

4.2.1 Material Properties and Stimuli Voltages

Firstly, the role of material properties on processor performance was examined. In §3.4, three simulated CAP conductive networks were developed for this purpose, including the Resistor Random Network (RRN), Non-Linear Random Network (NLRN) and Diode Random Network (DRN) models. Using these simulated material processor models, conductive networks with two input nodes, two output nodes and two configuration nodes were generated. Fig. 4.2a, 4.2b and 4.2c show surface plots of network response Y , relating to the sum of voltages from output nodes (Eq.(3.3.4)), for a range of data inputs a_1 and a_2 , for the RRN, NLRN and DRN models respectively. These plots show the behaviour of only a single of each type of material processor, not meant to be exhaustive, but selected to show an example of the underlying behavioural trends. Here, the two configuration voltages V_1^c and V_2^c are varied so that their role is highlighted. The network response Y is used as the criterion by which classification is made (Eq.(3.3.5)) and represents the output space which the inputs have been mapped onto. The line $Y = 0$, which is the threshold between the classes, is denoted as the decision boundary. For the unconfigured ($V_{c1} = V_{c2} = 0V$) RRN processor, the decision boundary is a straight line which passes through the origin. Unlike the RRN, the NLRN processor can achieve a smooth curved decision boundary due to its non-linear conduction characteristics. By contrast, for DRN processors, the decision boundary has distinct bends or ‘kinks’ brought about by rapid changes in conductivity when a diode is turned on. Hence, it is observed that the intrinsic properties of the nanomaterial, namely randomness and degree of (non-) linearity in conduction, have significant effects upon the shape of Decision Boundaries in untrained processors. Even nominally the same material

¹<https://direct.mit.edu/evco>

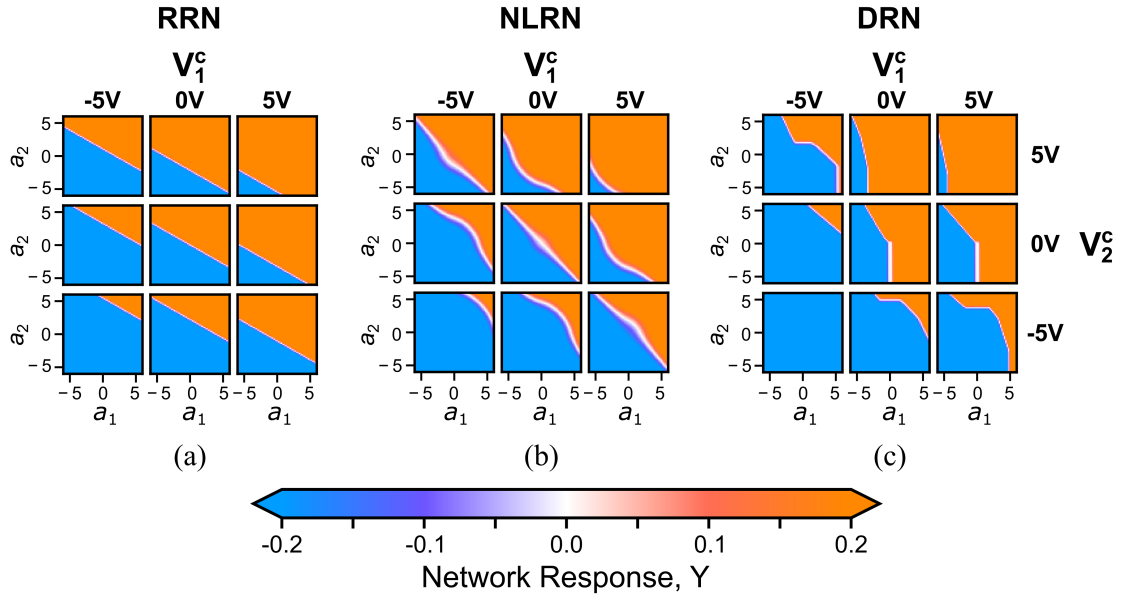


Figure 4.2: Examples of the role that the material and configuration voltages have on untrained EiM processor responses. Surface plot of the network response, Y as a function of input attributes a_1 and a_2 for an untrained (a) RRN, (b) NLRN, and (c) DRN processor, and the effect of varying their two configuration voltages V_1^c and V_2^c .

can have significantly different boundary shapes due to the differences in inter-node (i.e., inter electrode) characteristics.

A material without any evolvable stimuli can be used to map some inputs to a new output space. However, EiM processors have been shown to provide superior classification performance when trained configuration voltages are applied to the network [2]. Thus, Fig. 4.2 also displays the impact of changing the two configuration voltages for the considered networks. EiM processors often operate at low voltages [8, 9, 4] so the effect of applying $-5V$ to $5V$ is considered. These voltage stimuli behave similarly to input biases, however not only ‘shifting’ the output, but also varying how the inter electrode IV characteristics are being exploited, thereby altering how the inputs are mapped to the output space. Within the linear RRN network, application of configuration voltages only leads to a translation of the decision boundary, as a consequence of superposition, and that in no cases is a rotation of the decision boundary observed. For the NLRN and DRN network, a translation of the decision boundary is again observed when configuration voltages are applied, together with some changes in curvature, although not to an extent

where the relative orientation of the two classes is changed. These observations are of significant practical importance, as it shows that the orientation of the decision boundary for both Ohmic and non-Ohmic solid nanomaterials is a function of the random IV characteristics determined during fabrication. An inability to rotate the decision boundary via the configuration voltages to satisfy a particular classification problem may be expected to limit the fitness which an algorithm could obtain.

Basic Classification Performance

To investigate this finding further, the performance of the different networks as EiM processors was assessed for the 2DDS problem, split 80% – 20% to create training and test datasets respectively; as described in §3.6.1. The system was optimised using a DE/best/1/bin algorithm with a clipped boundary constraint, used with a mutation factor of $F = 0.8$, crossover rate of $CR = 0.8$ and population of $\lambda = 20$. These hyperparameter values were selected to match previous similar work [10] in which they were found to perform well; note that the small, simple systems optimised here are unlikely to be largely effected unless highly inappropriate hyperparameter values are selected. Fifteen different individual RRN, NLRN and DRN networks were considered and optimised as a classifier using the Differential Evolution (DE) algorithm and classification error objective function (Φ_{error}) as described in §2.3.1 and §2.4.1 respectively. Here, slightly larger networks were used, each of which had two inputs, two outputs and three configuration nodes. To ensure that randomness within the stochastic algorithm optimisation process did not obscure other trends, the evolution process was repeated on each network 5 times. The test fitness (Φ^{test})

Table 4.1: The mean test fitness $\bar{\Phi}$, standard deviation, and best test fitness Φ^* after 30 iterations, for the different EiM processors.

Dataset	Processor Type	$\bar{\Phi}$	std(Φ)	Φ^*
2DDS	RRN	0.000	0.000	0.000
	NLRN	0.000	0.000	0.000
	DRN	0.005	0.001	0.000
flipped2DDS	RRN	0.514	0.004	0.500
	NLRN	0.500	0.000	0.500
	DRN	0.500	0.000	0.500

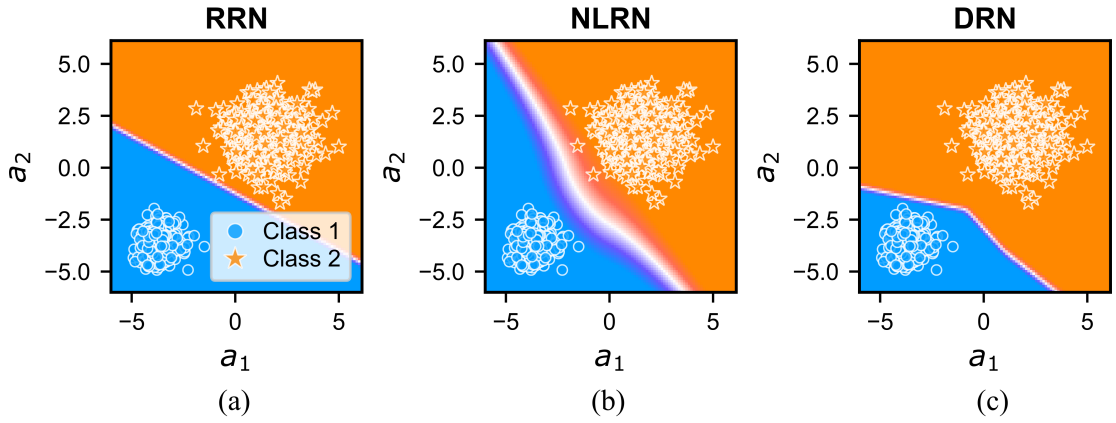


Figure 4.3: Example processor responses following training using the basic algorithm for 30 iterations. Surface plot of the network response, Y as a function of input attributes a_1 and a_2 for a (a) RRN, (b) NLRN, and (c) DRN processor, using the basic EiM algorithm on the 2DDS. The same colour scale as Fig. 4.2 is used.

results for each of the processor types after 30 iterations are shown in Table 4.1.

All the RRN and NLRN processors achieved a zero Φ_{error} fitness (i.e., 100% accuracy). However, some of the DRN processors failed to achieve a zero fitness. The more linear networks have simpler Decision Boundaries which are favourable for the simple 2DDS. Examples of the network response for a trained RRN, NLRN and DRN achieving 100% accuracy are shown in Fig. 4.3.

Using the same material processors and repeating the experiment on the flipped2DDS (Table 4.1) shows all the processor types becoming ‘stuck’, in this case being unable to improve their fitnesses below 0.5. This is because the processors now have unfavourable initial slopes of the decision boundary with respect to the dataset. As the basic algorithm is unable to rotate the decision boundary or make large changes in its curvature, the best that the EA can achieve is to misclassify 50% of the data. These results provide an explanation as to why some experimental data [11] show that nanomaterial composites fail to achieve acceptable classification accuracies following training.

4.2.2 Electrode Reconfiguration and Weighting

While on the one hand, the results in §4.2.1 suggest it should be possible to screen candidate processors prior to possibly lengthy and unsuccessful training, it is argued

that they also suggest a method to exploit nanomaterial composites more effectively. The success or not of the DE algorithm to exploit the nanomaterial is limited by the inter-node IV characteristics, which for solid ‘static’ nanomaterials, is determined at creation. However, it is possible to re-arrange the input and configuration nodes and thus realise different arrangements of inter-node IV characteristics without changing the material. Input electrodes can be used for data driven or configuration voltages, the function of an input electrode and the ability to ‘shuffle’ or relocate different inputs can be easily controlled while programming. On the other hand, a multiplexer could be used to physically reconnect an electrode as either an input (e.g., to a Digital to Analogue Converter (DAC)) or output (e.g., to a Analogue to Digital Converter (ADC)).

Indeed, some EiM approaches have this flexibility by allowing the EA to modify node location [12, 11]. It is reasoned that changes to the inter-node IV characteristics are likely to result in uncovering exploitable configurations of a nanomaterial composite’s conductive network.

This supposition is explored in Fig. 4.4 which shows four examples of the same network’s output response Y , with no configuration voltages applied, but with randomly reordered input (data & configuration) node arrangements for the same networks. Here, only four random input arrangements are shown as examples for each material type, this is not meant to be exhaustive. Indeed, the number of possible input permutations scales quickly with the number of inputs (e.g., for materials with four input nodes there are twenty four possible input arrangements etc.), making it difficult to investigate. In this work, to better understand its uses, the limitations of the technique are considered. It is observed that different input arrangements have a different shape of decision boundary that could be exploited to fit classification data. Furthermore, some slight variations in the slope of the decision boundary are observed, albeit in a discontinuous fashion. However, due to the interpretation scheme defined by Eq.(3.3.4) and Eq.(3.3.5), it is always the case that (e.g.) two positive inputs can only provide a positive (and therefore class 2) output. This in turn suggests that introducing evolving multiplication factors or ‘weights’ for the applied input voltages or read output voltages may be beneficial, as this may al-

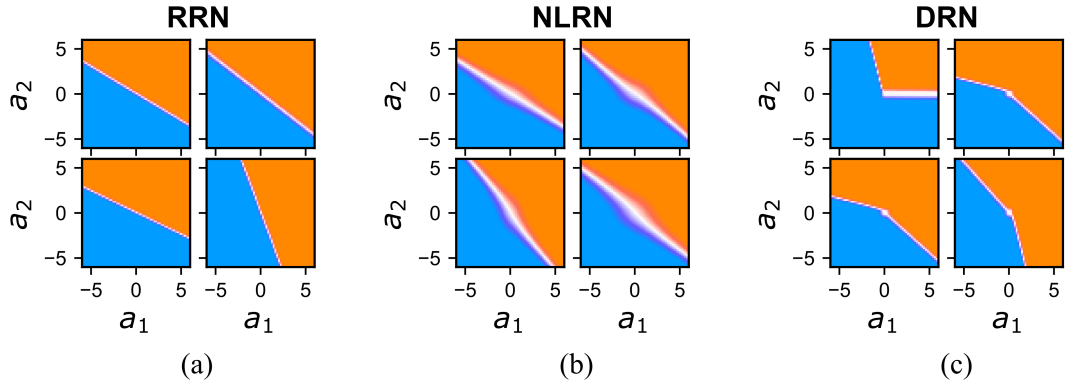


Figure 4.4: Examples of the impact that electrode reconfiguration can have on untrained EiM processor responses. Surface plot of the network response Y as a function of input attributes a_1 and a_2 for four randomly selected shuffle genes (i.e., permutations of the input voltage order) of an unconfigured (a) RRN, (b) NLRN, and (c) DRN processor. The same colour scale as Fig. 4.2 is used.

low unconstrained rotation of the decision boundary. Other types of interpretation scheme, previously mentioned in §2.2.1, might prove similarly limited without their own additional evolvable parameters.

The impact of varying output weights on the material response Y for an unconfigured RRN, NLRN and DRN processor is investigated in Fig. 4.5. Inverting the polarity of one output weight allowed the introduction of more complex behaviour into the material's response, such as the gradient for the RRN processor, and complex boundary shapes of the NLRN and DRN processors. As expected due to symmetry, inverting the polarity of both output weights caused the classes to swap. Using output weights of opposing polarities generally made the decision boundary less 'sharp' since subtracting values leads to Y values closer to zero, which in turn would make a physical EiM processor more susceptible to noise. Fig. 4.5 also shows the effect of input weights to the same RRN, NLRN and DRN networks. It is notable that changing input and output weights yield different responses, since input weights cause a change in input voltages which must propagate through the material network. This is particularly important for the DRN, since due to its asymmetric non-linear IV characteristics, changing the input weights impacts when diodes are turned on, so inverting the polarity of both input weights will not simply cause the classes to swap. Thus, a distinction is drawn between the effect of using input and output weights: input weights allow for variability on how the nanomaterials

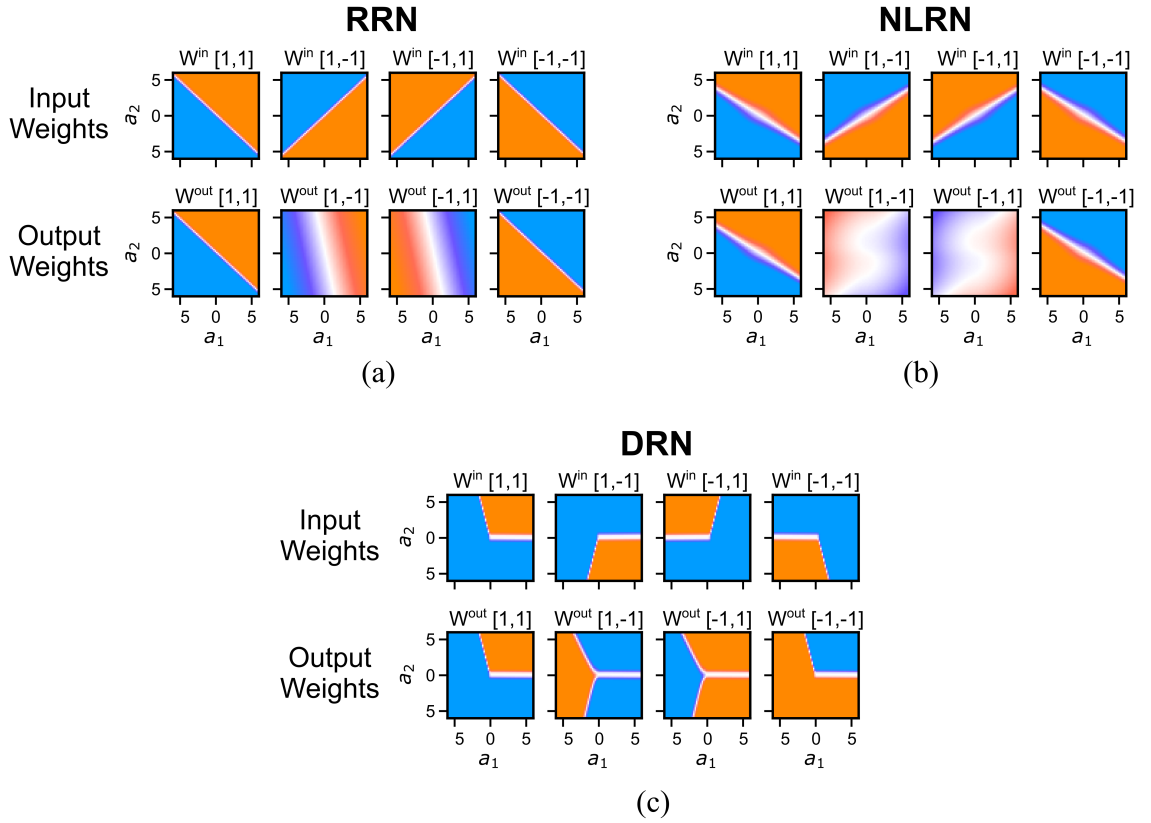


Figure 4.5: Examples of the effect of varying the input and output weightings on untrained EiM processor responses. Surface plot of the network response Y as a function of inputs a_1 and a_2 for an unconfigured (a) RRN, (b) NLRN, and (c) DRN processor with various output or input weightings applied. The same colour scale as Fig. 4.2 is used.

currently selected inter-node IV characteristic is exploited, whereas output weights allow for variation in how the output signals are interpreted and combined.

4.3 Advanced EiM Processor

In order to exploit the distinct impacts of re-arranged electrodes, and changing input or output weights upon the decision boundary, the vector of decision variables was modified to include these additional parameters as follows:

$$\mathbf{X} = [V_1^c, V_2^c, \dots, V_S^c, G_{sh}, w_1^{in}, w_2^{in}, \dots, w_R^{in}, w_1^{out}, w_2^{out}, \dots, w_Q^{out}]^T. \quad (4.3.1)$$

Here, G_{sh} is termed the ‘shuffle’ gene, which allows for reassignment of input electrodes to access different inter-node arrangements which lead to varying output

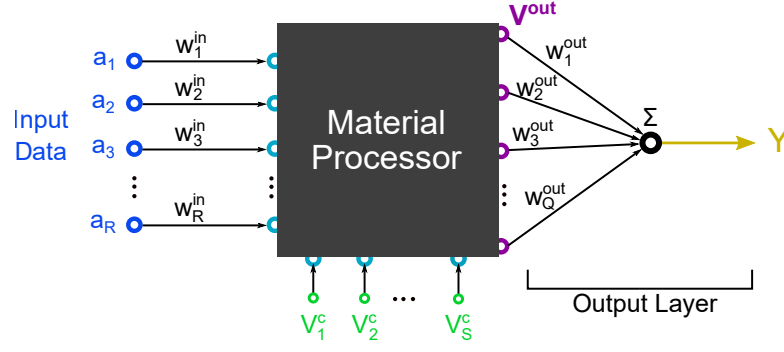


Figure 4.6: Structure of the advanced EiM processor structure. Input data and configurable stimuli (V^c) are applied to the material as voltages. The output voltages are summed to generate an overall response (Y) which is used to predict the binary class (\hat{y}). If enabled input weights (w_r^{in}) and output weights (w_q^{out}) are applied. A shuffle ‘gene’ can re-arrange the applied location of the inputs (both input data and configuration nodes).

responses as shown in Fig. 4.4. G_{sh} is an integer which defines a particular permutation of the ordering for where input and configuration voltages are applied. The input weights $w_r^{in} \in [-1, 1]$ scale the input voltages V_r^{in} applied at the data driven input electrodes r due to an input attribute a_j , such that:

$$V_r^{in}(k) = w_r^{in} \times a_r(k) , \quad (4.3.2)$$

where the total number of data driven input electrodes R is equal to the total number of data attributes J , and k is a particular data instance in a dataset length K . Finally, the output weights $w_q^{out} \in [-2, 2]$ for each output electrode q , allow for changes in the network response Y as follows:

$$Y(k) = \sum_{q=1}^Q w_q^{out} V_q^{out}(k) . \quad (4.3.3)$$

The inclusion of these additional decision parameters allows an exploiting algorithm to optimise the system beyond the simple voltage stimuli traditionally used. Additionally, this well-defined framework allows a reliable investigate into the roles and benefits which these new configuration parameters may have on the different material processors.

4.3.1 Electrode Allocation and Material Properties

Understanding how many configuration and output electrodes are required for good classifier performance will enable more reliable EiM processor construction. To investigate this, fifteen RRN, NLRN and DRN processors were generated, each with a fixed size of ten nodes. Two nodes were designated as data inputs; the remaining eight nodes were assigned as either configuration nodes, output nodes, or if unused were left floating. The effect of varying the number of allocated configuration and output nodes was then investigated, with the results presented in Fig. 4.7. These results show the classification performance on the con2DDS using the advanced EiM algorithm which ran for 50 iterations, and was repeated five times on each network to mitigate randomness in convergence. Results displayed in Appendix C.1 show that after four repetitions of the DE algorithm, the standard deviation of a materials' performance settles to a low value. Similarly, the effect of averaging over a number of the same type but different randomly generated 'materials' was also considered, the performance settled if more than five to ten materials were considered. Therefore, in these experiments, fifteen of each processor type were chosen to ensure that the average capability of a particular material type could be analysed without any one high or low performance processor skewing the results, while the simulation times could remain within reasonable lengths.

It was observed that the difference between nanomaterial conductivities (i.e., whether they are (a) RRN, (b) NLRN or (c) DRN) is more important in determining performance than electrode allocation. The number of configuration nodes seems to play a less important role when evolving the RRN, NLRN and DRN materials, and suggests that only a few voltage stimuli are required. However, it is speculated that materials with more complex properties, or large sparsely connected networks, will benefit more from the voltage configuration stimuli.

The DRN and NLRN EiM processors perform better than the RRN processors, this is because the materials' non-linear IV characteristics are being exploited to better curve the decision boundary and fit to the concentric data. The RRN processors can only produce linear Decision Boundaries which can only be placed tangentially to the data, similar to that shown in Fig. 4.8a, thus severely limiting the final fit-

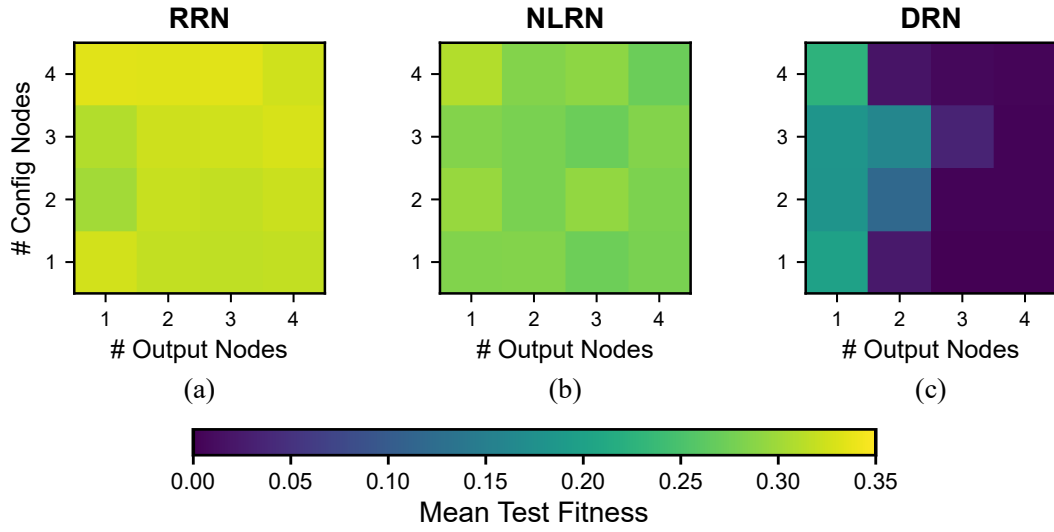


Figure 4.7: **Effect of varying which nodes are allocated as either configuration or output electrodes on the final test fitness after training.** Surface plot of the mean test fitness (from 15 material processors, each with 5 DE repetitions) after 50 iterations using the advanced EiM algorithm on the (a) RRN, (b) NLRN, and (c) DRN networks to classify the con2DDS. The materials all had a fixed size of ten nodes, but the number of nodes allocated as configuration or outputs was varied. Unallocated nodes were left floating.

ness. The NLRN processors, however, only achieve marginally better results than the RRN processors. The material's non-linearities help fit the data, as shown in Fig. 4.8b, but neither additional configuration nodes nor additional output nodes lead to significant improvement. It is speculated that the voltage outputs from the NLRN are too similar, meaning it is harder to combine the outputs to produce an enclosed decision boundary. The DRN networks contain more abrupt non-linearities within its IV characteristics, allowing its output voltages to be more easily combined to generate enclosed Decision Boundaries and achieve better fitnesses, similar to the response shown in Fig. 4.8c. However, the results from Fig. 4.7 showed that it is necessary to combine more than one output to achieve an enclosed area, and for consistently well performing con2DDS classification more than two output nodes are required.

It should be noted that the materials discussed here all have monotonically increasing IV characteristics. Therefore, a material processor with only a single output node/electrode cannot successfully classify the con2DDS (or an XOR problem); instead, at least 2 outputs are required. However, materials containing a Negative

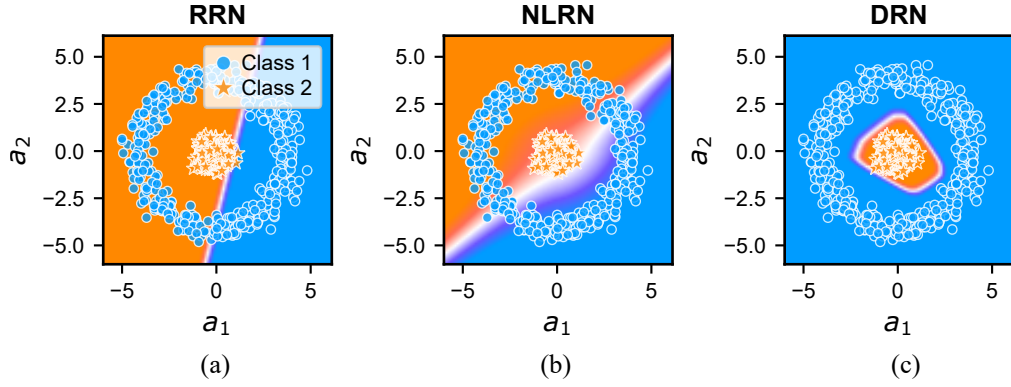


Figure 4.8: Example processor responses following training using the advanced EiM algorithm for 50 iterations. Surface plot of the network response, Y as a function of input attributes a_1 and a_2 for a (a) RRN, (b) NLRN and (c) DRN processor, using the advanced EiM algorithm with all the additional configuration parameters, on the con2DDS. The same colour scale as Fig. 4.2 is used.

Differential Region (NDR) can be exploited to solve the XOR problem with only one output [12, 9].

4.3.2 Effect of Modifying the Decision Vector

Work in this section investigates the impacts of including the ‘shuffle’ gene, input weights and output weights as configuration parameters into the decision vector, both individually and in combination, for the three networks, when solving the classification problems. Considering the results discussed in §4.3.1, and using the same fifteen RRN, NLRN and DRN material processors, three nodes were allocated as outputs, two nodes as inputs, and the remaining five as configuration nodes.

The con2DDS was used to train the systems for 50 iterations, but with the differing combinations of decision parameters, using the same hyperparameter values as those considered in §4.2.1. The results of the best member (θ) are recorded, and the convergence of the mean training fitness (from the results of the 15 different randomly generated material networks, each with 5 DE algorithm repetitions) is shown in Fig 4.9a, 4.9c and 4.9e for the RRN, NLRN and DRN processor types respectively. Box plots of the corresponding processors’ final test fitnesses are also shown in Fig 4.9b, 4.9d and 4.9f. For clarity, results are only displayed for the different individual additional configuration parameters (i.e., ‘shuffle’, input weights

or output weights) in conjunction with configuration voltages. Additionally, the performance when all configuration parameters are enabled is displayed. Results for all remaining combinations of configuration parameters are shown in Appendix C.2.

Using only configuration stimuli voltages (+) leads to poor fitnesses in all the processor types considered. The inability of this evolutionary scheme to explore different IV characteristics or rotate the decision boundary leads to similar behaviour observed in §4.2.1 when using the flipped2DDS led to ‘stuck at’ faults. By contrast, using shuffle (人) or input weights (★) in the decision vector allows for a wider exploitation of the material characteristics, improving performance. The ability of shuffle to discover useful IV characteristics and the use of input weights to exploit them is therefore fundamentally limited by the EiM processor’s material properties. This is easily distinguished by the fact that only schemes which used output weights (Y & ▲) could achieve test fitnesses below 0.160 (full table of results given in Appendix C.2). Therefore, for processors to achieve better fitnesses, it becomes essential that the interpretation scheme can evolve to successfully combine the material’s output voltage states. This enables the introduction of more complex boundary features, such as a fully enclosed area, which is needed for the concentric dataset.

While the scheme using all the additional configuration parameters (▲) achieved good eventual performance after the 50 iterations, it converged significantly slower than the output weight only scheme (Y) used for the NLRN and DRN processors. This suggests that not all the configuration parameters are needed, and their introduction can be either counter productive and/or inflates the search space. However, it is proposed that a marginal decrease in convergence speed is balanced against the significant gains in flexibility that a solution could take.

This experiment was repeated for the 2DDS and flipped2DDS as seen in Appendix C.3 & C.4 respectively. Most configuration schemes solved the 2DDS and achieved a zero fitness within fewer than 10 iterations. However, all the processor types fail to optimise the flipped2DDS unless either input or output weights are used. This again highlights the limitations of the configuration voltages and shuffle gene, which may only exploit the complexities of the given EiM processors’ material properties. As discussed in §4.2.2, to enable the decision boundary to ‘flip’ or rotate,

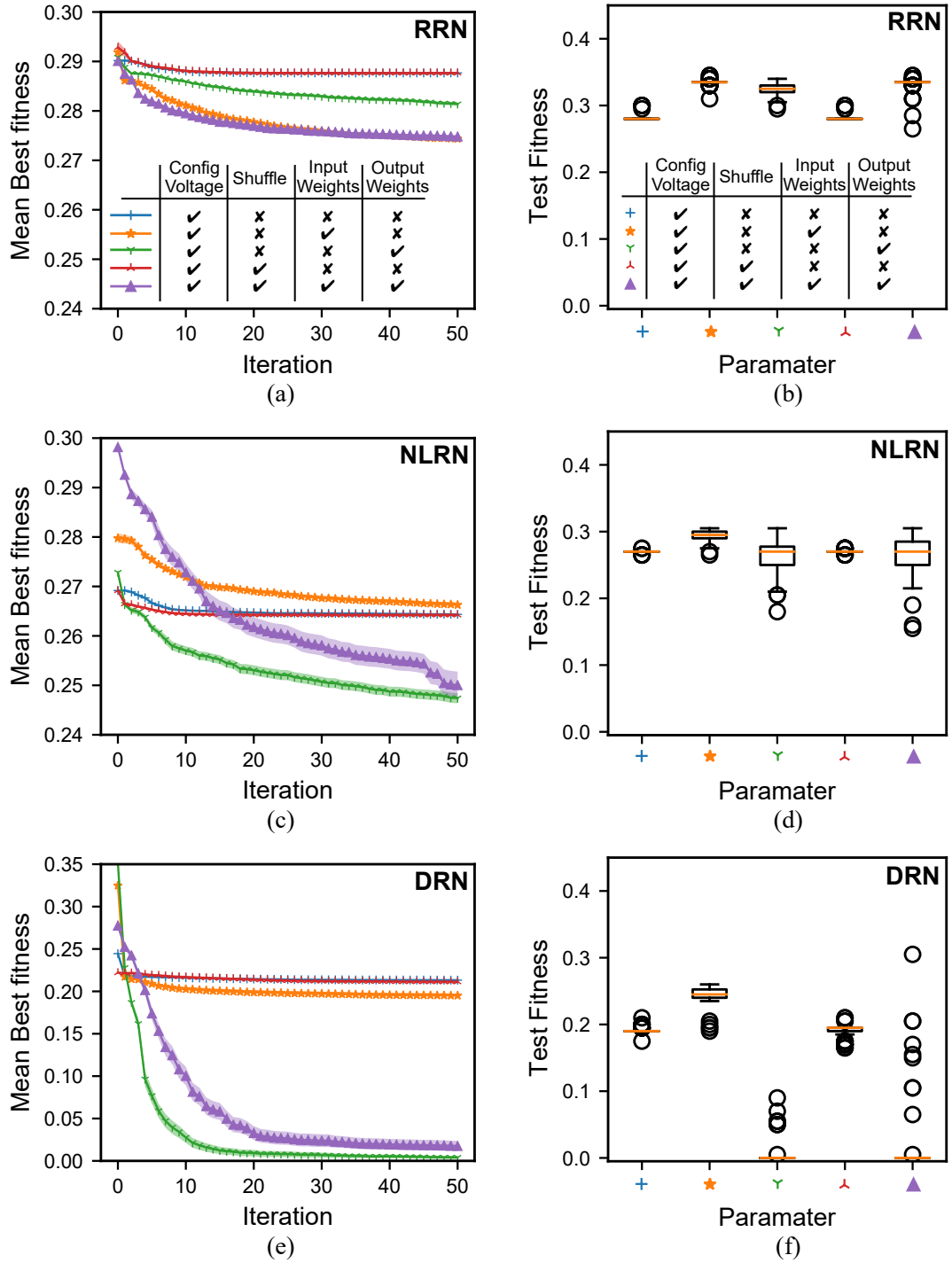


Figure 4.9: **Evolution of training fitness and final test fitness for EiM processors using different materials and configuration parameters, classifying the con2DDS.** The mean best training fitness & standard error (from 15 material processors, each with 5 DE repetitions) over 50 iterations, with different DE algorithm configuration parameters enabled, for (a) RRN, (c) NLRN, and (e) DRN processors using the con2DDS. These are paired with box & whisker plots of the final test fitness results for the (b) RRN, (d) NLRN, and (f) DRN processors.

input or output weightings must be used.

These results support Miller and Downing’s conjecture that materials with rich, complex physics should be good candidate EiM processors [13]. These results have shown that the form of non-linear conduction within the nanomaterial processor will play a role in eventual performance. The relative performance of one EiM processor with respect to another depends sensitively upon the EA and configuration parameters used, and furthermore, that the inherent capability of a nanomaterial for EiM may be ‘hidden’ or obstructed if an inappropriate algorithm is used. Design of algorithm and selection of nanomaterial are thus necessary to achieve good performance for a target application.

4.3.3 Classification Performance

For the sake of comparison, the performance of the basic (configuration voltage only) and advanced (all additional configuration parameters) EiM systems was compared to some common classification techniques. Specifically, this included Logistic Regression and Random Forest *sklearn* [14] models, using the default hyperparameters. The results are presented in Table 4.2, and include classification results on the con2DDS and MMDS datasets, each of which was again split 80% – 20% to create training and test datasets respectively. The con2DDS results are taken from the work discussed in §4.3.2. The MMDS results were generated using the same 15 ten node materials (with 5 repetitions) as in §4.3.1, using the same hyperparameter values as those considered in §4.2.1, but for 100 iterations.

The advanced EiM processors, which exploited all the additional evolvable parameters, can lead to a mean fitness reduction in excess of 49% compared to the ‘basic’ EiM processors for the MMDS dataset. However, if the nanomaterial processors’ properties are unsuitable to the computational task, such as the NLRN devices classifying the con2DDS, a better exploiting algorithm cannot achieve significant performance gains. Notably, when properly harnessed, these simple physically realisable networks can outperform both Logistic Regression and the Random Forest (100 trees) algorithm. However, the performance of these models would likely improve with some hyperparameter tuning.

Table 4.2: Classification performance of the discussed basic and advanced EiM algorithms final classification error test fitness, compared to other common algorithms and other work.

Dataset	Processor Type	$\bar{\Phi}$	std(Φ)	Φ^*
con2DDS	Basic EiM (NLRN)	0.270	0.002	0.265
	Basic EiM (DRN)	0.191	0.004	0.175
	Logistic Regression	-	-	0.490
	Random Forest	-	-	0.000
	Advanced EiM (NLRN)	0.266	0.029	0.155
	Advanced EiM (DRN)	0.000	0.058	0.000
MMDS	Basic EiM (NLRN)	0.485	0.000	0.485
	Basic EiM (DRN)	0.485	0.000	0.485
	Logistic Regression	-	-	0.228
	Random Forest	-	-	0.269
	Advanced EiM (NLRN)	0.236	0.020	0.204
	Advanced EiM (DRN)	0.245	0.037	0.210
	EiM (SWCNT/LC) [10]	0.2051	-	0.1885

Vissol-Gaudin et Al. (2017) [10] used a Single Walled Carbon Nanotube (SWCNT) / Liquid Crystal (LC) EiM processor to classify the MMDS. These carbon nanotubes suspended in a liquid crystal mixture are dynamic, using evolved voltage stimuli to manipulate the nanomaterial to move and form new connections with altered IV characteristics. Classification decisions were made using two output electrodes and an evolvable classification threshold. The SWCNT/LC EiM processor achieves a lower error than the simulated materials discussed in this paper. While some variation is likely due to a different selection of training & test data, it is hypothesised that the increase in performance is due to the conductive network's ability to internally re-configure inter-node connections, altering its IV characteristics. The 'material' processors considered in this chapter all have fixed IV characteristics. While this might present a less flexible *in-materio* processor, it allows for a single processor to be trained on many tasks, and switch between tasks by simply re-calling the trained configuration parameters.

4.3.4 Discussion

These findings can be interpreted as confirmation of the suitability of nanomaterials for EiM processors due to the variation in conduction mechanisms and conductivity [15]. Depending upon the material and device geometry chosen, the conduction pathways in a nanomaterial may have conduction that is Ohmic ($I \propto V$) or Poole-Frenkel ($I \propto V \exp(-(E_d - \beta\sqrt{V})/kT)$) in nature, or be limited by space-charge ($I \propto V^2$) or a Schottky/pn junction ($I \propto \exp(V)$), to name a few examples. Nanomaterials may also display a range of conduction mechanisms within the same composite, as well as variation in their conductivity. However, the inter-electrode IV characteristic of many nanomaterial-based EiM processors reported to date are due to a percolation network of individual nanoparticles [13, 1, 2], and in turn, the apparent diversity in conduction mechanisms that a type of nanoparticle may offer will be reduced due to averaging along the conduction path.

It is suggested that higher performance EiM processors may be realised as the inter-electrode distance approaches that of individual nanoparticles. One of the possible benefits of nano-structured devices, is the possibility of NDR which can be used to classify the XOR problem while only using a single material output [12, 9]. The material networks considered in this work were each assumed to have similar inter-node characteristics like those found in the literature. However, while maybe more challenging to produce, materials with heterogeneous properties might provide a more exploitable and better performing material network, and warrants further research.

Further to the modes of conduction observed, the nanomaterials used in EiM processors fall into one of two categories: *static* materials with fixed IV characteristics and *dynamic* materials which have variable IV characteristics. In this work, only models of static conduction networks and materials were considered; examples of such static nanomaterial processors include SWCNTs suspended in polymer matrices [4, 5, 11, 16, 17] and random resistor networks [4, 6]. However, some nanomaterials can also be dynamically changed by applying a voltage, as in the case of LCs [18, 19], SWCNTs suspended in a LC matrix [2, 20, 21], and memristors [22]. When processing non-temporal data, materials with no memory and a fast settling

time are desirable to allow a large bandwidth [9]. Although comparisons between static and dynamic EiM processors are sparse [23], it appears that *dynamic* materials have better performance in some circumstances.

It is proposed that the reconfiguration of the electrical network in *dynamic* materials allows different inter-node IV characteristics to be continuously explored by the EA, in much the same way as the ‘shuffle gene’ parameter operated here, and that this may be at least part of the reason why some *dynamic* materials have better EiM performance. However, many *dynamic* materials cannot be reset to a previous configuration (e.g. as is the case for SWCNTs suspended in LCs [20]), whereas the trained configuration parameters of a *static* material can be simply recalled. Whether the benefits of configuration recall in *static* materials or a dynamic, irreversible search space are more beneficial to EiM performance appears to be worthy of further study.

4.4 Summary

While EiM is a promising unconventional computing paradigm, analysis of EiM systems remains limited due to slow fabrication and training processes. In this chapter, EiM classifiers are formulated by combining Differential Evolution with an electrical model which is used as a proxy for real EiM material processors. This allowed for fast and efficient *in-simulo* experimentation of EiM processors. Using this framework, foundational issues with EiM processors were investigated. Different materials and evolvable configurable parameters were investigated in succession, allowing their effect to be isolated and analysed.

The findings presented explain why some nanomaterial based EiM processors fail to achieve good performance, and how the exploiting algorithm can be modified to mitigate these effects. Significantly, it is observed that the complexity of the ‘ideally selected’ material scales with the complexity of the problem, with acceptable solutions to simple classification problems being found more quickly with simple random resistor networks, and the solution of more complex problems being favoured by more complex random non-linear networks. Furthermore, differing modes of

reconfiguring the material, weighting the input data, and interpreting the material outputs are shown to have distinct advantages and limitations. Significantly, this chapter has demonstrated how these methods can be used in concert to better exploit the material processor and leverage better performance for EiM classifiers.

Looking forward, these results show that to create high performance EiM processors, rational design of an algorithm must be paired with an appropriate selection of nanomaterial for the target application. The framework developed allowed us to use a standardised EiM processor system, which was used to quantify these trade-offs and make informed decisions in the design of future EiM processors.

Bibliography

- [1] 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.
- [2] ———, “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 ANNE. CONDON, Eds. Cham: Springer International Publishing, 2016, pp. 130–141.
- [3] J. W. Lawson and D. H. Wolpert, “Adaptive Programming of Unconventional Nano-Architectures,” *Journal of Computational and Theoretical Nanoscience*, vol. 3, no. 2, pp. 272–279, Apr. 2006.
- [4] M. Dale, J. F. Miller, S. Stepney, and M. A. Trefzer, “Evolving Carbon Nanotube Reservoir Computers,” in *Unconventional Computation and Natural Computation*, ser. Lecture Notes in Computer Science, M. Amos and ANNE. CONDON, Eds. Cham: Springer International Publishing, 2016, pp. 49–61.
- [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. [Online]. Available: <http://aip.scitation.org/doi/10.1063/1.4915343>
- [6] M. Dale, S. Stepney, J. Miller, and M. Trefzer, “Reservoir computing in materio: An evaluation of configuration through evolution,” *2016 IEEE Symposium Series on Computational Intelligence (SSCI)*, 2016.

- [7] B. A. H. Jones, J. L. P. Chouard, B. C. C. Branco, E. G. B. Vissol-Gaudin, C. Pearson, M. C. Petty, N. Al Moubayed, D. A. Zeze, and C. Groves, "Towards Intelligently Designed Evolvable Processors," *Evolutionary Computation*, pp. 1–23, Mar. 2022. [Online]. Available: https://doi.org/10.1162/evco_a_00309
- [8] 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.
- [9] 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. [Online]. Available: <https://www.nature.com/articles/s41586-019-1901-0>
- [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. [Online]. Available: <http://ieeexplore.ieee.org/document/8123677/>
- [11] 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.
- [12] 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. [Online]. Available: <http://www.nature.com/articles/nnano.2015.207>
- [13] 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. [Online]. Available: <http://ieeexplore.ieee.org/document/1029882/>
- [14] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and É. Duchesnay, "Scikit-learn: Machine Learning in Python," *Journal of Machine Learning Research*, vol. 12, no. 85, pp. 2825–2830, 2011. [Online]. Available: <http://jmlr.org/papers/v12/pedregosa11a.html>
- [15] 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. [Online]. Available: <https://doi.org/10.1109/MC.2017.8>

- [16] 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.
- [17] 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. [Online]. Available: <http://www.oldcitypublishing.com/journals/ijuc-home/ijuc-issue-contents/ijuc-volume-10-numbers-5-6/ijuc-10-5-6-p-473-497/>
- [18] 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.
- [19] 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.
- [20] 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. [Online]. Available: <http://www.nature.com/articles/srep32197>
- [21] D. Volpati, M. K. Massey, D. W. Johnson, A. Kotsialos, F. Qaiser, C. Pearson, K. S. Coleman, G. Tiburzi, D. A. Zeze, and M. C. Petty, "Exploring the alignment of carbon nanotubes dispersed in a liquid crystal matrix using coplanar electrodes," *Journal of Applied Physics*, vol. 117, no. 12, p. 125303, Mar. 2015. [Online]. Available: <https://aip.scitation.org/doi/full/10.1063/1.4916080>
- [22] H. O. Sillin, R. Aguilera, H.-H. Shieh, A. V. Avizienis, M. Aono, A. Z. Stieg, and J. K. Gimzewski, "A theoretical and experimental study of neuromorphic atomic switch networks for reservoir computing," *Nanotechnology*, vol. 24, no. 38, p. 384004, Sep. 2013. [Online]. Available: <https://doi.org/10.1088%2F0957-4484%2F24%2F38%2F384004>
- [23] E. Vissol-Gaudin, "Evolutionary computation based on nanocomposite training: Application to data classification," Doctoral, Durham University, Durham University, 2020. [Online]. Available: <http://etheses.dur.ac.uk/13563/>

Chapter 5

Enhancements to EiM Processors

5.1 Chapter Overview	96
5.2 Experimental System Configuration	97
5.3 Batching	99
5.4 Binary Cross Entropy Objective Function	105
5.5 Regressed Output Layer	110
5.6 Fully Connected Input Layer	114
5.7 Summary	118
Bibliography	119

5.1 Chapter Overview

The previous chapter considered the fundamental framework & function of a typical monolithic Evolution in-Materio (EiM) processor structure, and considered how performance might be boosted by introducing new evolvable parameters into the EiM system. This chapter introduces a number of enhancements to the EiM paradigm. These include adapting the Differential Evolution (DE) algorithm to enable mini-batching of the training data, applying Binary Cross Entropy as the objective function to better place the decision boundary, considering a regressed output layer to better exploit a material processor, and investigating the use of a fully connected input layer to create a ‘smoother’ search space.

These techniques were efficiently investigated using the EiM processor model

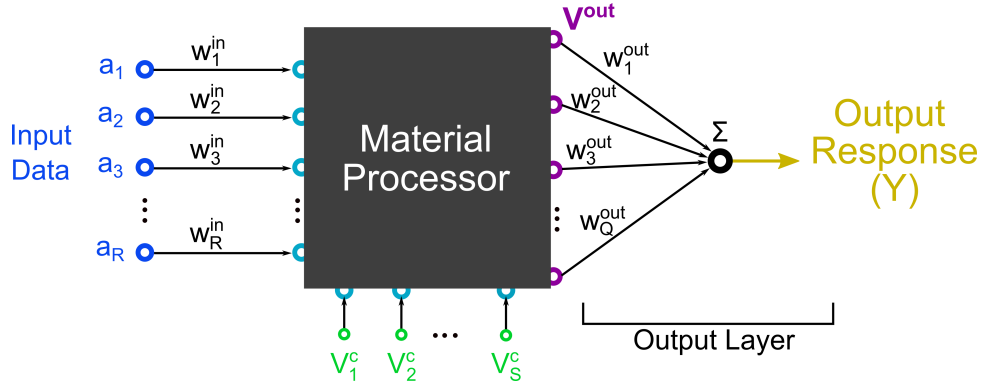


Figure 5.1: Replication of Fig. 4.6 showing a monolithic EiM processor using the expanded set of decision vectors defined in §4.3, where output signals are collected using an output layer and combined to create an overall processor response Y used to make a classification prediction \hat{y} .

discussed in §3.4. Simulated Diode Random Network (DRN) based Configurable Analogue Processors (CAPs) were optimised with DE using the standardised EiM processor model discussed in §4 which used the expanded set of decision variables as seen in Fig. 5.1, resulting in better leveraged performance for classification. Much of this chapter’s work was presented at IEEE International Conference on Rebooting Computing¹ (ICRC) 2021 and published in its peer reviewed proceedings [1].

5.2 Experimental System Configuration

To ensure a reliable investigation into the proposed algorithm and EiM enhancements, a ‘control’ must first be defined to ensure an appropriate comparison is achieved. This section outlines the method & EiM system (including the material, algorithm and selected hyperparameters) used throughout this chapter to measure baseline performance, which modified systems were compared against.

To efficiently investigate the effect and added performance of the different enhancements to monolithic EiM processors, simulated DRN conductive networks were trained using DE. This chapter focuses on the use of the simulated DRN material processor which acts as an exemplar of a highly non-linear material, found to perform well in §4.3.3, and containing properties which are found in nanomaterials.

¹<https://icrc.ieee.org>

To ensure a fair comparison between different modifications proposed later in the chapter, ten DRN materials were generated to act as a proxy for a nanomaterial processor, with the maximum & minimum system voltages selected as $\pm 10V$. Each DRN network had twelve nodes which correspond to electrodes in a physical system, and these materials (defined by specific random seeds) were re-used in all the experiments. Therefore, differences in performance can only be attributed to the algorithm exploiting the materials. However, Evolutionary Algorithms (EAs) are stochastic in nature, which leads to variations in convergence speed. To mitigate this randomness during experimentation, any algorithm (or algorithm modification) under consideration was repeated five times, using the same five random seeds, on each of the ten materials. The mean results from these fifty executions were used to evaluate the proposed algorithm or EiM structural changes.

In this chapter, several datasets containing different numbers of attributes are used. Therefore, it is important that the material electrodes (i.e., simulated material nodes) were appropriately and consistently assigned. Assuming the selected dataset D contains J attributes, then J nodes were allocated as input nodes. To help ensure exploitability, the inputs are projected to higher dimensional output states; here, $J + 2$ nodes were allocated as output nodes. The remaining nodes were allocated as configuration voltage input stimuli, using the typical monolithic EiM processor as seen in Fig. 5.1.

To exploit the DRN networks as EiM processors and perform classification, DE was used to optimise the vector of decision variables \mathbf{X} which included the expanded set of evolvable parameters as described in §4.3 (with $w^{in} \in [-1, 1]$, $w^{out} \in [-2, 2]$ and $V^c \in [-10, 10]$). Specifically, the DE/best/1/bin algorithm was used with a reflection bounds constraint and classification error objective function, as described in §2.3.1 & §2.4.1 respectively. To determine appropriate DE algorithm hyperparameters values, a grid search of the mutation factor (F) and crossover rate (CR) was considered while using a fixed population size of $\lambda = 20$ over twenty epochs, used to classify the con2DDS and sp2DDS datasets. To enable manageable simulation times, here only, just five different generated DRN were used. For each hyperparameter combination, these DRNs were each optimised for classification using the

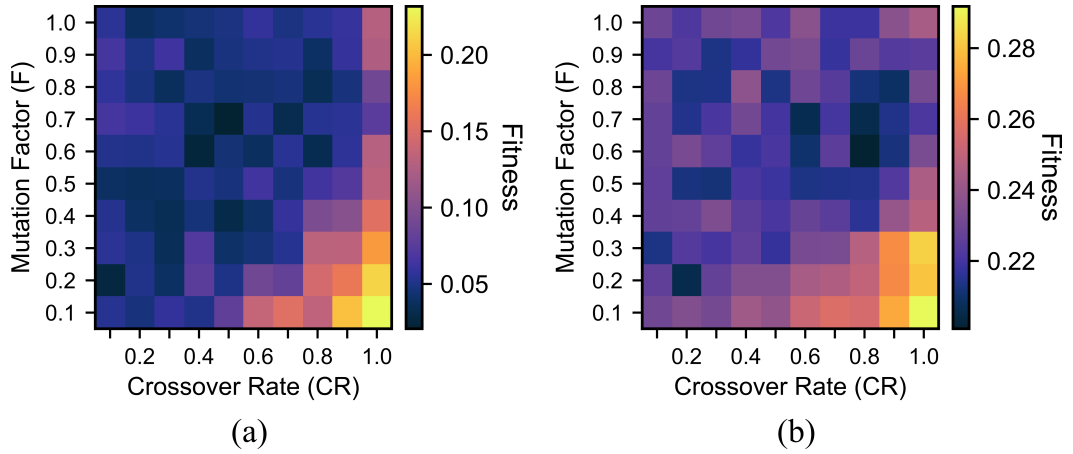


Figure 5.2: Hyperparameter sweep of mutation factor (F) and crossover rate (CR) when using DE to optimise monolithic EiM processors to classify the (a) con2DDS and (b) sp2DDS. Each result is the mean from five DRN conductive networks, each trained as an EiM classifier for 30 epochs, with performance averaged over three DE repetitions.

DE algorithm, which was repeated three times (reusing the same three seeds). A surface plot of the final mean test fitness (i.e., the classification error) is shown in Fig. 5.2, from which it can be observed that a band of darker, lower fitness and well performing hyperparameter combinations appears to exist. In this chapter, a mutation factor of $F = 0.6$, crossover rate of $CR = 0.4$, and population size $\lambda = 20$ was selected, and datasets were split 70% – 30% to create a balanced training and test subset respectively, unless otherwise stated.

5.3 Batching

EiM systems are limited by the available memory provided by the microprocessors used to control the Hardware Interface which interacts with a nanomaterial electrode array. Examples from previous work include a Mbed with 32KB of RAM [2] or the MECOBO with 128KB RAM [3]. As execution speeds within *in-materio* processors improve, possibly up to 100MHz [4] or more, it becomes increasingly important to 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

Algorithm 5.1: Pseudocode for basic DE with batching.

```

Initialise a random population  $\mathbf{p}$ ;
Evaluate initial population fitnesses on batch  $B_0$ ;
Assign the best member of  $\mathbf{p}$  as  $\theta_0$ ;
for  $i = 0, 1, 2, \dots, I$  do
    for  $e = 0, 1, \dots, E$  do
        if  $E \neq 1$  then
            Re-evaluate parent population  $\mathbf{p}$  on batch  $B_j$ ;
        Generate trial population  $\mathbf{t}$ ;
        Evaluate trial population fitnesses on batch  $B_j$ ;
        Update population  $\mathbf{p}$  with respect to  $\mathbf{t}$ ;
        Update the best member  $\theta_i$ ;
    end for
    Evaluate best member  $\theta_I$  using the test data;

```

commonly used to split up the data into smaller groups or ‘batches’ [5], this has several benefits including a smaller memory footprint, more network updates and helping to prevent the model from over fitting (further discussed in §2.5.2). This chapter’s first contribution is the examination of the effect of introducing batching to DE.

5.3.1 Experimental Implementation

As mentioned above, batching (or mini-batching) is a technique used in ANNs [6, 5], which involves dividing up the training data into smaller groups known as batches, as discussed in §2.5.2. The use of small batch sizes leads to a significantly smaller memory footprint, requiring less expensive hardware, and in the case of ANNs is found to improve generalisation performance [5]. 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 a single epoch has occurred.

Previous work introduced batching into a DE algorithm used to perform neuroevolution (i.e., EA based training) of an ANN [7]. Recall from §3.6.1 that the selected classification dataset D is normally split into two subsets: a training set D^{train} used to optimise the system parameters, and an unseen test set D^{test} used to evaluate the unbiased performance of the model. Here, the training set is further split equally into E balanced batches B_0, B_1, \dots, B_{E-1} , where the batch size is some

Algorithm 5.2: Pseudocode for basic DE with batching and validation subset.

```

Initialise a random population  $\mathbf{p}$ ;
Evaluate initial population fitnesses;
Assign the best member of  $\mathbf{p}$  as  $\boldsymbol{\theta}$ ;
 $\boldsymbol{\theta}_0^{global} = \infty$ ;
for  $i = 0, 1, 2, \dots, I$  do
    for  $e = 0, 1, \dots, E$  do
        if  $E \neq 1$  then
            Re-evaluate parent population  $\mathbf{p}$  on batch  $B_j$ ;
            Generate trial population  $\mathbf{t}$ ;
            Evaluate trial population fitnesses on batch  $B_j$ ;
            Update population  $\mathbf{p}$  with respect to  $\mathbf{t}$ ;
            Update the best member  $\boldsymbol{\theta}$ ;
        Considering the validation data, update the global best solution  $\boldsymbol{\theta}_{i+1}^{global}$ ;
    Evaluate best member  $\boldsymbol{\theta}_{I+1}^{global}$  using the test data;

```

fraction of the total number of training data instances $bs \approx \frac{1}{E}K^{train}$. These batches are used sequentially to train the population, where each batch results in a single generational update, and the best population member $\boldsymbol{\theta}$ is tracked. However, to achieve a reliable comparison between the parent/previous generation population (which was evaluated using a batch B_{e-1}) and a new trial/child population (evaluated using the current batch B_j), it is necessary to compare the trial fitness to a re-evaluated parent fitness (using the current batch B_j). Therefore, when batching, the training data must be processed twice, increasing the total number of training data computations carried out per epoch. The pseudocode describing this optimisation process is presented in Algorithm 5.1. When $E = 1$, there is only one batch which is the entire training set D^{train} ; in this special case, there is no need to re-evaluate the parent population, and the process is identical to the original DE Algorithm 2.1.

Similarly, an implementation of an algorithm which exploits both batching and a validation data subset can be considered. In this case, the dataset D is split into three subsets: a training set D^{train} , a validation set D^{valid} , and a test set D^{test} . Again, the training data is split equally into E batches, and the test data is used for the unbiased evaluation of the system's performance. However, the validation set is

used to provide a uniform evaluation of the population at the end of an epoch. While not implemented here, it is noted that validation fitness could be used to tune the system’s hyperparameters or monitor performance to implement early stopping [8]. Here, a ‘global’ best member θ^{global} is introduced, which tracks the most successful solution found for the validation subset – updated only when a population member achieves a lower fitness during a validation evaluation at the end of an epoch. The pseudocode describing the batching with validation optimisation process is presented in Algorithm 5.2.

5.3.2 Performance

In this section, the effect of batching using Algorithm 5.1 with a fixed computational budget of 5×10^5 training data instances (rather than a selected number of epochs) is considered, at which point the EA terminated. Additional published work [1] considers the use of the validation dataset under similar conditions. The more challenging con2DDS and BankNote datasets were considered and used to compare the Algorithm 5.1 with varying batch sizes.

Before examining experimental results, the effect of batching on the optimising DE algorithm’s speed of convergence is considered. When batching, the parent fitness must be re-computed for each new batch. Therefore, double the amount of training data is utilised per epoch than the original Algorithm 2.1 or the special case with no batching (i.e., Algorithm 5.1 with $E = 1$) would require. When using the basic and batching algorithms without a validation set, it is assumed that the data is split 70% – 30% to create a balanced training and test subset respectively. Similarly, if using a validation step as in Algorithm 5.2, it is assumed the dataset would instead be split 60% – 20% – 20% to create the training, validation and test subset respectively. Therefore, the number of computations per epoch “ n_{epoch} ” for the basic DE Algorithm 2.1 would be:

$$n_{epoch}^{basic} = 0.7K , \quad (5.3.1)$$

where K is the number of data instances in the selected dataset D . For Algorithm

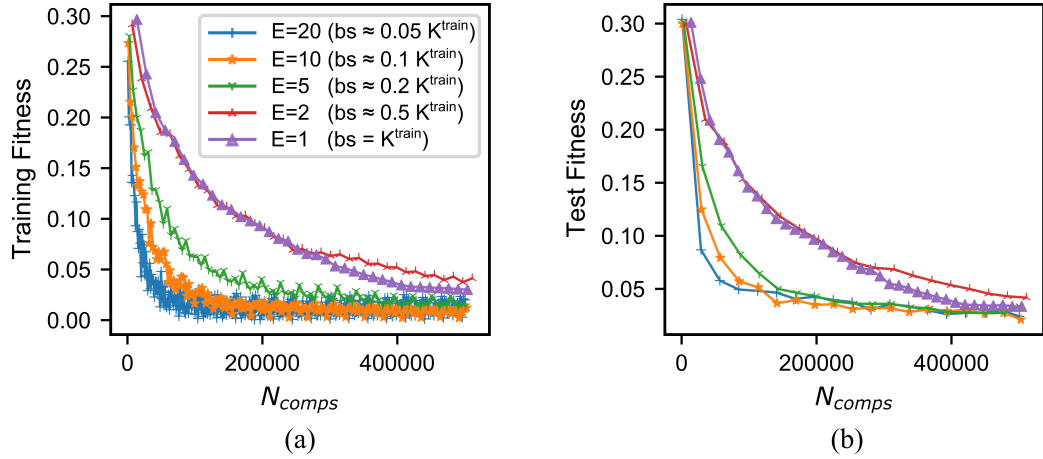


Figure 5.3: The (a) training and (b) test fitness convergence when optimising DRNs as EiM processors to classify the con2DDS dataset, using Algorithm 5.1 with $E = 1, 2, 5, 10, 20$.

5.1, when $E > 1$, the number of computations per epoch n_{epoch}^{batch} is:

$$n_{epoch}^{batch} = 2 \times 0.7K = 1.4K , \quad (5.3.2)$$

and for the batching with validation step Algorithm 5.2, when $E > 1$, the number of computations per epoch n_{epoch}^{valid} is:

$$n_{epoch}^{valid} = 2 \times \underbrace{0.6K}_{train} + \underbrace{0.2K}_{valid} = 1.4K . \quad (5.3.3)$$

So, if the algorithms were compared for an identical number of epochs, the original Algorithm 2.1 would process significantly more data in the allotted evolutionary period. In order to provide a meaningful comparison, the systems should instead be evolved using a fixed number of computations (N_{comps}) i.e., total allotted ‘computational’ budget.

Having established the fair termination criterion used, the results are now explored in more detail. The evolution of the mean best training fitness and the mean test fitness when solving the con2DDS is shown in Fig. 5.3a & 5.3b respectively. Similarly, the classification results for the Banknote dataset are shown in Fig. 5.4. When using small batch sizes, training fitness converged much less smoothly in a jagged or ‘noisy’ manner, caused by differences in fitness that each batch of data

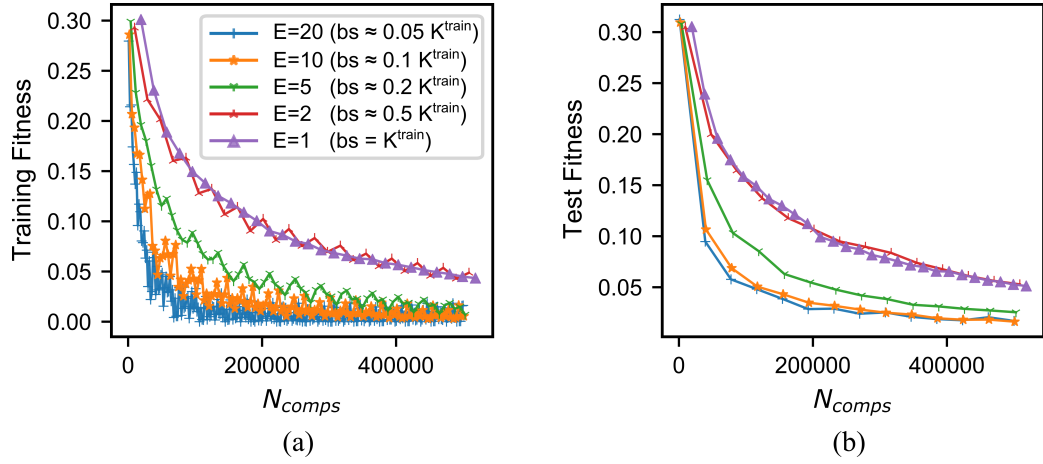


Figure 5.4: The (a) training and (b) test fitness convergence when optimising DRNs as EiM processors to classify the Banknote dataset, using Algorithm 5.1 with $E = 1, 2, 5, 10, 20$.

achieves. As the batch size decreased (i.e., the number of batches E was increased), the EiM processors converged substantially faster and achieved a lower mean final test fitness.

Significantly, the introduction of batching led to an increased number of generations per epoch. Assuming the same data subset split as before, the number of computations (i.e., number of data instances processed) used for a single generational update γ for the original basic DE algorithm is:

$$\gamma^{basic} = 0.7K , \quad (5.3.4)$$

and the number of computations per generation when batching (Algorithm 5.1 where $E > 1$) is on average:

$$\gamma^{batch} = \frac{1.4K}{E} , \quad (5.3.5)$$

and similarly the number of computations per generation for the batching and validation Algorithm 5.2 (when $J > 1$) would be on average:

$$\gamma^{valid} = \frac{1.4K}{E} . \quad (5.3.6)$$

The large increase in convergence speed was attributed to this faster rate of useful generational updates when batching. In other words, Algorithm 5.1 used fewer data

instances per generation than the basic DE algorithm when $E > 2$. This should allow physical EiM devices to be trained more quickly and be more computationally efficient. However, if the batch size is too small, it would be expected that too little information from the dataset D would be present, leading to large variations in training population fitness for each batch. This could result in poor population updates and a lack of EA convergence to an acceptable solution.

In this work, the same sequence of batches were used each epoch. However, shuffling the examples (i.e., the order batches are used) would likely lead to faster learning [6, 9, 8]. Similarly, other methods could be borrowed from progress made on ANN, such as batch normalisation [5].

5.4 Binary Cross Entropy Objective Function

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 [10] 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. It is proposed that incorporating such a ‘Figure of Merit’ into the EAs objective function will lead to EiM systems with superior decision boundary placement compared to systems operating with the commonly implemented classification error [11, 2, 10]. To this end, this section investigates the adaptation of Binary Cross Entropy (BCE) as a modified objective function for an EiM’s exploiting EA for the first time.

5.4.1 Experimental Implementation

Classification error (described in §2.4.1) 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

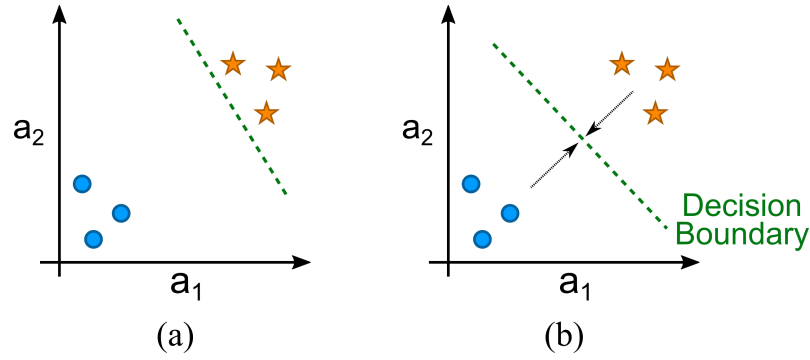


Figure 5.5: Example decision boundary for the (a) discrete classification error objective function where evolved systems with 100% accuracy may be susceptible to noise, and (b) Binary Cross Entropy objective function which uses information from the classified data to maximise the likelihood of successful classification [1]. ©2021 IEEE.

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 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 (i.e., better or worse proximity to the data) due to the inevitable presence of noise, as shown in Fig. 5.5a. 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 [10], a more consistently favourable decision boundary should be achieved, as shown in Fig. 5.5b.

BCE or log loss is an established loss function for machine learning binary classification tasks, described in §2.4.3. Cross entropy generates larger loss (i.e., fitness) values as the predicted probability of a label diverges from the value of the actual label. To adapt BCE as an objective function for a typical EiM system shown in Fig. 5.1, the raw output of the EiM processor must be first constricted to $\in [0, 1]$. To do this, a sigmoid function $\sigma(k)$, as seen in Fig. 5.6a, is used to constrain the output response Y , such that for a particular input data instance k :

$$\sigma(k) = \frac{1}{1 + e^{-Y(k)}} , \quad (5.4.7)$$

where $Y(k)$ is the network response of the system defined in Eq.(4.3.3). Finally, as

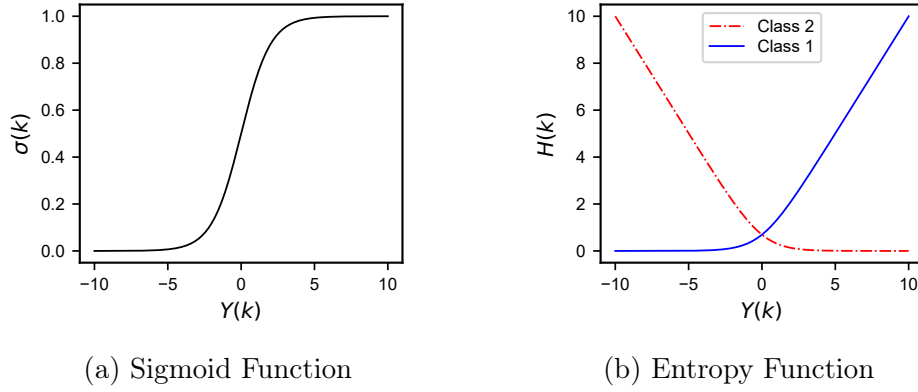


Figure 5.6: Outputs of the sigmoid $\sigma(k)$ and entropy $H(k)$ functions for the collected network response Y of the material processor for a particular data instance k . The entropy function is different depending on the true value of the class.

described in §2.4.3, the entropy or log loss $H(k)$ is then computed:

$$H(k) = \begin{cases} -\ln(1 - \sigma(k)), & \text{if } y(k) = 1 \\ -\ln(\sigma(k)), & \text{if } y(k) = 2 \end{cases}, \quad (5.4.8)$$

and the fitness of a group of training data can be determined using the BCE objective function:

$$\Phi_{bce} = \frac{1}{K} \sum_{k=1}^K H(k). \quad (5.4.9)$$

This 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. 5.6b (recalling from §3.3 that an instance producing $Y(k) < 0$ is predicted as class 1, and with $Y(k) \geq 0$ is predicted as class 2).

5.4.2 Performance

The EiM processors were used to classify the 2DDS, con2DDS and Banknote datasets, over 30 epochs, with the basic DE Algorithm 2.1 (i.e., without mini-batching). They were firstly trained using the classification error (‘error’) and then using the Binary Cross Entropy (‘BCE’) objective function. A histogram of the final test output response Y for all the trained processors (and their training repetitions) is plotted in

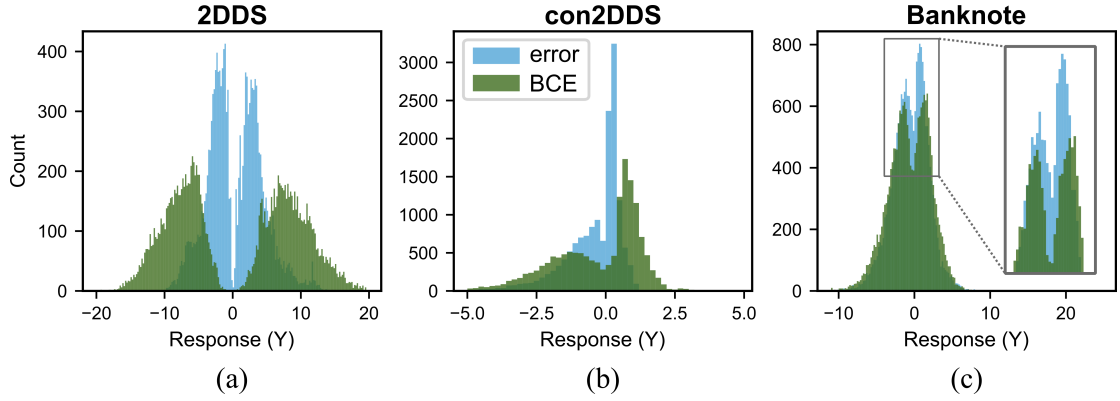


Figure 5.7: Histogram of the accumulated final test data outputs (i.e., material processor responses Y) for all material and algorithm repetitions trained on the (a) 2DDS, (b) con2DDS, and (c) Banknote datasets comparing the use of a classification error and BCE fitness metric. (Bins=0.2)

Fig. 5.7a, 5.7b and 5.7c for the 2DDS, con2DDS and Banknote datasets respectively. The error evolved systems produced output responses much closer to the decision boundary (i.e., $Y = 0$). By contrast, the Binary Cross Entropy evolved systems generate output responses which are ‘pushed’ further away from the decision boundary (this is particularly visible in Fig. 5.7a), meaning these EiM processors are placing boundaries between the data much more successfully.

The BCE objective function requires the output layer (Eq.(4.3.3)) to evolve and exploit the sigmoid (Eq.(5.4.7)) to learn which data instances are considered ‘deep’ or not. Therefore, insufficient flexibility within the output layer, such as inappropriate maximum and minimum output weights, could limit the final fitness an EiM processor could achieve. It should be noted that Algorithm 5.2 could grant more system flexibility, with the opportunity to select different objective functions for the training and validation evaluations, e.g., allowing the optimisation of hyperparameters using a specific validation fitness metric.

To help quantify the performance of boundary placement, ROC curves were produced for the aggregate of these results (from the five repetitions on each of the ten materials) and used to compare the models, as shown in Fig. 5.8. The Area Under Curve (AUC) value indicates how well a model can distinguish between classes [12]. The ‘error’ and ‘BCE’ evolved models achieve similar AUC values, but these results do not capture the differences of boundary ‘sharpness’.

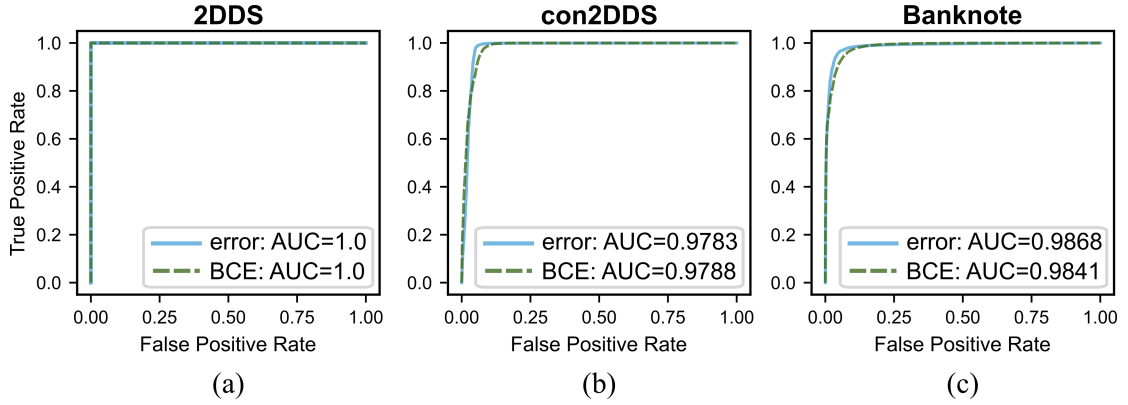


Figure 5.8: Receiver Operating Characteristic (ROC) for the results of all material and algorithm repetitions trained on the (a) 2DDS, (b) con2DDS, and (c) Banknote datasets comparing the use of a classification error and BCE fitness metric.

To further investigate the performance of boundary placement, the different systems' resilience to noise was examined, the test data was combined with varying levels of Gaussian noise $N(0, \sigma^2)$ and then evaluated on the trained systems. A set of five random seeds was used to create five noisy versions of the test data subset, these were then combined to create an enlarged 'noisy' test subset, used to allow a fair comparison between the trained EiM systems. The reduction in mean test accuracy was plotted against the variation of the standard deviation of introduced Gaussian noise, as seen in Fig. 5.9. As expected, the BCE evolved system was more resilient to a depreciation in performance caused by noisy inputs, due to the decision boundary being placed in a location which provides a better probabilistic

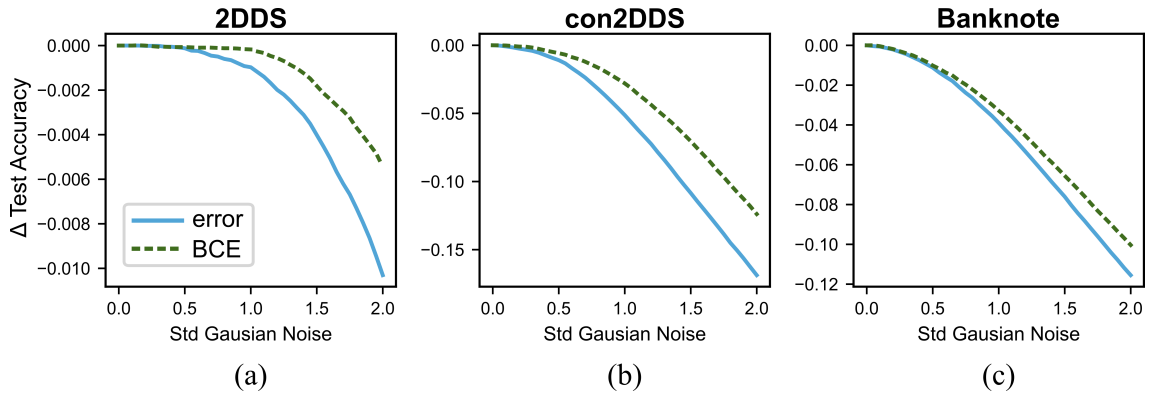


Figure 5.9: Plot showing the reduction in mean test accuracy of the trained EiM systems as increasing levels of Gaussian noise is introduced to the test data on the (a) 2DDS, (b) con2DDS, and (c) Banknote datasets, comparing the classification error and BCE evolved systems.

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.

5.5 Regressed Output Layer

Ensuring that a material is consistently and successfully exploited for EiM has proved challenging. Work in the §4.3 showed that a material's outputs can require some level of more advanced interpretation, else the performance of an EiM processor might be hidden. Indeed, materials with monotonically increasing Current-Voltage (IV) characteristics will require some output voltage interpretation process of combination and/or threshold comparison to solve complex classification problems. Other materials, such as those with IV characteristics containing Negative Differential Region (NDR) [13, 4], can achieve more complex output behaviour that might require less complex interpretation. EiM processors have therefore traditionally contained some evolvability in the output interpretation, e.g. output weights [14, 11] or evolvable thresholds [2]. In this section, the introduction of a regression step into the EA algorithm is proposed, used to produce a readout (i.e., output) layer for the material, and enhance the assessment and exploitation of a CAP's output states. This results in an optimised output layer every time a population member is evaluated, a significant improvement compared to evolving the output layer.

Such an implementation begins to resemble an EA optimised Extreme Learning Machine [15], considered further in §6.3. Similarly, if extended into the temporal domain, the use of regression brings the EiM computational paradigm one step closer to physical Reservoir Computing (RC) [16].

5.5.1 Experimental Implementation

Work in §4.2.2 has shown that the use of output weights was important for EiM processors, which allowed a material processor’s output states to be combined and produce more complex Decision Boundaries which significantly improved results for classification problems. This suggests that the interpretation scheme used to extract information from an EiM material’s outputs are of great importance, and if one such scheme is inappropriately chosen, it may “hide” the true computational performance possessed by the material.

As discussed above, interpretation schemes for EiM processors have typically been fixed or are evolved using a few parameters such as classification thresholds or output weights. Instead, this work used ridged regression [17] to efficiently generate and optimise the 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’s 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.

5.5.2 Performance

Standard EiM processors using an output layer defined and evolved in the decision vector \mathbf{X} were used to classify the con2DDS, sp2DDS and Banknote datasets. The basic DE Algorithm 2.1 (i.e., without mini-batching) was used for 40 epochs with a classification error (Φ_{error}) objective function. 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) was generated. The convergence of the mean test fitness of the Standard and Regressed EiM processors for the different datasets is shown in

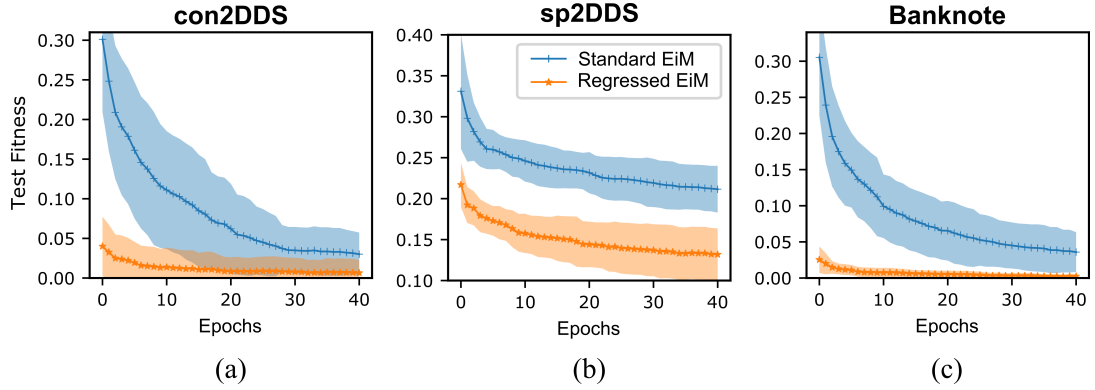


Figure 5.10: Evolution of the mean test fitness, with standard deviation, comparing the Standard evolved and Regressed EiM training methods on the (a) con2DDS, (b) sp2DDS, and (c) Banknote dataset.

Fig. 5.10, and the test results for the first and final epoch are reported in Table 5.1.

After just one epoch, the Regressed EiM processors achieved a 86.7%, 34.6% and 91.7% lower mean test fitness score than the Standard evolved EiM processors for the con2DDS, sp2DDS and Banknote datasets respectively. Therefore, much of the performance of the EiM processors is clearly tied to the output interpretation scheme. For Standard EiM, the interpretation scheme was dependent on an output layer defined by the output weights contained in the decision vector, and possibly well performing configured materials are being hidden by poorly evolved output weights. By contrast, the use of ridge regression allowed the output layer to be independently optimised, and therefore ensure that a particular material configuration and its respective outputs are exploited fully for the target problem.

As the evolutionary period progressed, the mean test fitness of the Standard EiM

Table 5.1: Final test results for the Standard evolved and Regressed EiM, using the classification error objective function.

Dataset	Processor Type	First Epoch			Final Epoch		
		$\bar{\Phi}$	$\text{std}(\Phi)$	Φ^*	$\bar{\Phi}$	$\text{std}(\Phi)$	Φ^*
con2DDS	Standard EiM	0.3011	0.0914	0.0800	0.0301	0.0273	0.0000
	Regressed EiM	0.0400	0.0374	0.0000	0.0065	0.0160	0.0000
sp2DDS	Standard EiM	0.3311	0.0702	0.2467	0.2115	0.0286	0.1600
	Regressed EiM	0.2170	0.0270	0.1667	0.1318	0.0320	0.0600
Banknote	Standard EiM	0.3053	0.0808	0.0922	0.0359	0.0276	0.0000
	Regressed EiM	0.0253	0.0183	0.0000	0.0024	0.0030	0.0000

processors rapidly improved. This was due to the optimisation of both the material configuration and the interpretation scheme. However, in the case of Regressed EiM, the performance gains were only driven by an improvement of the material configuration. While the Regressed EiM systems achieved good initial guesses, further optimisation by the DE algorithm did still lead to performance improvements. After 40 epochs, the mean test fitness of the Regressed EiM processors reduced by 83.8%, 39.3% and 90.5% for the con2DDS, sp2DDS and Banknote dataset respectively. It should be expected 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.

The introduction of a regression stage allowed for the efficient generation of an exploiting readout layer, without the need to optimise output configuration parameters within the slower EA process. These results show that Regressed EiM presents a much better representation of the true capabilities of the material processor, and can significantly improve the speed of convergence to an acceptable classification accuracy. These benefits should translate directly to practical examples of EiM systems, such as Single Walled Carbon Nanotube (SWCNT) networks [18, 19, 20]. Additionally, the regression optimised output weights are not bounded (unlike DE's decision vector), and this flexibility may also allow smaller, more subtle differences in a nanomaterials output voltages to be identified and exploited.

The EiM processors are also compared to the application of ridge regression on the raw data, shown in Table 5.2, which both the Standard EiM and Regressed EiM approaches outperform. The material processor is acting similarly to a kernel, in this case exploiting the complex properties of the conductive network to transform input data into useful, higher dimensional representations. This is very similar to the behaviour of reservoirs in RC [16]. 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 instead be exploited to produce unconventional processors. It is hypothesised that extending the EiM paradigm to

Table 5.2: Final mean \bar{A} and best A^* accuracy test results for the Evolved and Regressed EiM compared to some common *sklearn* [17] classification methods.

Dataset	Processor Type	\bar{A}	A^*
con2DDS	Standard EiM	0.9699	1.0000
	Regressed EiM	0.9935	1.0000
	Ridge Regression	-	0.5233
	Random Forest	-	1.0000
sp2DDS	Standard EiM	0.7885	0.8400
	Regressed EiM	0.8682	0.8682
	Ridge Regression	-	0.7367
	Random Forest	-	0.9767
Banknote	Standard EiM	0.9641	1.0000
	Regressed EiM	0.9976	1.0000
	Ridge Regression	-	0.9806
	Random Forest	-	0.9976

RC will lead to more flexible reservoirs and performance gains, warranting further research.

5.6 Fully Connected Input Layer

The shuffle gene was previously introduced (§4.3) as a method for rearranging the input order of the applied input voltages (both input data and stimuli) and allow the *in-materio* processor to exploit different internode IV characteristics for the task at hand. However, the shuffle gene is a discrete integer (used to index a particular input permutation) which creates discontinuities in the search landscape, potentially impacting reliable convergence.

Here, the use of a fully connected layer is proposed, similar to a linear layer used in ANNs. This would create a smoother objective function fitness landscape and allow a gradual exploration of internode IV characteristics, likely boosting the performance of optimising algorithms.

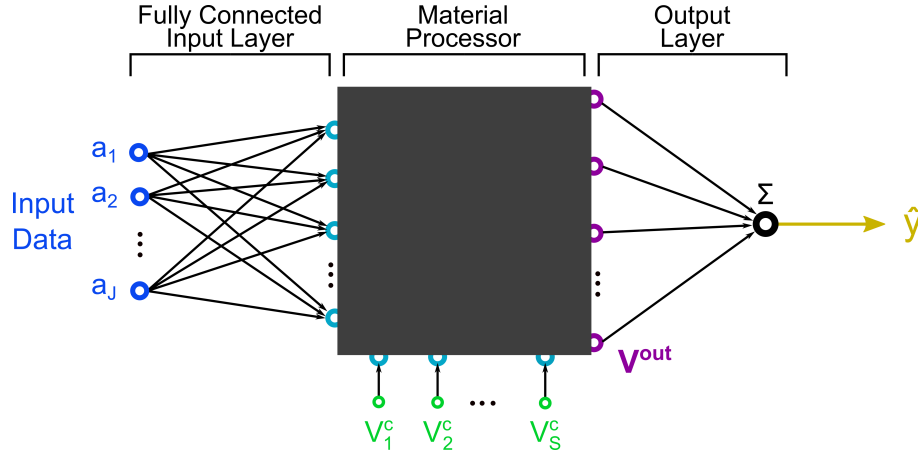


Figure 5.11: Monolithic EiM processor configured for binary classification, using a *fully connected input layer* which combines input attributes to produce an input ‘data’ voltage value.

5.6.1 Experimental Implementation

As discussed in §3.3 and §4.3, the attributes $a_j(k)$ of each data instance k are typically ‘directly connected’ to a corresponding input node/electrode:

$$V_r^{in}(k) = w_r^{in} \times a_r(k) , \quad (5.6.10)$$

where V_r^{in} is the applied voltages to the data driven input electrodes r , w_r^{in} are corresponding input weights, the total number of data driven input electrodes R is equal to the total number of data attributes J , and that a shuffle gene G_{sh} could be used to select a particular input permutation of the arrangement of the input nodes.

Instead, a fully connected linear layer was implemented, as seen in Fig. 5.11, where each input voltage was a weighted sum of the input attributes. This is equivalent to using an ANN layer with linear activation functions (see §2.5.1 for more details). Therefore, the voltage applied to each data input node becomes:

$$V_r^{in}(k) = \mathbf{w}_r^{in} \cdot \mathbf{a}(k) = \sum_j^J w_{rj}^{in} a_j(k) , \quad (5.6.11)$$

where V_r^{in} is the applied voltages to the data driven input electrodes r , $a_j(k)$ is the j^{th} input attribute for data instance k , $\mathbf{w}_r^{in} = [w_{r1}^{in}, w_{r2}^{in}, \dots, w_{rj}^{in}]$ is the vector of input weights, and $\mathbf{a}(k) = [a_1, a_2, \dots, a_j]$ is the vector of input attributes. Note that

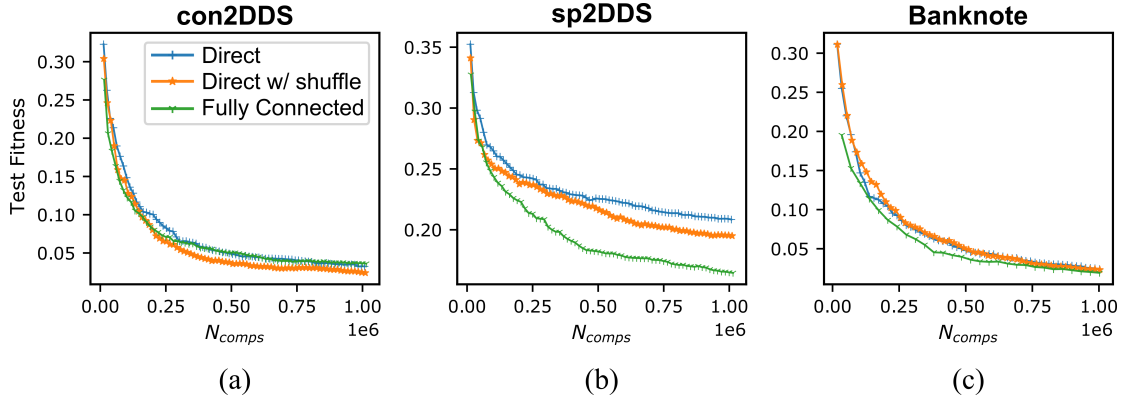


Figure 5.12: Convergence of the mean test fitness when using different styles of evolved input layers to optimise an EiM processor to classify the (a) con2DDS, (b) sp2DDS, and (c) Banknote datasets.

the applied voltage must remain within the range of a system’s real possible values, so a ‘clipped’ boundary condition is implemented where $V^{min} \leq V_r^{in}(k) \leq V^{max}$.

These different input schemes use different evolvable parameters and differently sized decision vectors \mathbf{X} . Therefore, to maintain fair comparisons, the same population size could no longer be used across all the systems. Instead, the population size was scaled with respect to the number of evolvable parameters d (i.e., the dimension of the decision vector); here, selected to be $\lambda = 1.5 \times d$. Additionally, similar to the work in §5.3, some of these differently sized systems are likely to be advantaged if a fixed number of epochs are used to optimise the system. Therefore, these systems were evolved for a fixed computational budget (N_{comps}) i.e., a fixed number of data instances which are processed.

5.6.2 Performance

The basic DE Algorithm 2.1 (i.e., without mini-batching) was used to optimise the EiM processors using a classification error (Φ_{error}) objective function, and a budget of $N_{comps} = 1 \times 10^6$. These were used to classify the con2DDS, sp2DDS and Banknote datasets using the basic directly connected input layer (Eq.(5.6.10)), the directly connected input layer with the shuffle gene evolvable parameter, and also the fully connected input layer (Eq.(5.6.11)). The mean test fitness convergence results are shown in Fig. 5.12, and the final test results are presented in Table 5.3.

The directly connected input layer without shuffle performed consistently worst

since it lacked the ability to explore new inter-node IV characteristics, as discussed in §4.2.2. The directly connected input layer with shuffle performed well; the shuffle gene allowed for discrete ‘jumps’ around the search space, which in the case of the con2DDS allows for fast convergence to well performing solutions.

The sp2DDS presents the most challenging dataset, with its spiral formation requiring the material processors’ non-linearities to be combined very successfully. With this dataset, it was clearly seen that the fully connected layer significantly outperforms the directly connected layers. The combination of attributes in the input stage allows for a smooth search space and much more successful ‘fine-tuning’ such that better solutions can be discovered.

The fully connected input layer allows for more flexible system topology, where a one-to-one match of the number of attributes and number of allocated input nodes is no longer required. However, a fully connected input layer might significantly increase the number of evolvable parameters within the system. Specifically, J times the number of input weight parameters are required, potentially effecting the efficiency (i.e., number of computations per amount of data) as systems scale to larger sizes.

Table 5.3: Final test results for the different input scheme for the monolithic EiM processors.

Dataset	Input Connection Layer	$\bar{\Phi}$	std(Φ)	Φ^*
con2DDS	Direct	0.0325	0.0472	0.0000
	Direct with Shuffle	0.0240	0.0271	0.0000
	Fully Connected	0.0361	0.0305	0.0000
sp2DDS	Direct	0.2084	0.0491	0.1533
	Direct with Shuffle	0.1950	0.0305	0.1300
	Fully Connected	0.1646	0.0216	0.1167
Banknote	Direct	0.0239	0.0253	0.0000
	Direct with Shuffle	0.0231	0.0179	0.0000
	Fully Connected	0.0191	0.0170	0.0000

5.7 Summary

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. In this chapter, experiments were conducted using simulations of physically realisable circuits used to produce conductive Diode Random Networks which were paired with DE to produce EiM processors. These were used to efficiently experiment and explore improvements to the EiM framework and algorithm.

The DE algorithm was adapted to enable mini-batching of the training data. It was found that smaller batch sizes enabled the algorithm to converge more quickly to a final solution due to the reduced number of computations needed per generational update. However, extremely small batch sizes (with respect to the training subset) might not contain enough information from the dataset, possibly leading to poor generational updates.

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 ensured that a successful comparison between a trial population member and its parent would always be possible, leading to more reliable & smoother convergence, but also superior decision boundary placement.

Regression was used as an additional intermediate step to train the output (i.e., readout) layer of a 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 outperform the standard EiM algorithm. This highlights that slow to evolve, or inappropriate interpretation schemes can "hide" the performance of well configured materials.

Finally, the implementation of a fully connected input layer was considered. This allowed for a continuous transition between different material IV characteristics 'states' previously accessed using the shuffle gene, which re-ordered the directly

connected input arrangement. The fully connected layer allowed for a smoother solution search space, and meant significantly better classification performance could be achieved on highly non-linear datasets.

EiM processors have the potential to harness complex nanomaterial properties to produce efficient, unconventional computing devices. This chapter shows how a traditional EiM system can be adapted and enhanced to produce more robust and better performing unconventional processors, ensuring future physical nanomaterial processors are fully exploited.

Bibliography

- [1] Benedict. A. H. Jones, N. Al Moubayed, D. A. Zeze, and C. Groves, “Enhanced methods for Evolution in-Materio Processors,” in *2021 International Conference on Rebooting Computing (ICRC)*, Nov. 2021, pp. 109–118. [Online]. Available: <https://doi.org/10.1109/icrc53822.2021.00026>
- [2] 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. [Online]. Available: <http://ieeexplore.ieee.org/document/8123677/>
- [3] 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.
- [4] 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. [Online]. Available: <https://www.nature.com/articles/s41586-019-1901-0>
- [5] D. Masters and C. Luschi, “Revisiting Small Batch Training for Deep Neural Networks,” *arXiv:1804.07612 [cs, stat]*, Apr. 2018. [Online]. Available: <http://arxiv.org/abs/1804.07612>
- [6] Y. Bengio, “Practical Recommendations for Gradient-Based Training of Deep Architectures,” in *Neural Networks: Tricks of the Trade: Second Edition*, ser. Lecture Notes in Computer Science, G. Montavon, G. B. Orr, and K.-R. Müller, Eds. Berlin, Heidelberg: Springer, 2012, pp. 437–478. [Online]. Available: https://doi.org/10.1007/978-3-642-35289-8_26

- [7] M. Baiocchi, G. Di Bari, A. Milani, and V. Poggioni, “Differential Evolution for Neural Networks Optimization,” *Mathematics*, vol. 8, no. 1, p. 69, Jan. 2020. [Online]. Available: <https://www.mdpi.com/2227-7390/8/1/69>
- [8] S. Ruder, “An overview of gradient descent optimization algorithms,” Jun. 2017. [Online]. Available: <http://arxiv.org/abs/1609.04747>
- [9] Y. A. LeCun, L. Bottou, G. B. Orr, and K. R. Müller, “Efficient backprop,” in *Neural Networks*, ser. Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics). Springer Verlag, 2012, pp. 9–48. [Online]. Available: <http://www.scopus.com/inward/record.url?scp=84872543023&partnerID=8YFLogxK>
- [10] 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.
- [11] 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.
- [12] S. Gupta and A. Gupta, “Dealing with Noise Problem in Machine Learning Data-sets: A Systematic Review,” *Procedia Computer Science*, vol. 161, pp. 466–474, Jan. 2019. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S1877050919318575>
- [13] 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. [Online]. Available: <http://www.nature.com/articles/nnano.2015.207>
- [14] B. A. H. Jones, J. L. P. Chouard, B. C. C. Branco, E. G. B. Vissol-Gaudin, C. Pearson, M. C. Petty, N. Al Moubayed, D. A. Zeze, and C. Groves, “Towards Intelligently Designed Evolvable Processors,” *Evolutionary Computation*, pp. 1–23, Mar. 2022. [Online]. Available: https://doi.org/10.1162/evco_a_00309
- [15] T. Matias, R. Araújo, C. H. Antunes, and D. Gabriel, “Genetically optimized extreme learning machine,” in *2013 IEEE 18th Conference on Emerging Technologies Factory Automation (ETFA)*, Sep. 2013, pp. 1–8.
- [16] 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. [Online]. Available: <http://www.sciencedirect.com/science/article/pii/S0893608019300784>

- [17] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and É. Duchesnay, “Scikit-learn: Machine Learning in Python,” *Journal of Machine Learning Research*, vol. 12, no. 85, pp. 2825–2830, 2011. [Online]. Available: <http://jmlr.org/papers/v12/pedregosa11a.html>
- [18] 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. [Online]. Available: <http://aip.scitation.org/doi/10.1063/1.4915343>
- [19] 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. [Online]. Available: <http://www.oldcitypublishing.com/journals/ijuc-home/ijuc-issue-contents/ijuc-volume-10-numbers-5-6/ijuc-10-5-6-p-473-497/>
- [20] 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. [Online]. Available: <http://www.nature.com/articles/srep32197>

Chapter 6

In-Materio Neural Networks

6.1 Chapter Overview	122
6.2 In-Materio Neural Network Structure	124
6.3 In-Materio Extreme Learning Machines	128
6.4 Neuroevolution of Physical iM-NNs	135
6.5 In-Materio AutoEncoders	146
6.6 Summary	152
Bibliography	153

6.1 Chapter Overview

There is a strong desire to produce efficient but powerful computing and Machine Learning (ML) at “the edge” [1]. Physical analogue systems have remained an attractive option, due to many analogue devices’ high theoretical throughput and low-energy consumption [2]. However, traditional monolithic Evolution in-Materio (EiM) processors that use a single substrate are unlikely to scale well due to diminishing interactions as inter-electrode distances grow.

This chapter tackles scaling and performance issues by drawing on ML and Artificial Neural Network (ANN) concepts, to move beyond the typical monolithic EiM processors discussed in the previous two chapters. By stacking configurable analogue processing units in parallel and operating them as realisations of ‘physical’ or ‘*in-materio*’ neurons, an in-Materio Neural Network (iM-NN) was constructed.

Similarly, *virtual* iM-NNs were introduced, where a single physical neuron is re-used as several virtual neurons.

Methods to train these novel iM-NNs are needed. Back propagation and gradient descent are the cornerstone techniques used in modern ANN ML, as described in §2.5.2. However, conductive substrates and physical neurons can be hard to model, so are often treated as black boxes, making it difficult to perform such standard ML techniques. Optimisation methods which have been found to work well when training non-differentiable activation functions are required. Extreme Learning Machines (ELMs) are one such non-iterative method [3, 4] which leverages a randomly initialised ANN, only performing a forward pass through the network and attempting to exploit the Hidden Layer (HL) output states using regression. Similarly, neuroevolution avoids the need for back propagation by using a stochastic, iterative Evolutionary Algorithm (EA) to train ANNs.

Firstly, it was observed that the physical substrates in EiM processors operated similarly to an ELM's randomly initialised ANN, projecting inputs to new, useful, output states. Drawing from this, an in-Materio Extreme Learning Machine (iM-ELM) system was proposed and efficiently investigated using Diode Random Network (DRN) based Simulated in-Materio (SiM) neurons. Similarly, *virtual* iM-ELMs were considered for the first time. This novel iM-ELM method was presented at the Seventeenth International Conference on Parallel Problem Solving from Nature ¹ (PPSN XVII) 2022 and published in its peer reviewed proceedings [5].

Following this, more typical iterative based training methods were considered using iM-NNs constructed with conductive Lambda Diode Network (LDN) based Physical in-Materio (PiM) neurons. These were investigated in a lab, using the Raspberry Pi and custom Hardware Interface experimental setup described in §3.5. A comparison of different neuroevolution training methods for these iM-NNs was performed for the first time, and considered the effect of batching or tuning population size when operating under computational budget limited scenarios. This work has been submitted for publication [6].

¹<https://ppsn2022.cs.tu-dortmund.de>

Finally, having demonstrated the feasibility of iM-NNs and investigated different training methods, more complex Neural Network (NN) structures were investigated. Specifically, a realisation of a novel in-Materio AutoEncoder was implemented in a lab using LDN based PiM neurons. It was trained similarly to a neuroevolved ELM [7], using ridge regression to optimise the output layer and an EA to train the remaining system parameters. This was used to successfully perform dimensionality reduction on a handwritten digits dataset.

6.2 In-Materio Neural Network Structure

In this section, a new method to operate and exploit several material substrates in parallel is proposed. Traditionally, as described in Chapters 2 & 4, EiM processors have used a single nanomaterial substrate with one-to-one input-attribute to input-electrode mappings, where each attribute is applied as a voltage to a corresponding input node/electrode. However, as datasets become more complex with more attributes, the size of a device's substrate would need to physically grow. It can be postulated that in real microelectrode-based nanomaterial processors, larger networks might lead to fewer 'interactions' between distant electrodes, leading to poorer performance, i.e., 'monolithic' EiM processors may struggle to scale as the data does.

In order to overcome this problem, inspiration from artificial single hidden layer feedforward neural networks (SLFNs) can be taken, as shown in Fig. 6.1, suggesting that conventional EiM processors should be stacked in parallel. Therefore, conductive network or nanomaterial based Configurable Analogue Processors (CAPs) are instead operated more similarly to an '*in-materio*' or 'physical' neuron. These physical neurons can be used to construct a Neural Network like structure, referred to here as an iM-NN, which contains similarities to a typical ANN as described in §2.5. However, rather than a conventional artificial neuron, now physical conductive substrates (but potentially any exploitable material or medium) are used as the HL neurons. Input voltage signals can be formulated from the data using a typical linear layer, similar to that introduced in §5.6. The output voltages from the physical

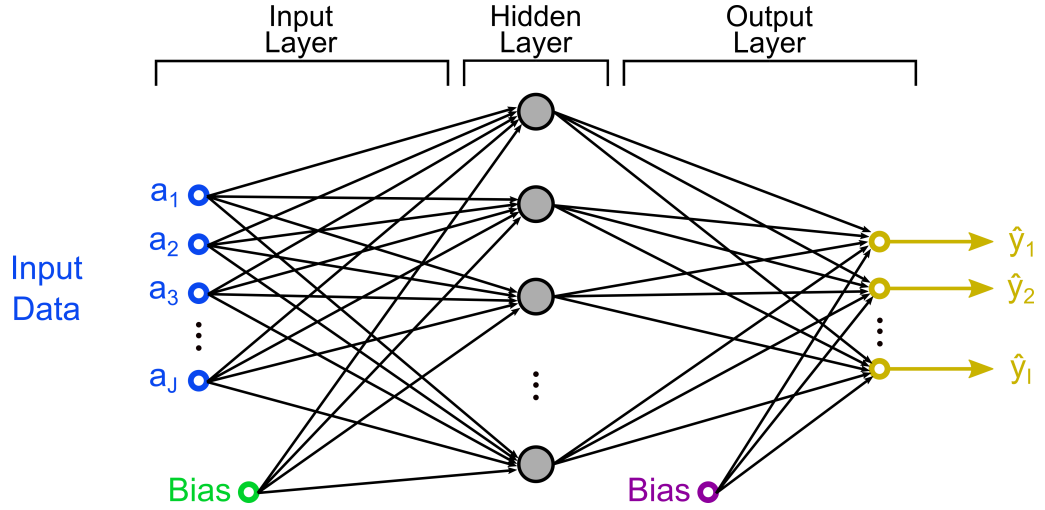


Figure 6.1: Basic structure of an artificial SLFN.

neurons are the HL's output states, used to predict or classify the data; it should be noted that an EiM material processor generally projects the applied input data voltages to a higher dimensional number of output voltages.

Therefore, several CAP based physical neurons are used within the HL, rather than the single substrate typically found in a monolithic EiM processor. Recall from §3.2 that a single CAP contains P -inputs (made from S stimuli and R data signals) and Q -outputs; in the case of a conductive nanomaterial, these would be allocated from available electrodes on a micro-electrode array used to apply and read voltages. A fully connected linear layer can be used to generate the 'data' input signals. For a particular data instance k defined by its input attributes $\mathbf{a}(k)$, the physical neuron HL's input voltages would be as follows:

$$V_{r,m}^{in}(k) = \mathbf{w}_{r,m}^{in} \cdot \mathbf{a}(k) + b_{r,m}^{in} , \quad (6.2.1)$$

where $V_{r,m}^{in}$ is the input to the r^{th} input 'data' node/electrode of the m^{th} physical HL neuron with corresponding weights $\mathbf{w}_{r,m}^{in}$ and bias $b_{r,m}^{in}$. Similarly to the fully connected layer considered in §5.6, the applied voltage must remain within the range of a system's real possible values, so a 'clipped' boundary condition is implemented where $V^{min} \leq V_r^{in}(k) \leq V^{max}$. Any remaining input nodes of the physical neuron are assigned as configuration voltage stimuli (V^c), altering how the neuron behaves, analogues to altering its activation function (as further discussed in §6.4.1).

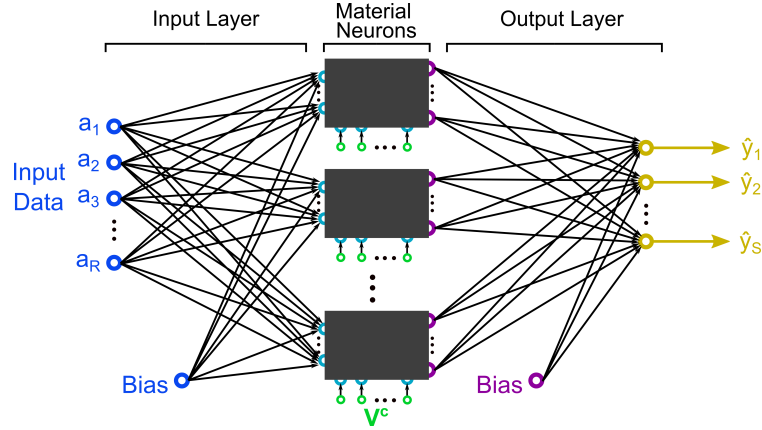


Figure 6.2: Structure of an iM-NN exploited for classification, using a fully connected linear input & output layer and configurable voltage stimuli V^c which alter a neuron's physical behaviour.

Once the input voltages are applied, output voltages are read and interpreted as the HL output states. The outputs of the m^{th} physical neuron can be considered to be the result of a black box transformation \mathcal{M}_m (described in §3.2):

$$\mathbf{V}_m^{out}(k) = \mathcal{M}_m(\mathbf{V}_m^{in}(k), \mathbf{V}_m^c), \quad (6.2.2)$$

where $\mathbf{V}_m^c = [V_{1,m}^c, V_{2,m}^c, \dots, V_{S,m}^c]$ is the vector of input configuration voltages, S is the number of voltage stimuli per material, and $\mathbf{V}_m^{out} = [V_{1,m}^{out}, V_{2,m}^{out}, \dots, V_{Q,m}^{out}]$ is the vector of output node voltages for the m^{th} physical HL neuron.

Having sampled the physical HL's output voltage states, they are combined using a fully connected linear output layer to make a prediction $\hat{\mathbf{y}}$ as follows:

$$\hat{\mathbf{y}}(k) = \sum_{m=1}^M (\mathbf{w}_m^{out} \cdot \mathbf{V}_m^{out}(k)) + \mathbf{b}^{out}, \quad (6.2.3)$$

where \mathbf{w}_m^{out} is the corresponding output layer weights for the m^{th} material neuron and \mathbf{b}^{out} are the bias'. The structure of the proposed iM-NN is presented in Fig. 6.2.

This system structure focused on using linear input and output layers because of their simplicity, but also to emphasise the performance of the physical neurons leveraged in this work. Additionally, there is good potential to realise and integrate these linear layers physically, such as using memristor based matrix multiplication

[8, 9, 10].

6.2.1 Directly Connected Input Layer

ANNs are considered by some to be over-parameterised [11]. The complexity engineering approach [12] advises that one should seek to limit the number of controllable parameters in an attempt to emphasise and harness emergent behaviour from a system's rich intrinsic properties. This suggests that efforts should be made to limit the number of optimisable system parameters, meaning the iM-NNs might benefit from concepts such as weight agnostic and minimal neural network topologies, which have been found to be beneficial in ANNs [13].

A sparsely connected network is implemented by removing random connections from a typical fully connected NN [14] (i.e., where weights and biases are randomly set to zero). This method can be used to reduce the number of optimisable system parameters, in an attempt to better exploit the NN structure itself [15]. Taking this concept to the extreme, to minimise the number of network connections, led to the development of a '*directly connected*' input layer. This is an iM-NN input layer where each of the HL's physical neuron voltage data input nodes/electrodes receives only one (weighted but unbiased) input attribute, implemented as follows:

$$V_{m,r}^{in} = w_{m,r}^{in} \times a_{C_{m,r}} , \quad (6.2.4)$$

where $C_{m,r} \in \{1, 2, \dots, J\}$ defines which attribute a_j is being passed to a particular physical neuron's data input node/electrode r , and $w_{m,r}^{in}$ is that connection's associated weight.

6.2.2 Virtual iM-NNs

It should be highlighted that there is the possibility of re-using a single nanomaterial substrate as several 'virtual' physical neurons [16]. In the past, EiM processors have been configured and re-configured to perform a wide variety of operations. In the case of the conductive networks exploited as EiM processors within this work, these different applications are achieved by training evolvable external stimuli and other

parameters. Therefore, a single material which can achieve significantly different behaviours (i.e., modes of operation), could be re-used as several distinct virtual physical neurons.

Conceptually, a virtual iM-NN uses the same structure as the normal iM-NN shown in Fig. 6.2. However, the physical neuron based HL is no longer implemented by several physical parallel material substrates. Instead, virtual neurons are implemented by re-using only a single substrate. These virtual neurons are grouped into the HL, and their outputs are determined similarly to Eq.6.2.2, but only a single physical neuron is utilised:

$$\mathbf{V}_m^{out}(k) = \mathcal{M}(\mathbf{V}_m^{in}(k), \mathbf{V}_m^c) , \quad (6.2.5)$$

where m is a virtual neuron in a HL containing M virtual neurons, and \mathcal{M} is the black box transformation defined by the single physical neuron which is being re-used. To compute the entire HL outputs, this single physical neuron must be used recursively, where each virtual neuron is evaluated sequentially.

Operating a single material processor as several virtual neurons provides significant EiM system design flexibility. However, these systems lose their ability to benefit from the iM-NN's highly parallelisable structure.

6.3 In-Materio Extreme Learning Machines

As discussed in §1.4, ELMs and Reservoir Computing (RC) present a good analogy for *in-materio* processors since both involve the exploitation of random networks. These systems depend on the underlying assumption that a randomised network/reservoir will produce useful and often higher dimensional output states that are used to process the data more successfully. Implementing physical realisations of such systems could lead to low power, efficient and fast systems [17] which can operate at ‘the edge’. RC was developed from Recurrent Neural Networks and is generally employed to process temporal data; whereas ELMs were developed from SLFNs and are generally employed to process non-temporal data [3]. There remains significant opportunity to develop classical or quantum substrates [18] for both RC

and ELM.

In this section, the proposed iM-NN structure is exploited with a directly connected input layer and trained as an ELM, described as iM-ELM. This illustrates a simple training method which can be used to exploit nanomaterial substrates as ‘physical’ or ‘*in-materio*’ neurons. To enable efficient investigation of this novel structure, DRNs were used as a proxy for physical nanomaterials (as described in §3.4), which were solved using fast, reliable Simulation Program with Integrated Circuit Emphasis (SPICE) simulations. These were leveraged to implement Simulated in-Materio (SiM) neurons. Several classification datasets were considered to investigate the performance of these iM-ELM for various HL sizes and physical conductive network topologies. Finally, drawing from the previous chapters which showed that EiM processors can be successfully re-configured, a material ‘re-use’ system was implemented, whereby a single SiM neuron was re-used to create several virtual physical HL neurons. Work considering the proposed iM-ELM was presented and published in a peer reviewed proceedings [5].

6.3.1 Experimental Implementation

Several DRN based SiM neurons were organised in a iM-NN like structure, as described in §6.2. However, in an attempt to comply with the complexity engineering approach [12], the *directly connected* input layer, outlined in §6.2.1, was used. These systems were exploited as ELMs, the method of which is described in the Theory §2.5.2. In short, the input layer and configurable parameters (i.e., w^{in} , C and V^c) were randomised and fixed. Training data was then processed and used to optimise an output layer with Ridge Regression, which was found to provide more stability than typical Moore-Penrose inverse [19]. The test data was then used to evaluate the system’s classification performance. The method of combining a physical neuron based SLFN and ELM training is referred to as an iM-ELM, as shown in Fig. 6.3.

To efficiently investigate the proposed iM-ELM structure, simulated DRN conductive networks were leveraged as these systems’ neurons. These SiM neurons acted as a proxy for a typical nano-material network, and allowed for fast experimentation of this new system structure. Recall from §3.4 that these networks contain:

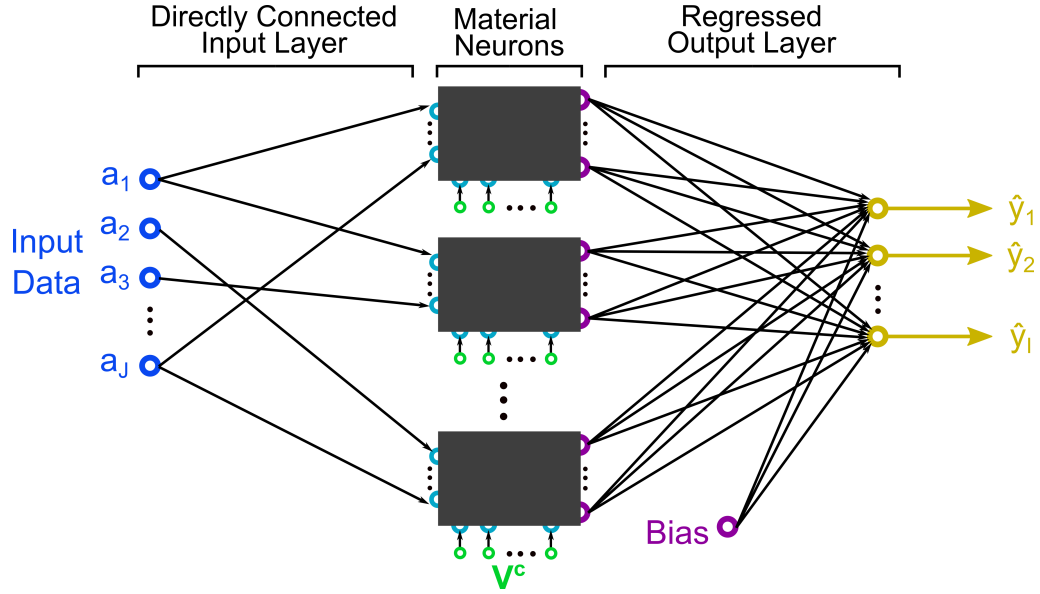


Figure 6.3: Basic structure of a physical neuron based SLFN exploited as an iM-ELM.

R voltage driven input data nodes, S voltage driven configuration stimuli, and Q measured output voltage nodes, calculated using a DC analysis. Therefore, the general size and topology of these simulated and randomly generated DRN based SiM neurons could be defined using $[P, S, Q]$. Several different SiM neurons could then be stacked in parallel to form the iM-ELM structure.

The system can be defined using a vector of decision variables \mathbf{X} , as discussed in §3.3. Expanding this to include all the iM-ELM adjustable parameters, the system's decision vector can be defined as:

$$\mathbf{X} = [w_{1,1}^{in}, \dots, w_{1,R}^{in}, C_{1,1}, \dots, C_{1,R}, \dots, V_{1,1}^c, \dots, V_{1,S}^c, w_{M,1}^{in}, \dots, w_{M,R}^{in}, C_{M,1}, \dots, C_{M,R}, V_{M,1}^c, \dots, V_{M,S}^c]^T, \quad (6.3.6)$$

which here is re-written as the system's parameter vectors stacked together²:

$$\mathbf{X} = [\mathbf{w}^{in}, \mathbf{C}, \mathbf{V}^c]^T, \quad (6.3.7)$$

where \mathbf{w}^{in} is a vector containing all the input layer's input weights for the M material neurons in the HL, \mathbf{C} are all the *directly connected* input layer's index

²The component vectors/matrices are flattened and stacked together to form a 1d array.

values $C_{m,r}$ used to select an attribute to be passed to a particular input data node r on a material m , and \mathbf{V}^c is a vector containing all the configuration voltages for the M materials. The maximum and minimum system voltages were selected as ± 10 V, and input weights were constrained between $w^{in} \in [-1, 1]$, and configuration voltages between $V^c \in [-10, 10]$.

Now, any single iM-NN or population \mathbf{p} of iM-NNs (i.e., multiple initialisations of \mathbf{X}) can be randomly generated and trained as an ELM network using the simple Pseudocode described in Algorithm 6.1.

Algorithm 6.1: Pseudocode for iM-ELM.

Initialise random population of solutions \mathbf{p} ;
Train \mathbf{p} by optimising the readout layer using the training data;
Evaluate population using the test data $\Phi(\mathbf{p})$;

Finally, it is highlighted that there is the possibility to re-use a single simulated material network as several ‘virtual’ neurons. By randomly initialising different configurable parameters, but using only a single DRN based SiM neuron, several virtual neurons are generated, as discussed in §6.2.2. Each of these will manifest their own unique internal Current-Voltage (IV) characteristics, which the ELM system will attempt to exploit. Such a network is referred to as a *Virtual* iM-ELM. Several datasets were considered, split 70% – 30% to create a training and test subset as described in §3.6.1. These were benchmarked against some common *sklean* classification techniques [20] using their default hyperparameters, with results shown in Table 6.1. It is noted that marginal gains in performance would likely be achieved with hyperparameter tuning.

Table 6.1: Test results for the datasets when using several common classification methods. Best accuracy highlighted in bold.

Dataset	Ridge Reg		Logistic Reg	SVM rbf	Random Forest
	Φ_{mse}	<i>Accuracy</i>	<i>Accuracy</i>	<i>Accuracy</i>	<i>Accuracy</i>
sp2DDS	0.1575	0.7367	0.7367	0.9733	0.9767
diabetes	0.1714	0.7532	0.7532	0.7403	0.7446
wine	0.0760	0.9259	1.0000	0.9815	1.0000
aca	0.1184	0.3659	0.8502	0.8213	0.8599
wdbc	0.0738	0.9357	0.9591	0.9532	0.9298

6.3.2 Performance

The performance of iM-ELMs of increasing HL sizes, used to classify several datasets, was considered using a Mean Squared Error (*mse*) objective function. Specifically, the number of DRN based SiM neurons in the HL was increased from one to fifteen (beyond which performance was generally considered to plateau). Twenty different random seeds were used to generate the SiM neurons within twenty different iM-ELMs systems. The same twenty seeds were used for each HL size incrementation, meaning that each iM-ELM system continued to include the same SiM neurons that were used in its corresponding previous smaller networks. Therefore, the change in performance of the iM-ELM networks could be considered as they were made bigger.

For each iM-ELM a ‘population’ of 100 uniform randomly generated decision vectors was considered (i.e., randomly initialised input layer and configuration parameters), which was observed to provide a good insight into performance and maintain reasonable simulation times. Recall from §3.4 & §6.3.1 that these DRN based SiM neurons’ consist of a fully interconnected conductive network containing three main classes of nodes: P input voltage nodes for data, S input voltage nodes for configuration/stimuli altering the SiM neurons’ behaviour, and Q output voltage nodes. Notably, the *directly connected* input layer used here connects each data input node to only a single data attribute; so, if too few neurons are in use, then not all data attributes may be ‘connected’. The experiment is performed with three increasingly larger SiM neuron topologies (denoted using $[P, S, Q]$): (i) [2,2,4], (ii) [3,3,5], and (iii) [4,4,6]. This provides some initial insight on the effect of scaling the size of non-linear conductive network based neurons.

The performance of these iM-ELMs is compared against the mean test fitness of 2000 artificial (Moore-Penrose) ELMs and (Ridge Regression) RR- ELMs; the number of which was selected to match the total computational expense (i.e., total number of data instances) used over the 20 iM-ELMs systems. These ANN based ELMs used the sigmoid activation function for their artificial HL neurons. Whilst many activation functions exist [21], the sigmoid function is widely used [22] and can achieve good performance in most cases [23].

All these systems were used to classify the sp2DDS, diabetes, wine, aca and

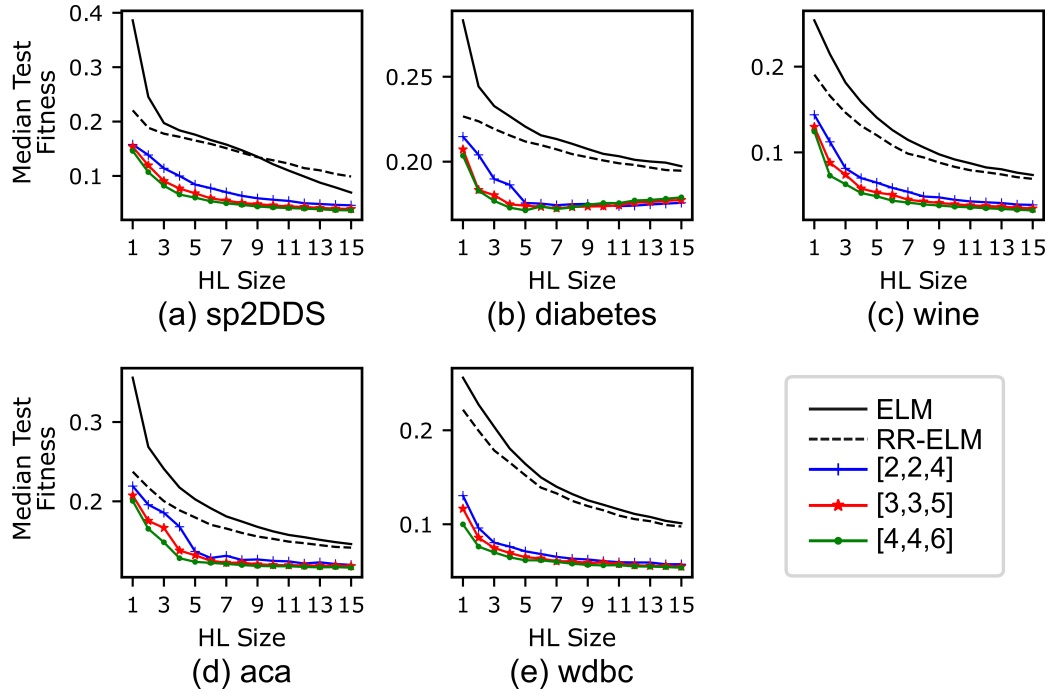


Figure 6.4: Median mse test fitness of all the (20 systems, each with 100 parameter initialisations) iM-ELMs for each HL size increment, used to classify the (a) sp2DDS, (b) diabetes, (c) wine, (d) aca and (e) wdbc datasets. Three different material neuron topologies are considered ($[P, S, Q]$), and these are compared to the mean accuracy of 2000 traditional artificial ELMs and RR-ELMs.

wdbc datasets (details given in §3.6.1) which have been benchmarked using several common classification methods as seen in Table 6.1. All the datasets were normalised and then scaled to the maximum and minimum system voltages (i.e., ± 10 V). Results comparing the different material topology based iM-ELMs and traditional ELMs are given in Fig. 6.4. These show the median test fitness (Φ_{mse}) as the HL increments in size. Generally, the iM-ELMs outperformed the artificial ELM and RR-ELM systems of equivalent network sizes. The simulated DRN based neurons successfully generated useful, higher dimensional output states which were exploited as an ELM, and these out-perform their artificial neuron counterparts. As more SiM neurons were operated in parallel, the median fitness improved. Notably, the larger and more complex SiM neuron topologies (i.e., when using larger DRN with more input, configuration, and output nodes/electrodes) achieved lower median fitnesses for iM-ELMs with the same size of HL. Thus fewer neurons were required within the SLFN HL to achieve comparable results with networks leveraging ‘less capable’, smaller SiM neurons.

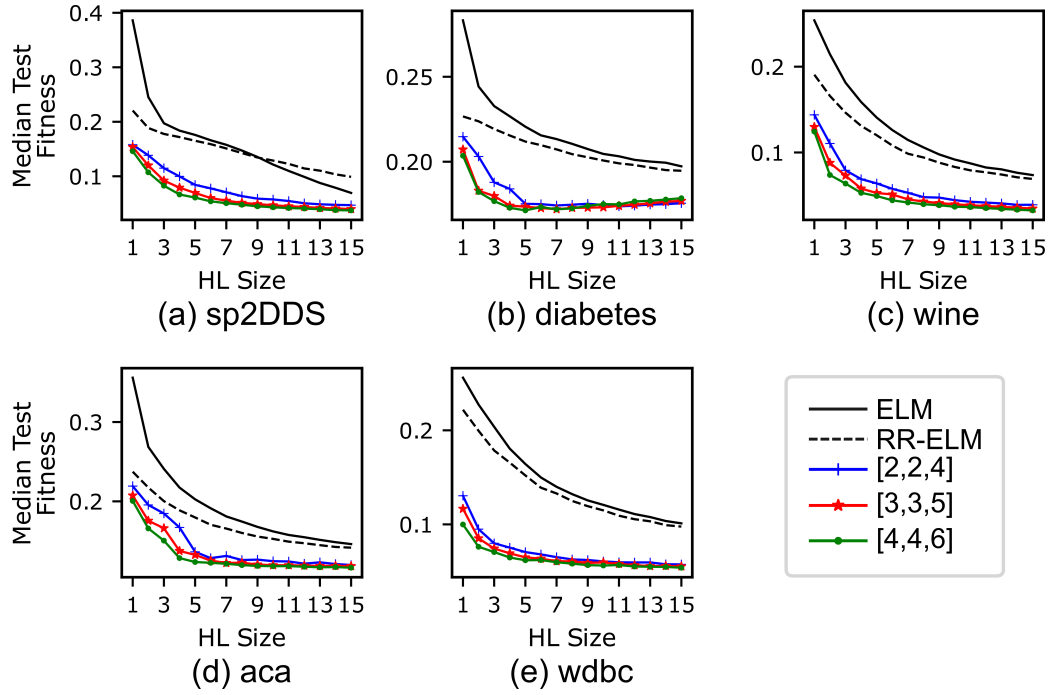


Figure 6.5: Median mse test fitness of all the (20 systems, each with 100 parameter initialisations) *Virtual* iM-ELMs for each HL size increment, used to classify the (a) sp2DDS, (b) diabetes, (c) wine, (d) aca and (e) wdbc datasets. Three different material neuron topologies are considered ($[P, S, Q]$), and these are compared to the mean accuracy of 2000 traditional artificial ELMs and RR-ELMs.

The best accuracy achieved, across all HL sizes, for the different material neuron topologies and datasets, is shown in Table 6.2. The iM-ELMs considered could significantly outperform some of the common classification methods presented in Table 6.1. Indeed, the best iM-ELMs also compare favourably with the traditional artificial ELM networks, generally outperforming or matching the best obtained accuracy.

Table 6.2: Best accuracy achieved from the different systems, from across the different HL sizes and neuron topologies ($[P, S, Q]$). The best accuracy for each dataset is highlighted in bold.

Dataset	iM-ELM			<i>Virtual</i> iM-ELM			ELM	RR-ELM
	[2,2,4]	[3,3,5]	[4,4,6]	[2,2,4]	[3,3,5]	[4,4,6]		
sp2DDS	0.9933	0.9967	0.9933	0.9933	0.9967	0.9933	0.9900	0.9667
diabetes	0.7922	0.7965	0.8009	0.7922	0.7922	0.7965	0.7922	0.7879
wine	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
aca	0.8744	0.8792	0.8792	0.8744	0.8841	0.8841	0.8841	0.8841
wdbc	0.9708	0.9766	0.9766	0.9708	0.9708	0.9766	0.9766	0.9708

As discussed in §6.2.2 and §6.3.1, any single material could be re-used as several virtual neurons. The different randomly initialised parameters (input weights, connections and configuration voltages) should enable each virtual neuron to behave in a sufficiently independent manner. To investigate this, the previous analysis is repeated, i.e., generating twenty SLFN for each HL size, each with 100 random initialisations. However, now each iM-ELM's HL contains virtual neurons which were formulated from only a single DRN based SiM neuron, as detailed in §6.2.2. The median test fitness of these *Virtual* iM-ELMs is plotted against the size of the SLFN in Fig. 6.5, and the best ever achieved accuracies are shown in Table 6.2. The *Virtual* iM-ELMs attained a very similar performance to the previously discussed iM-ELMs systems, which exploited several different DRN based SiM neurons. This suggests that a single DRN based SiM neuron could successfully produce several virtual instances, achieved by exploiting the wide range of non-linear IV characteristics which can be tuned and selected by the voltage stimuli and input layer respectively. These *Virtual* iM-ELMs allow for significantly more flexibility when implementing experimental systems, only requiring a single material substrate to create SLFNs containing several HL neurons. However, by ‘re-using’ only a single material substrate (rather than operating several material substrates simultaneously in parallel), a *Virtual* iM-NN loses its ability to benefit from its otherwise highly parallelisable structure.

6.4 Neuroevolution of Physical iM-NNs

Work in the previous section showed how iM-NNs trained as ELMs performed well in comparison to conventional artificial ELMs of a similar size – showing that more ‘computation’ can be offloaded to and achieved within a multi-input, multi-output conductive network based physical neuron, compared to a conventional artificial neuron. However, a comparison of more typical iterative based algorithms used to train such novel iM-NNs remains lacking. It is highlighted that any meaningful comparison must be made with respect to (*w.r.t*) a fixed computational ‘budget’, since the application of data to a processing material (or medium) and reading of

its output states will likely be a bottleneck in any physical system.

While EAs are powerful and flexible optimisers, back propagation and gradient descent are the cornerstone techniques used in modern ML. However, physical material neurons are often considered to be black boxes and non-differentiable. The desire to implement gradient descent and improve *in-materio* devices has manifested in two distinct ways. Firstly, physical *in-materio* devices which have been modelled to enable derivatives to be calculated. Either analytical physical models [24] or trained models (such as a Deep Neural Network [25]) can be used. In this case, training can either occur fully *in-simulo* (i.e., only using the model) or as a hybrid ‘physics aware training’ [24] where the forward pass is executed *in-materio* and the backward pass is computed using the constructed model. However, the success of training depends on the accuracy and reliability of the model, which can be difficult to achieve, especially in high dimensional systems. Secondly, physical perturbations can be introduced to calculate local gradients and therefore enable gradient descent [26, 27]. Recent work used orthogonal sinusoids for each input of an *in-materio* device (single perceptron like structure) to compute gradients in parallel, similar to homodyne detection [28]. However, training speed is limited by the lowest frequency of the set of orthogonal sinusoids, and this method assumes small perturbations are only linearly transformed, but they are in fact likely to be affected by a noisy environment and be sensitive to strong non-linearities or temporal properties present in some nanomaterials, such as charge trapping. It is hypothesised that effective training might be achieved by combining the most successful elements of EAs and gradient descent methods.

Unlike the iM-ELM work carried out in the previous section (which used SiM neurons), the iM-NNs constructed here used Physical in-Materio (PiM) neurons, and results were collected using lab-based experiments. Specifically, these physical iM-NNs were implemented using conductive Lambda Diode (LD) based PiM neurons, and experimentation was performed with a Raspberry Pi and custom Hardware Interface (HI).

Three EAs were considered to train this novel physical neural network architecture, which is described as the neuroevolution of the iM-NNs. These algorithms

include Differential Evolution (DE), which has been successfully used to exploit more traditional *in-materio* devices [29, 30, 31]. OpenAI Evolutionary Strategy (OpenAI-ES) [32] which is a type of Natural Evolution Strategy (NES) [33] that uses a pseudo-population to predict the natural gradient and perform gradient descent. And finally, the popular Covariance Matrix Adaptation Evolution Strategy (CMA-ES) [34]. The iM-NNs successfully classified several ML classification problems, verifying the feasibility of the proposed physical neuron based system. The ability to batch (or “mini-batch”) is highlighted, and was found to allow for more efficient budget usage and speed up the EAs’ convergence. Work presented here has been submitted to a journal for publication [6], titled *Training in-Materio Neural Networks*.

6.4.1 Physical LDN Neuron

A LDN was used to produce a PiM neuron with complex, non-monotonic IV characteristics. These were constructed in a lab where voltages were applied and read using a custom Printed Circuit Board (PCB) (i.e., our ‘Hardware Interface’) as described in §3.5.2. A Raspberry Pi was used to host the running EAs and operate the experiments.

The LDN was arbitrarily constructed, as shown in Fig. 6.6a, and discussed in §3.5.2. The network contained six nodes: three inputs and three outputs. Of the inputs, two were assigned as ‘data inputs’ (V^{in}) and one as a ‘configurable stimuli’ (V^c). Selecting a small network allowed for better visualisation and interpretation of the PiM neuron’s behaviour. This behaviour is examined in Fig. 6.6b, which shows the voltages on each output for a two-dimensional, 0.5 V interval, sweep on the data inputs, for a select few values of configuration voltages. This provides an insight into the material’s ability to project inputs to higher dimensional outputs, or in other words, the PiM neuron’s function \mathcal{M} defined in Eq.(6.2.5). The behaviour of the PiM neuron is directly related to the configuration of the LDN. Here, complex characteristics and enclosed boundaries are observed. This is similar to the properties found in the dopant-atom networks [35], whose non-linear and non-monotonically increasing hopping conduction characteristics were exploited for

classification. However, Fig. 6.6b also emphasises the usefulness of a physical neuron with multiple outputs. If one output is not suited to the particular task at hand, then the characteristics of a different output, or a combination of outputs, might be. It is speculated that this could be important in mitigating poor performance of physical neurons manufactured using and exploiting randomly interconnecting nanomaterials (e.g., carbon nano-tubes [30]) where one or more output nodes/electrodes might happen to perform poorly or be weakly connected. Indeed, using a combination of EiM processor output states was found to increase performance and flexibility in chapter 4.

Each data point in Fig. 6.6b consists of the average of 30 samples. The accumulated residuals from the whole sweep can be seen as a histogram in the Appendix B.3. The standard deviation of the residuals of all output values from across this sweep are as follows: $std(V_1^{out}) = 3.16 \text{ mV}$, $std(V_2^{out}) = 6.91 \text{ mV}$, and $std(V_3^{out}) = 3.05 \text{ mV}$.

6.4.2 Experimental Implementation

An iM-NN was physically created as shown in Fig. 6.2 using LDN based PiM neurons and the experimental HI discussed in §3.5. Due to hardware limitations,

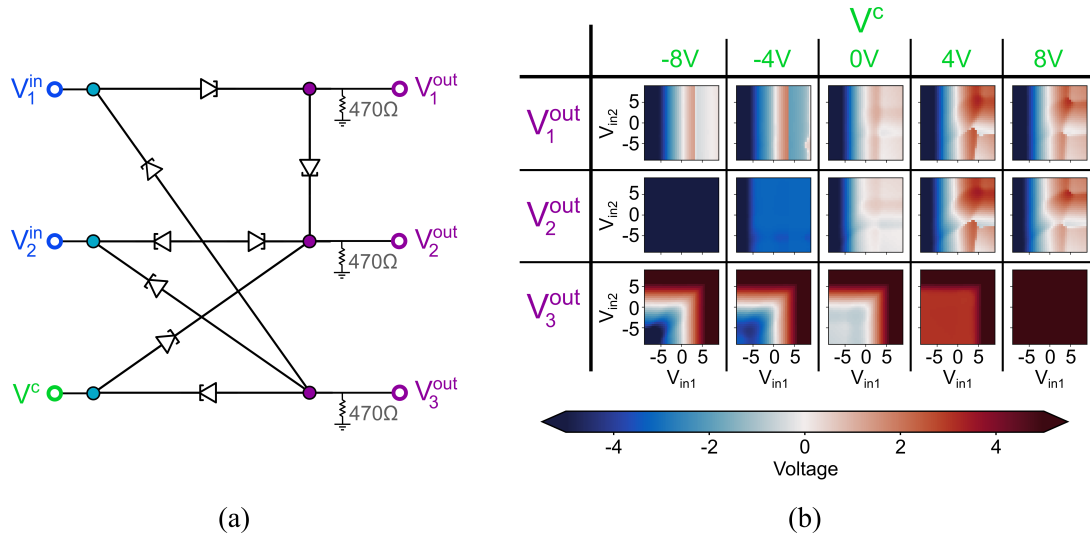


Figure 6.6: The (a) Lambda Diode Network (LDN) used in experimentation and leveraged as a Physical in-Materio (PiM) neuron, and (b) the surface plot of the LDN's physical outputs for a 2D sweep of V_1^{in} & V_2^{in} , and a selection of configuration voltages V^c .

Table 6.3: Test results for the datasets when using several common classification methods. The best accuracy for each dataset is highlighted in bold.

Dataset	Linear Reg	Logistic Reg		SVM rbf	Random Forest
	<i>Accuracy</i>	Φ_{ce}	<i>Accuracy</i>	<i>Accuracy</i>	<i>Accuracy</i>
sp2DDS	0.7367	0.4811	0.7367	0.9733	0.9767
banknote	0.9806	0.0800	0.9806	1.0000	0.9976
iris	0.8000	0.2651	0.9778	0.9778	0.9778
raisin	0.8519	0.3659	0.8481	0.8519	0.8556
wine	0.9630	0.1412	1.0000	0.9815	1.0000

only a single PiM neuron was utilised in this work. Therefore, to construct larger iM-NNs ‘virtual’ neurons were used, as outlined in §6.2.2, where a single substrate is re-used as several virtual neurons. To aid convergence, fully connected layers were used to help formulate a smooth fitness surface or ‘solution space’ (unlike the sparse directly connected input layer used by the iM-ELMs). The Cross Entropy (CE) objective function was selected, as described in §2.4.4, to calculate the fitness of a particular solution / system configuration for multi-class classification.

An alternative way to perceive these PiM neurons is as introducing some relational inductive bias [36], where the input layer produces an embedding space for each PiM neuron. These individual configurable analogue processing units [37] are then leveraged to produce useful higher dimension projections of the input space.

Several datasets were considered, split 70% – 30% to create a training and test subset as described in §3.6.1, and benchmarked against some common classification techniques as shown in Table 6.3. To train the iM-NNs to classify the datasets, three EAs (each detailed in §2.3) were considered: DE, OpenAI-ES, and CMA-ES. In the following, the properties and hyperparameters used for these algorithms is outlined. These values were selected using a grid search of different hyperparameters when the algorithms were attempting to classify the hm2DDS, and these results can be found in the Appendix D. Unless otherwise stated, the algorithms operated under the following conditions:

DE: A DE/best/1/bin algorithm is used with a reflection bounds constraint as described in §2.3.1. Hyperparameters include F , CR and λ . A 2D surface sweep of mutation factor & crossover rate was performed, and $F = 0.5$ &

$CR = 0.6$ were found to yield good performance.

OpenAI-ES: The OpenAI-ES is used with Adaptive Moment Estimation (Adam) optimiser and a ‘clip’ boundary constraint as described in §2.3.2 and §2.5.2. Hyperparameters include σ , α and λ , but also the additional hyperparameters introduced by Adam optimiser: $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\varepsilon = 10^{-8}$. A 2D surface sweep of the Gaussian noise & learning rate was performed, and $\sigma = 0.001$ & $\alpha = 0.01$ were found to yield good performance.

CMA-ES: As discussed in §2.3.3, hyperparameters include σ and λ ; however, as commonly done, population size was automatically assigned as $\lambda = 4 + 3 \log(d)$. A sweep of the initial step size (i.e., initial standard deviation of noise) was performed, and the results show that $\sigma = 0.15$ achieves good performance (*w.r.t* normalised parameter boundaries $\in [0, 1]$).

In summary, recalling the structure of a single HL iM-NN as shown in Fig. 6.2, the EAs were attempting to optimise a vector of the system’s parameters which contains all the parameter vectors stacked together ³:

$$\mathbf{X} = [\mathbf{w}^{in}, \mathbf{b}^{in}, \mathbf{V}^c, \mathbf{w}^{out}, \mathbf{b}^{out}]^T. \quad (6.4.8)$$

Since physical real-valued system parameters are being optimised, real-world constraints were required. The maximum and minimum allowed voltages were selected as ± 9.5 V, this applied to both the linear-layer generated input ‘data’ voltages and the configuration voltage stimuli. However, DE also required boundaries for all its decision variables \mathbf{X} . These boundaries were similarly applied to the OpenAI-ES and CMA-ES trained systems, which were as follows: input layer weights $w^{in} \in [-1, 1]$, output layer weights $w^{out} \in [-2, 2]$, and biases $b \in [-2, 2]$.

To ensure a fair comparison between EAs, the neuroevolution was executed for a fixed budget, rather than a selected number of epochs. Similarly to §5.3.2, the budget was defined by the ‘number of computations’ (N_{comps}), which was the number of times data inputs were executed on the physical HL neurons. For a fully realised

³The component vectors/matrices are flattened and stacked together to form a 1d array.

iM-NN, all the PiM neurons, and therefore an input data instance, can be operated in parallel. However, for the virtual iM-NN used here, each virtual HL neuron must be computed sequentially. In this case, a single input data instance results in several computations, where a single ‘computation’ involves the application of data to a virtual PiM neuron, i.e., three voltage sets to, and three voltage reads from the LDN.

6.4.3 Performance

Population and Batch Size

In this work, the classification capabilities of the iM-NNs are highlighted, when optimised using different EAs, with a fixed computational budget. Hyperparameters such as the population size, or varying the batch size, will affect the amount of budget used each epoch, and alter the ‘length of convergence’ or the total number of epochs executed. Therefore, it should be possible to tune such hyperparameters and ensure an efficient use of an available budget.

Indeed, it has been shown that batching can speed up the DE algorithm by enabling more useful population updates per amount of data, as described in §5.3. This work considers the effect of batch size (bs) and population size (λ) when solving the sp2DDS dataset using an iM-NN with three PiM neurons in the HL. Each algorithm was used to solve the dataset with a budget of $0.8 \times 10^6 N_{comps}$. This was repeated three times, using the same three random seeds, for each hyperparameter combination, and the final mean test loss values were used to produce the surface plots displayed in Fig. 6.7. It can be seen that CMA-ES was more capable of achieving better fitnesses (i.e., losses) when optimising the same iM-NN within the set budget.

Most significant was the result that altering the batch and population size could lead to significantly better performance during the allotted limited computational budget. A smaller population size reduces the number of computations per generation, and a smaller batch size increases the number of generational updates per epoch, both leading to a more efficient use of the budget during each epoch. However,

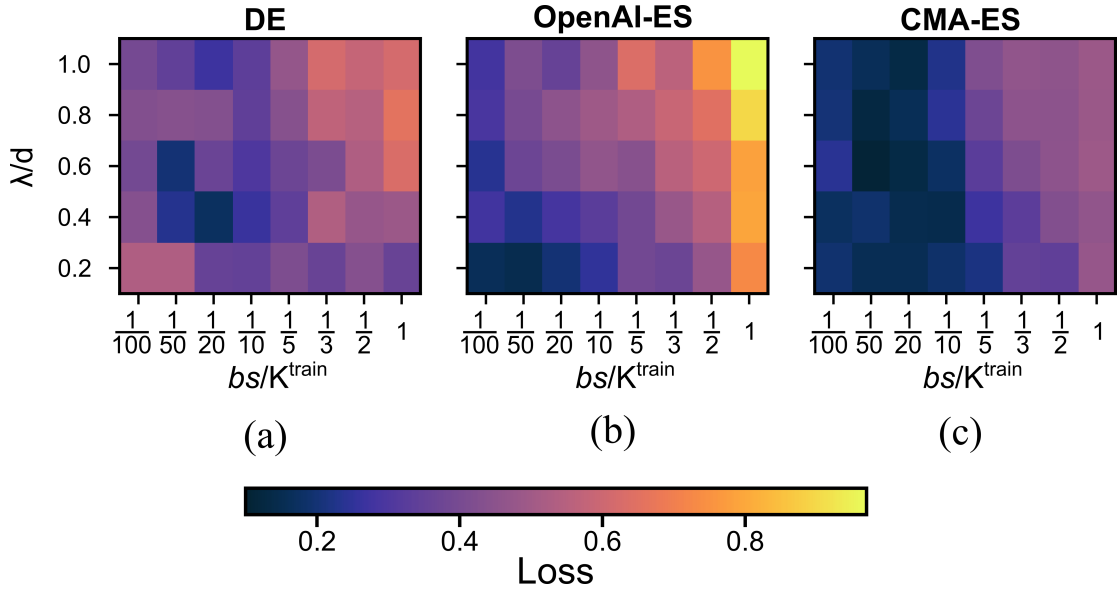


Figure 6.7: Surface plots of the (a) DE, (b) OpenAI-ES, and (c) CMA-ES algorithms’ final mean test CE loss classifying the sp2DDS after a fixed budget of 800000 N_{comps} , for a sweep of different population sizes λ (*w.r.t* the iM-NN’s number of parameters d) and batch sizes bs (*w.r.t* the total number of training instances K^{train}), using a three HL neuron iM-NN.

these techniques are limited, since using too small a population size or insufficiently large batches could result in poor convergence, where each generation holds too little information to make ‘useful’ updates.

For DE, the literature suggests larger population sizes, between $3d$ and $8d$, are a good plausible first choice [38]. However, from this hyperparameter investigation, it is found that DE can perform well with smaller population sizes *w.r.t* the number of evolvable dimensions d and the fixed computational budget. It is observed that a trade-off exists where significantly smaller population sizes led to worse final test fitnesses due to poor convergence, but larger population sizes also led to poor final test fitnesses due to slow convergence since each generation consumes a large fraction of the available budget. Although it was shown that a good combination of population and batch size does exist for maximum exploitation of the available data, it might require an involved investigation to determine these values.

The OpenAI-ES and CMA-ES algorithms appear to be much more resilient to small population and batch sizes. Here, the OpenAI-ES utilised the Adam optimiser which maintains a first and second moment ‘moving average’ which helps mitigate

adverse effects from abrupt loss changes when using small batch sizes. Similarly, CMA-ES learns a second order model of the objective function, which also makes it resilient to small batch sizes. CMA-ES is well-known for its ability to use small population sizes.

These results suggest that while DE can perform well, it remains sensitive to hyperparameter tuning and likely benefits from larger computational budgets. When applied appropriately, OpenAI-ES can achieve competitive results, but required smaller batch sizes to make efficient use of an allotted budget. Meanwhile, the CMA-ES uses the fewest number of hyperparameters and yet remained robust when using small batch sizes.

Algorithm Performance

The three algorithms were used to optimise iM-NNs to classify the sp2DDS, banknote, iris, raisin & wine UCI datasets, described in §3.6.1. Each algorithm was repeated six times, using the same six random seeds, on each dataset. Details of the iM-NN structure and algorithm configurations are given in Table 6.4 along with final test results. Additionally, the mean test fitness (i.e., CE loss) convergence for the various datasets are shown in Fig. 6.8.

Clearly, the iM-NNs were successfully exploited by the optimising algorithms to solve the classification problems. While these datasets are relatively small, this allowed smaller iM-NNs to be used, helpful for faster investigation of these physical systems, and also enabled the original CMA-ES algorithm (without alterations required for larger dimensional problems) to be utilised. Indeed, good performance was achieved using these smaller networks and low budgets, highlighting the proof of concept for these physical *in-materio* neurons.

For all the datasets, CMA-ES was the algorithm which both on average converged fastest and most consistently, achieving the best final fitness values in all but one case. The CMA-ES trained iM-NNs also outperformed Logistic Regression on all the datasets. In particular, both a lower CE loss (than Logistic Regression) and a higher accuracy (than all the reference methods in Table 6.3) was achieved for the raisin dataset. The CMA-ES benefits from a resilience to rugged search landscape,

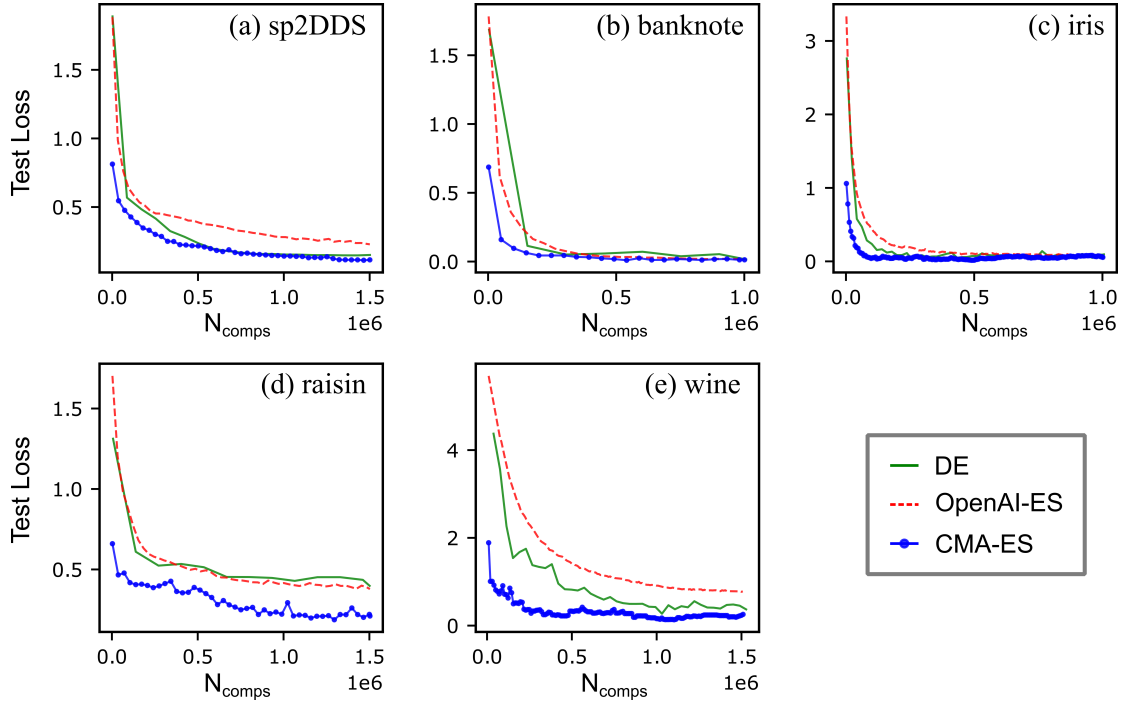


Figure 6.8: Comparison of the algorithms' mean test CE loss convergence, under a fixed budget, for the (a) sp2DDS, (b) banknote, (c) iris, (d) raisin, and (e) wine datasets.

step-size control which helps prevents premature convergence, and the ability to use a small population to efficiently utilise the allotted computational budget. The lack of extended hyperparameter tuning also makes it an attractive option.

DE did manage to perform well in several of the datasets, but its need to use larger population sizes (*w.r.t* the number of dimensions within the system) slows convergence by utilising a large proportion of the budget each epoch, leading to smaller numbers of total epochs. However, since DE is a more traditional population based EA, it was more tolerant (and therefore less likely) to getting stuck in local minima, similarly to CMA-ES.

The OpenAI-ES achieved much smoother convergence than either DE or CMA-ES. However, it appeared to suffer from slower convergence. The benefit of OpenAI-ES is that it performs typical gradient descent, albeit with an estimated natural gradient update. Therefore, common ML methods, such as Adam, could be used. It is hypothesised that implementing other such modern ANN training methods would likely boost performance.

6.4.4 Further Discussion

The test system was built using low cost Integrated Circuit (IC) components and a custom PCB; its implementation using python could achieve a very modest $\sim 2000N_{comps}/s$ (recall that here a single ‘computation’ consists of three voltage sets and three voltage reads), but allowed for significant flexibility during the design process. A more dedicated system focused on high performance would likely achieve a very significant increase in operational speed by reducing the physical signal set/read cycle bottleneck. In some systems, it is not unreasonable to expect execution speeds within *in-materio* processors could improve to 100 MHz [35] or more.

This work utilised a LD conductive network as a proxy for a *static* material i.e., a nanomaterial substrate which has fixed IV characteristics. Such devices could operate at high speeds and be trained for several tasks, and switch between them by simply recalling the appropriate configuration parameters. However, *dynamic* materials, which have varying IV characteristics, could also be used. For example, a Liquid Crystal (LC)/Single Walled Carbon Nanotube (SWCNT) mixture can be evolved to alter internal connections between electrodes to perform classifica-

Table 6.4: Final test results for the iM-NN’s neuroevolution, where $\bar{\Phi}$ is the mean final fitness, \bar{A} is the mean final accuracy and A^* is the accuracy of the run which achieved the smallest test fitness Φ^* . Best achieved loss for each dataset is highlighted in bold.

Dataset	Budget ($10^6 N_{comps}$)	N° HL Neurons	Batch Size (bs)	Algorithm	Pop Size (λ)	Performance				
						$\bar{\Phi}$	$std(\Phi)$	Φ^*	\bar{A}	A^*
sp2DDS	1.5	3	$\frac{1}{20}K^{train}$	DE	$0.5d \sim 20$	0.1508	0.0356	0.1064	0.9533	0.9800
				OpenAI-ES	15	0.2281	0.0852	0.1291	0.8983	0.9467
				CMA-ES	15	0.1151	0.0329	0.0782	0.9550	0.9700
banknote	1	3	$\frac{1}{20}K^{train}$	DE	$0.5d \sim 26$	0.0151	0.0095	0.0040	0.9931	1.0000
				OpenAI-ES	15	0.0198	0.0118	0.0027	0.9956	1.0000
				CMA-ES	15	0.0128	0.0143	0.0001	0.9972	1.0000
iris	1	3	$\frac{1}{3}K^{train}$	DE	$0.5d \sim 31$	0.0956	0.0558	0.0004	0.9778	1.0000
				OpenAI-ES	16	0.0810	0.0969	0.0178	0.9852	1.0000
				CMA-ES	16	0.0545	0.0367	0.0058	0.9815	1.0000
raisin	1.5	3	$\frac{1}{10}K^{train}$	DE	$0.5d \sim 35$	0.3918	0.0332	0.3588	0.8302	0.8481
				OpenAI-ES	16	0.3781	0.0445	0.3266	0.8500	0.8926
				CMA-ES	16	0.2106	0.1128	0.0811	0.9340	0.9815
wine	1.5	4	K^{train}	DE	$0.5d \sim 77$	0.3693	0.1049	0.2622	0.9290	0.9074
				OpenAI-ES	19	0.7792	0.3841	0.2827	0.8827	0.9444
				CMA-ES	19	0.2556	0.1148	0.0745	0.9660	0.9815

tion [30, 29]. But such *dynamic* materials might be better suited to reinforcement learning or processing temporal data. Indeed, by extending the EiM paradigm to process temporal data, this would in effect create an evolvable physical RC [17]. Significantly, the RC paradigm often limits itself to a fixed reservoir, but when operating in the physical domain it is hypothesised that some reservoir tuning might lead to better performance and increased flexibility.

Finally, it is noted that this work used virtual iM-NNs, where a single LDN based PiM neuron was re-used as each HL neuron. This means each data instance needed to be executed sequentially on each ‘virtual’ HL neuron, increasing the proportion of computational budget and time spent each epoch. For immediate gains in execution speed, each HL neuron should be physically realised and executed in parallel. However, as noted in §6.2.2, this trades off flexibility of physical implementation with parallelism in the realisation of these physical NN designs.

6.5 In-Materio AutoEncoders

So far, this chapter has developed methods of creating physical realisations of NN and examined different training methods. This allowed the creation of networks of physical neurons which could process larger, more complex datasets with more attributes, scaling in a way that a single material processor might not. Having established the use of iM-NNs, new types of NN structures can be investigated beyond typical SLFN classifiers.

In this section, an AutoEncoder (AE) structure is considered and used to perform dimensionality reduction. Unlike a classifier, an AE simply seeks to recreate the inputs at the outputs, and are therefore trained in an unsupervised manner. If a constriction is introduced into a NN, then a new lower dimensional set of features is produced for the data (i.e., encoding) known as the latent space. If the original inputs can be reconstructed using these features (i.e., decoding) then the AE has discovered a new ‘compressed’ form of the data. A five layer artificial AE structure is shown in Fig. 6.9, showing how the AE is split into an Encoder (which generates the features) and Decoder (which reconstructs the original data). Further details on

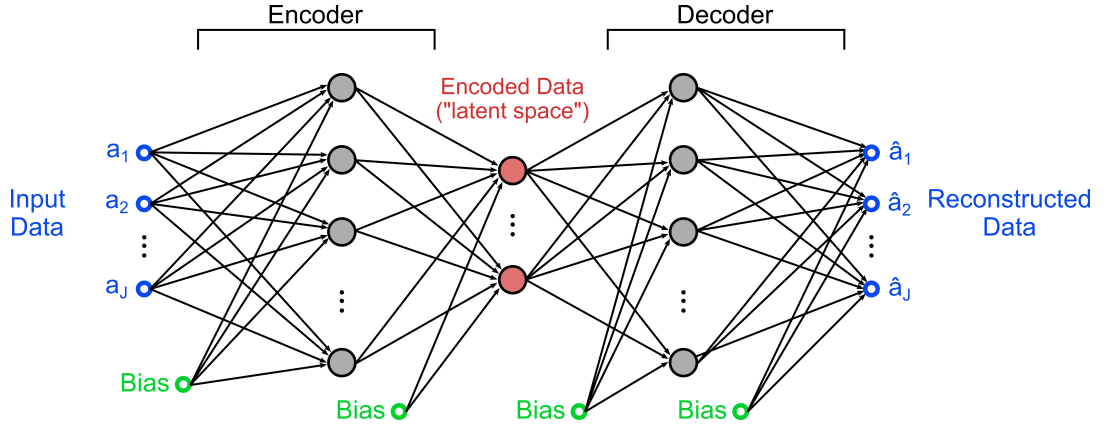


Figure 6.9: Structure of a five layer ANN based AutoEncoder.

the theory behind AEs can be found in §2.6.

6.5.1 Experimental Implementation

A novel in-Materio AutoEncoder (iM-AE) structure was proposed which leveraged physical neurons, as shown in Fig 6.10. Notice that a five layer AE was used, rather than the simplest three layer AE introduced in §2.6. The five layers used in the iM-AE are as follows:

- (i) Artificial input layer,
- (ii) Physical neuron based “encoder layer”,
- (iii) Central artificial neuron based HL used to produce the “latent space”,
- (iv) Physical neuron based “decoder layer”,
- (v) Artificial output layer.

Previous work in Chapters 4 & 5 have shown that by combining a material processor’s output states, more complex classification boundaries can be achieved. Therefore, the central artificial HL and artificial output layer are leveraged as combinatorial layers, encoding and reconstructing data respectively. These two artificial layers use linear activations, ensuring the performance of the proposed iM-AEs are determined by the non-linear capabilities of the physical neuron HLs. The five layer AE structure outlined is symmetrical, and therefore could be used to produce stacked AEs [39] and possibly train deep iM-NNs.

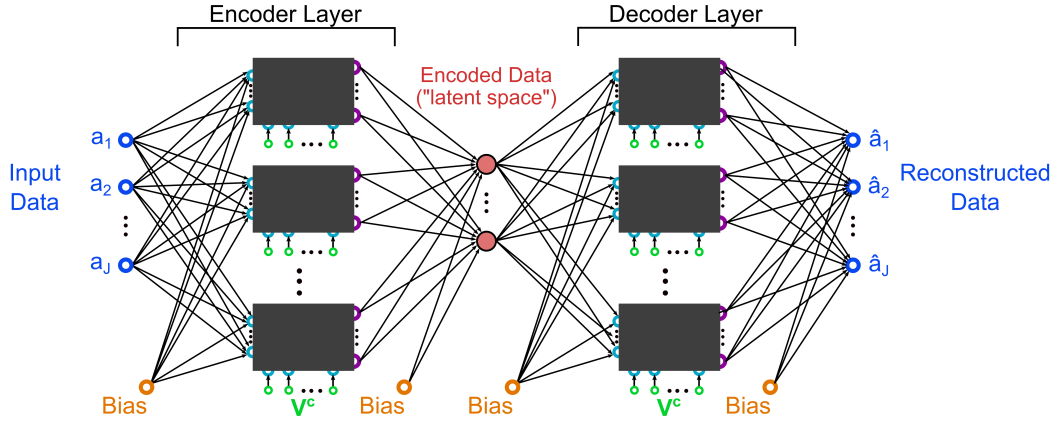


Figure 6.10: Structure of the implemented in-Materio AutoEncoder (iM-AE) using two physical neuron based Hidden Layers in a five-layer network.

To train these iM-AEs, an EA was combined with a ridge regression optimised output layer. This is similar to optimising an ELM using an EA [7, 40], and helps ensure that the outputs are successfully reconstructed from the decoder physical neuron HL. Indeed, by training the output layer separately, boundary values are not required for its weights & biases and might allow for finer tuning, as discussed in §5.5. In summary, an EA attempts to optimise a vector of the system's parameters which contains all the parameter vectors stacked together:

$$\mathbf{X} = [\mathbf{w}^{in,en}, \mathbf{b}^{in,en}, \mathbf{V}^{c,en}, \mathbf{w}^{out,en}, \mathbf{b}^{out,en}, \mathbf{w}^{in,de}, \mathbf{b}^{in,de}, \mathbf{V}^{c,de}]^T, \quad (6.5.9)$$

where $\mathbf{w}^{in,en}, \mathbf{b}^{in,en}, \mathbf{V}^{c,en}$ are the physical encoder layer's input weights, biases and configuration voltages, $\mathbf{w}^{out,en}, \mathbf{b}^{out,en}$ are the physical encoder layer's output weights and voltages, $\mathbf{w}^{in,de}, \mathbf{b}^{in,de}, \mathbf{V}^{c,de}$ are the physical decoder layer's input weights, biases and configuration voltages. The physical decoder layer's output weights and biases are independently optimised by ridge regression.

To illustrate the implementation of the proposed iM-AE structure, an example system was constructed leveraging the LDN as several virtual PiM neurons as discussed in §6.4.1, using the Raspberry Pi and HI detailed in §3.5.

This iM-AE was then used to encode a two-dimension representation of the digits dataset (§3.6.2) by performing dimensionality reduction, using the OpenAI-ES (with $\alpha = 0.015$, $\sigma = 0.001$, and $\lambda = 20 \approx \frac{d}{100}$) to optimise the system's evolvable

parameters using a budget of 4×10^7 N_{comps} . The single LDN was used to generate six virtual PiM neurons, and these same six virtual neurons were re-used in the physical encoder and decoder HLs.

Finally, an objective function was required to compute some error related loss to judge the performance of the AE's reconstruction. Unfortunately, the commonly used metrics such as the mse loss are greatly affected by the scale of the system's output values. Often, in ANNs values and data are normalised or bound between $\in [0, 1]$. This allows easy comparison between models. However, in real conductive *in-materio* systems, values are dependent on the test system's voltage limits, making comparison between EiM processor and ANN recorded mse loss results inappropriate. Therefore, a Normalised Mean Squared Error ($nmse$) objective function is introduced here, which allows a better comparison between different types of conductive *in-materio* systems, regardless of their operating voltages. The $nmse$ is defined by:

$$\Phi_{nmse} = \frac{1}{K(V^{max} - V^{min})^2} \sum_{k=1}^K (y(k) - \hat{y}(k))^2, \quad (6.5.10)$$

where V^{max} and V^{min} are the maximum and minimum system voltages respectively.

6.5.2 Performance

The resulting training and test reconstruction fitness convergence curves are given in Fig. 6.11, showing that the iM-AE was optimised during evolution. Therefore, the handwritten 8×8 pixel images were being successfully encoded as (and reconstructed from) a two-dimensional set of features ($\mathbb{R}^{64} \Rightarrow \mathbb{R}^2$). The final encoded features of the optimised system's test data are shown in Fig. 6.12a, and examples of test inputs and their corresponding reconstructed outputs are shown in Fig. 6.12b.

To assess the quality of the latent space, unsupervised KMeans clustering [41, 20] was applied to the encoded two-dimensional features. Then, as discussed in §2.6, these clusters were evaluated using the Clustering Accuracy (CA) and Adjusted Rand Index (ARI) metrics. The change in test feature clustering performance during evolution is plotted in Fig. 6.11b, showing that the data encoding improved as the system was optimised, leading to a more successful lower dimensional representation.

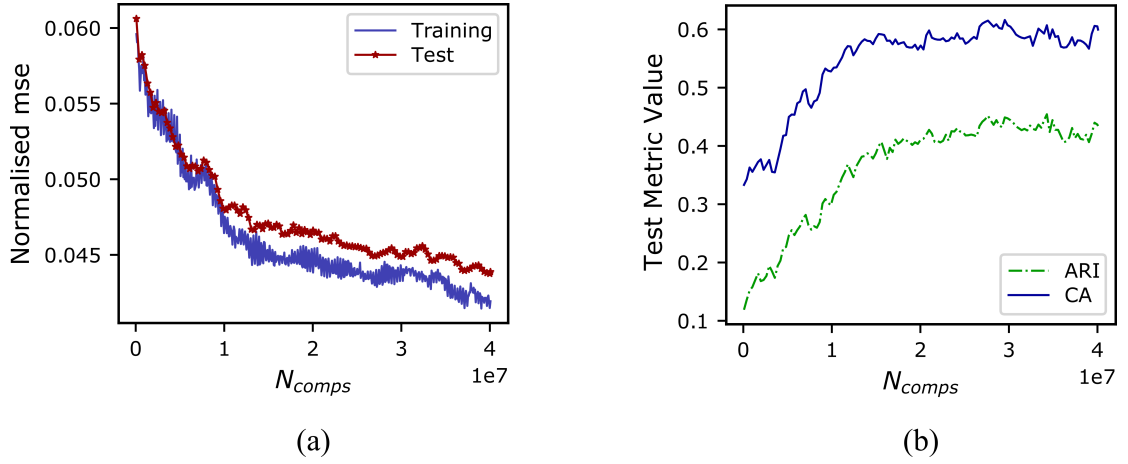


Figure 6.11: Convergence of (a) training and test fitnesses and (b) clustering metrics including Clustering Accuracy (CA) and Adjusted Rand Index (ARI).

Ideally, the encoded features should outperform those generated using Principal Component Analysis (PCA). Table 6.5 gives the clustering results of the iM-AE and PCA's encoded 2D features, showing that the example iM-AE performed better.

Finally, this example LDN neuron based physical iM-AE was compared against a back-propagation optimised ANN AutoEncoder. A five layer artificial AE was implemented using Keras [42] and trained using Adam optimise with a *mse* loss (comparable to the *nmse* loss used for the iM-AE example), using ReLU activation functions for the encoder and decoder layer (i.e., first and third HL), and linear activations in the central (i.e., second) HL and output layer. The size of the encoder and decoder layers was swept between three and sixty in increments of three, while

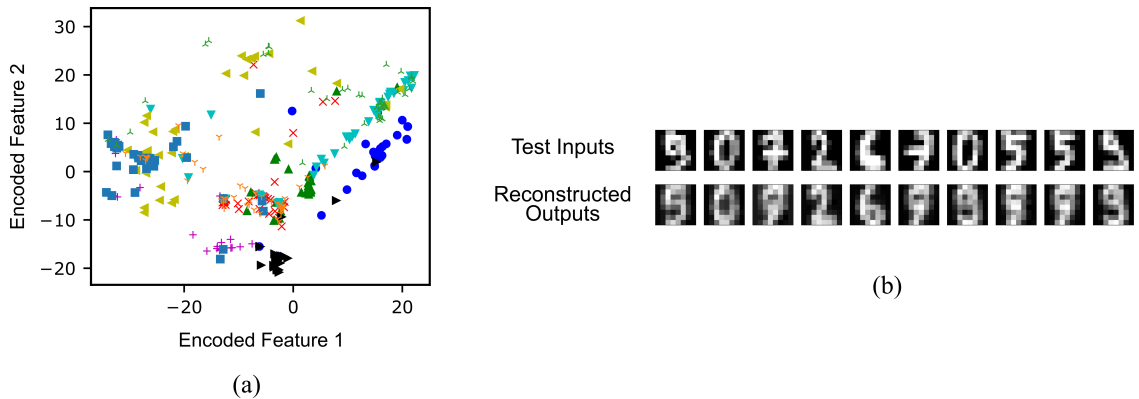


Figure 6.12: Final result's (a) encoded test data features, where labels are distinguished by colour and marker type, and (b) examples of test data inputs with their corresponding reconstructed outputs.

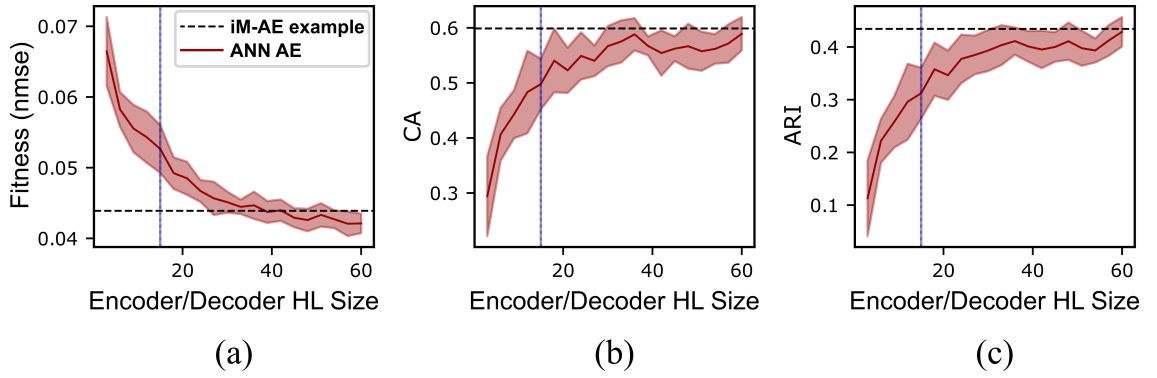


Figure 6.13: The artificial AEs mean final (a) reconstruction loss, and encoded 2D data’s clustering (b) CA, and (c) ARI for differently sized networks on the digits dataset. The final performance of the example iM-AE is marked for comparison, and the point at which the artificial AEs contains the same number of parameters is denoted by a vertical blue line.

the central HL contained only two neurons to perform dimensionality reduction as before. For each network size, ten systems were considered, each trained for 500 epochs. The resulting mean final test loss/fitness and encoded test data clustering performance is shown in Fig. 6.13.

Direct comparisons between the artificial and *in-materio* based AEs are difficult to make due to the different training methods and systems used. Therefore, to ensure competitive results, the artificial AE’s were each trained for approximately four times the number of epochs than the iM-AE example above. Even so, the iM-AE, containing only six physical LDN based neurons in the encoder/decoder HLs, far outperforms the artificial AEs of a similar size. While the iM-AE network contained only a few decoder/encoder HL neurons, each of these PiM neurons had three input and three output voltage electrodes, leading to more inter-layer connections and inflating the number of weights and biases required. Therefore, rather than comparing artificial and physical AEs containing the same number of neurons, networks

Table 6.5: Final test results for different trained methods encoding the digits dataset as two-dimensional features.

Method	Clustering metric	
	CA	ARI
iM-AE	0.5989	0.4343
PCA	0.5867	0.3914

containing the same number of parameters should be considered. The vertical blue line in Fig. 6.13 denotes artificial AE networks containing the same number of parameters as the iM-AE example (approximately 2080 parameters). Notably, the iM-AE still outperforms artificial AEs of similar parameter sizes, showing that the LDN based PiM neurons outperformed their artificial counterparts, and introduced additional meaningful computational capabilities to the system.

6.6 Summary

Typical single substrate ‘monolithic’ EiM processors are unlikely to scale well to larger ML tasks. Instead, CAPs can be operated in parallel, creating realisations of physical neurons which can be used in a NN like structure; referred to as iM-NNs. Methods to exploit, train and implement such novel systems were considered for the first time.

Firstly, iM-NNs were constructed by grouping physical neurons into a SLFN like structure and training them as an ELM. To enable efficient analysis, DRN based SiM neurons were used. It was found that these iM-ELMs could significantly outperform some common classification methods, as well as traditional (artificial) ELMs of a similar size. As the size of the conductive DRN based SiM neurons increased (i.e., the number of nodes/electrodes used in the conductive network), the performance of iM-ELMs with similar HL sizes improved.

Drawing from previous work with EiM processors, which showed that a single nanomaterial substrate can be tuned for a range of operations, a method to re-use a single substrate as several ‘virtual’ neurons was used. These virtual iM-ELMs, which leveraged only a single SiM neuron, performed comparably to the iM-ELMs which used several different SiM neurons. This suggests that conductive substrates, like the DRN based SiM neurons, can manifest different internal IV characteristics which were successfully exploited as several distinct virtual neurons.

Using a LDN as a realisation of a PiM neuron, a real SLFN iM-NN structure was implemented using a Raspberry Pi and Hardware Interface. These novel iM-NNs were investigated and optimised via neuroevolution – using DE, OpenAI-ES

and CMA-ES. All three algorithms benefited from an implementation of batching and a tuned population size, which helped ensure the efficient use of a limited computational budget. The algorithms successfully optimised LDN based PiM neuron iM-NNs to classify some ML datasets, showing the PiM neuron's non-monotonic properties were successfully exploited. These iM-NN outperformed some common classification techniques, and this work demonstrated the feasibility and scalability of these systems.

Having established the use and training of physical neuron based NN structures, a more complex five-layer NN was considered and operated as an AE. This iM-AE again leveraged LDN based PiM neurons within its encoder and decoder layers, but used artificial neurons in a central combinatorial layer used to generate the latent space. The iM-AE was trained with OpenAI-ES and a regressed output layer to perform dimensionality reduction on a hand drawn digits dataset. This outperformed PCA and artificial AE systems which contained a similar number of optimisable parameters. These iM-AEs could be used to produce stacked AEs and possibly train deep iM-NNs.

These iM-NNs draw on traditional ML techniques but leverage analogue systems as physical neurons. Indeed, simulated diode and physical Lambda Diode based conductive networks were shown to perform well as neurons within these networks. Notably, a single physical neuron could be re-used as several virtual HL neurons, each operating distinctly. While this forgoes the benefits of parallelised operation, it grants significant flexibility in the realisation of these physical NN designs. This work presents a method to produce scalable *in-materio* devices, which could lead to efficient *unconventional computing* assets in the future.

Bibliography

- [1] M. G. S. Murshed, C. Murphy, D. Hou, N. Khan, G. Ananthanarayanan, and F. Hussain, "Machine Learning at the Network Edge: A Survey," *ACM Computing Surveys*, vol. 54, no. 8, pp. 170:1–170:37, Oct. 2021. [Online]. Available: <http://doi.org/10.1145/3469029>
- [2] 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. [Online]. Available: <http://arxiv.org/abs/2007.12371>
- [3] G.-B. Huang, Q.-Y. Zhu, and C.-K. Siew, “Extreme learning machine: Theory and applications,” *Neurocomputing*, vol. 70, no. 1, pp. 489–501, Dec. 2006. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S0925231206000385>
- [4] J. Zhang, Y. Li, W. Xiao, and Z. Zhang, “Non-iterative and Fast Deep Learning: Multilayer Extreme Learning Machines,” *Journal of the Franklin Institute*, vol. 357, no. 13, pp. 8925–8955, Sep. 2020. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S0016003220302751>
- [5] Benedict. A. H. Jones, N. Al Moubayed, D. A. Zeze, and C. Groves, “In-Materio Extreme Learning Machines,” in *Parallel Problem Solving from Nature – PPSN XVII*, ser. Lecture Notes in Computer Science, G. Rudolph, A. V. Kononova, H. Aguirre, P. Kerschke, G. Ochoa, and T. Tušar, Eds. Cham: Springer International Publishing, 2022, pp. 505–519, “Reproduced with permission from Springer Nature”. [Online]. Available: https://doi.org/10.1007/978-3-031-14714-2_35
- [6] —, “Training In-Materio Neural Networks,” *Paper submitted for publication*, 2023.
- [7] T. Matias, R. Araújo, C. H. Antunes, and D. Gabriel, “Genetically optimized extreme learning machine,” in *2013 IEEE 18th Conference on Emerging Technologies Factory Automation (ETFA)*, Sep. 2013, pp. 1–8.
- [8] X. Hu, G. Feng, S. Duan, and L. Liu, “A Memristive Multilayer Cellular Neural Network With Applications to Image Processing,” *IEEE Transactions on Neural Networks and Learning Systems*, vol. 28, no. 8, pp. 1889–1901, Aug. 2017.
- [9] A. Amirsoleimani, F. Alibart, V. Yon, J. Xu, M. R. Pazhouhandeh, S. Ecoffey, Y. Beilliard, R. Genov, and D. Drouin, “In-Memory Vector-Matrix Multiplication in Monolithic Complementary Metal–Oxide–Semiconductor-Memristor Integrated Circuits: Design Choices, Challenges, and Perspectives,” *Advanced Intelligent Systems*, vol. 2, no. 11, p. 2000115, 2020. [Online]. Available: <https://onlinelibrary.wiley.com/doi/abs/10.1002/aisy.202000115>
- [10] A. J. Pérez-Ávila, E. Pérez, J. B. Roldán, C. Wenger, and F. Jiménez-Molinos, “Multilevel memristor based matrix-vector multiplication: Influence of the discretization method,” in *2021 13th Spanish Conference on Electron Devices (CDE)*, Jun. 2021, pp. 66–69.
- [11] K. A. Sankararaman, S. De, Z. Xu, W. R. Huang, and T. Goldstein, “The Impact of Neural Network Overparameterization on Gradient Confusion and Stochastic Gradient Descent,” Jul. 2020. [Online]. Available: <http://arxiv.org/abs/1904.06963>

- [12] N. Ganesh, “Rebooting Neuromorphic Hardware Design – A Complexity Engineering Approach,” *arXiv:2005.00522 [cs]*, Sep. 2020. [Online]. Available: <http://arxiv.org/abs/2005.00522>
- [13] A. Gaier and D. Ha, “Weight Agnostic Neural Networks,” *arXiv:1906.04358 [cs, stat]*, Sep. 2019. [Online]. Available: <http://arxiv.org/abs/1906.04358>
- [14] A. Ardakani, C. Condo, and W. J. Gross, “Sparsely-Connected Neural Networks: Towards Efficient VLSI Implementation of Deep Neural Networks,” Mar. 2017. [Online]. Available: <http://arxiv.org/abs/1611.01427>
- [15] D. C. Mocanu, E. Mocanu, P. Stone, P. H. Nguyen, M. Gibescu, and A. Liotta, “Scalable training of artificial neural networks with adaptive sparse connectivity inspired by network science,” *Nature Communications*, vol. 9, no. 1, p. 2383, Jun. 2018. [Online]. Available: <https://www.nature.com/articles/s41467-018-04316-3>
- [16] S. Ortín, M. C. Soriano, L. Pesquera, D. Brunner, D. San-Martín, I. Fischer, C. R. Mirasso, and J. M. Gutiérrez, “A Unified Framework for Reservoir Computing and Extreme Learning Machines based on a Single Time-delayed Neuron,” *Scientific Reports*, vol. 5, no. 1, p. 14945, Oct. 2015. [Online]. Available: <http://www.nature.com/articles/srep14945>
- [17] 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. [Online]. Available: <http://www.sciencedirect.com/science/article/pii/S0893608019300784>
- [18] P. Mujal, R. Martínez-Peña, J. Nokkala, J. García-Beni, G. L. Giorgi, M. C. Soriano, and R. Zambrini, “Opportunities in Quantum Reservoir Computing and Extreme Learning Machines,” *Advanced Quantum Technologies*, vol. 4, no. 8, p. 2100027, 2021. [Online]. Available: <http://onlinelibrary.wiley.com/doi/abs/10.1002/qute.202100027>
- [19] G. Li and P. Niu, “An enhanced extreme learning machine based on ridge regression for regression,” *Neural Computing and Applications*, vol. 22, no. 3, pp. 803–810, Mar. 2013. [Online]. Available: <https://doi.org/10.1007/s00521-011-0771-7>
- [20] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and É. Duchesnay, “Scikit-learn: Machine Learning in Python,” *Journal of Machine Learning Research*, vol. 12, no. 85, pp. 2825–2830, 2011. [Online]. Available: <http://jmlr.org/papers/v12/pedregosa11a.html>
- [21] D. E. Ratnawati, Marjono, Widodo, and S. Anam, “Comparison of activation function on extreme learning machine (ELM) performance for classifying the

- active compound,” *AIP Conference Proceedings*, vol. 2264, no. 1, p. 140001, Sep. 2020. [Online]. Available: <http://aip.scitation.org/doi/abs/10.1063/5.0023872>
- [22] J. Wang, S. Lu, S.-H. Wang, and Y.-D. Zhang, “A review on extreme learning machine,” *Multimedia Tools and Applications*, May 2021. [Online]. Available: <https://doi.org/10.1007/s11042-021-11007-7>
- [23] W. Cao, J. Gao, Z. Ming, and S. Cai, “Some Tricks in Parameter Selection for Extreme Learning Machine,” *IOP Conference Series: Materials Science and Engineering*, vol. 261, p. 012002, Oct. 2017. [Online]. Available: <https://doi.org/10.1088/1757-899x/261/1/012002>
- [24] L. G. Wright, T. Onodera, M. M. Stein, T. Wang, D. T. Schachter, Z. Hu, and P. L. McMahon, “Deep physical neural networks enabled by a backpropagation algorithm for arbitrary physical systems,” *Nature*, vol. 601, no. 7894, pp. 549–555, Jan. 2022. [Online]. Available: <http://arxiv.org/abs/2104.13386>
- [25] H.-C. Ruiz Euler, M. N. Boon, J. T. Wildeboer, B. van de Ven, T. Chen, H. Broersma, P. A. Bobbert, and W. G. van der Wiel, “A deep-learning approach to realizing functionality in nanoelectronic devices,” *Nature Nanotechnology*, vol. 15, no. 12, pp. 992–998, Dec. 2020. [Online]. Available: <http://www.nature.com/articles/s41565-020-00779-y>
- [26] M. Jabri and B. Flower, “Weight perturbation: An optimal architecture and learning technique for analog VLSI feedforward and recurrent multilayer networks,” *IEEE Transactions on Neural Networks*, vol. 3, no. 1, pp. 154–157, Jan. 1992.
- [27] J. Alspector, R. Meir, B. Yuhas, A. Jayakumar, and D. Lippe, “A Parallel Gradient Descent Method for Learning in Analog VLSI Neural Networks,” in *Advances in Neural Information Processing Systems*, vol. 5. Morgan-Kaufmann, 1992. [Online]. Available: <https://proceedings.neurips.cc/paper/1992/hash/1595af6435015c77a7149e92a551338e-Abstract.html>
- [28] M. N. Boon, H.-C. R. Euler, T. Chen, B. van de Ven, U. A. Ibarra, P. A. Bobbert, and W. G. van der Wiel, “Gradient Descent in Materio,” May 2021. [Online]. Available: <http://arxiv.org/abs/2105.11233>
- [29] 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. [Online]. Available: <http://ieeexplore.ieee.org/document/8123677/>
- [30] 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. [Online]. Available: <http://www.nature.com/articles/srep32197>

- [31] B. A. H. Jones, J. L. P. Chouard, B. C. C. Branco, E. G. B. Vissol-Gaudin, C. Pearson, M. C. Petty, N. Al Moubayed, D. A. Zeze, and C. Groves, "Towards Intelligently Designed Evolvable Processors," *Evolutionary Computation*, pp. 1–23, Mar. 2022. [Online]. Available: https://doi.org/10.1162/evco_a_00309
- [32] X. Zhang, J. Clune, and K. O. Stanley, "On the Relationship Between the OpenAI Evolution Strategy and Stochastic Gradient Descent," Dec. 2017. [Online]. Available: <http://arxiv.org/abs/1712.06564>
- [33] D. Wierstra, T. Schaul, J. Peters, and J. Schmidhuber, "Natural Evolution Strategies," in *2008 IEEE Congress on Evolutionary Computation (IEEE World Congress on Computational Intelligence)*. Hong Kong, China: IEEE, Jun. 2008, pp. 3381–3387. [Online]. Available: <http://ieeexplore.ieee.org/document/4631255/>
- [34] N. Hansen, "The CMA Evolution Strategy: A Tutorial," Apr. 2016. [Online]. Available: <http://arxiv.org/abs/1604.00772>
- [35] 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. [Online]. Available: <https://www.nature.com/articles/s41586-019-1901-0>
- [36] P. W. Battaglia, J. B. Hamrick, V. Bapst, A. Sanchez-Gonzalez, V. Zambaldi, M. Malinowski, A. Tacchetti, D. Raposo, A. Santoro, R. Faulkner, C. Gulcehre, F. Song, A. Ballard, J. Gilmer, G. Dahl, A. Vaswani, K. Allen, C. Nash, V. Langston, C. Dyer, N. Heess, D. Wierstra, P. Kohli, M. Botvinick, O. Vinyals, Y. Li, and R. Pascanu, "Relational inductive biases, deep learning, and graph networks," Oct. 2018. [Online]. Available: <http://arxiv.org/abs/1806.01261>
- [37] 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. [Online]. Available: <http://ieeexplore.ieee.org/document/1029882/>
- [38] 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.
- [39] P. Vincent, H. Larochelle, I. Lajoie, Y. Bengio, and P.-A. Manzagol, "Stacked Denoising Autoencoders: Learning Useful Representations in a Deep Network with a Local Denoising Criterion," *The Journal of Machine Learning Research*, vol. 11, pp. 3371–3408, Dec. 2010.
- [40] B. Li, Y. Li, and X. Rong, "The extreme learning machine learning algorithm with tunable activation function," *Neural Computing and Applications*, vol. 22, no. 3, pp. 531–539, Mar. 2013. [Online]. Available: <https://doi.org/10.1007/s00521-012-0858-9>

-
- [41] Douglas. Steinley, "K-means clustering: A half-century synthesis," *British Journal of Mathematical and Statistical Psychology*, vol. 59, no. 1, pp. 1–34, 2006. [Online]. Available: <https://onlinelibrary.wiley.com/doi/abs/10.1348/000711005X48266>
- [42] F. Chollet and e. al, "Keras," 2015. [Online]. Available: <https://keras.io>

Chapter 7

Conclusions

7.1 Hypothesis & Chapter Overview	159
7.2 Chapters and Contributions Summary	160
7.3 Thesis Conclusion	162
7.4 Further Work	163
Bibliography	166

7.1 Hypothesis & Chapter Overview

Evolution in-Materio (EiM) is a computational paradigm in which an Evolutionary Algorithm (EA) reconfigures a material's (or medium's) properties to achieve a specific computational function. The main purpose of this thesis was to explore fundamental questions about what material properties or configurable parameters lead to better performing EiM processors, and advance the EiM paradigm. Initial work used a simulated model-based approach to answer these questions, which enabled efficient investigation and analysis compared to the typically slow physical development and lab-based experimentation process. Additionally, drawing from concepts in the wider Machine Learning (ML) field, new *in-materio* device structures were considered. These contributed to addressing scaling issues found in typical monolithic (single substrate) EiM processors, but also boosted the performance of these *unconventional computing* devices.

This chapter details the contributions made by each of the results chapters,

before summarising with a thesis conclusion. Finally, several recommendations for future work are made.

7.2 Chapters and Contributions Summary

7.2.1 Chapter 4

The first results chapter considered how well performing monolithic EiM processors could be designed through the selection of nanomaterials and an EA for a target application. A physical Simulation Program with Integrated Circuit Emphasis (SPICE) based model of a nanomaterial network was used which allowed for both randomness, and the possibility of Ohmic and non-Ohmic conduction, which are characteristics often found in EiM processor substrates. These differing networks were exploited by Differential Evolution (DE), which optimised several configuration parameters (configuration voltage stimuli, a shuffle gene to reorder inputs, and signal weights) to solve different classification problems. It was found that intrinsic nanomaterial conductive properties could be exploited by the differing configuration parameters, clarifying the role and limitations of these techniques. The material properties altered the behaviour of the EiM processors, showing that material choice is important when formulating well performing devices.

7.2.2 Chapter 5

Having established simulation tools to investigate the performance of EiM systems, enhancements to monolithic EiM processors operating as classifiers were considered. Firstly, an adapted DE algorithm that included batching (or mini-batching) that enabled more useful generational updates per epoch, speeding up convergence to an acceptable solution. Secondly, the introduction of Binary Cross Entropy (BCE) as an objective function for the EA; unlike the commonly used classification error objective function, this was a continuous fitness metric that helped place the decision boundary in a better probabilistic position. Thirdly, 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’ or be slow to discover the true performance of a Configurable Analogue Processor (CAP). Finally, the introduction of a fully connected input layer allowed for a smooth search space and more flexible system topology.

7.2.3 Chapter 6

While typical monolithic EiM processors have been found to be promising *unconventional computing* devices, there remain challenges in scaling devices to larger, more complex problems. Drawing from other ML techniques, it was proposed that EiM processors could be stacked in parallel and operated more similarly to a Neural Network (NN). These novel in-Materio Neural Network (iM-NN) devices utilised conductive network based CAPs as realisations of physical neurons in a NN like structure.

These iM-NNs were first simulated and trained as Extreme Learning Machines (ELMs), which were found to outperform Artificial Neural Network (ANN) based ELMs of a similar size, and perform better with larger conductive network based neurons. The use of virtual neurons (where a single physical neuron is re-used as several virtual neurons) allowed for similar performance with more flexible system implementation, but sacrificed parallelism. Physical iM-NNs were then implemented using non-monotonically conductive Lambda Diode Network (LDN) based physical neurons, a Raspberry Pi and custom Hardware Interface (HI). These were successfully trained using neuroevolution, and the introduction of batching and tuning of population size improved the performance.

Having demonstrated the feasibility of constructing and training iM-NNs classifiers, a more complex unsupervised dimensionality reduction task was considered by constructing a physical LDN neuron based AutoEncoder (AE). This in-Materio AE was implemented in the lab and successfully optimised using OpenAI Evolutionary Strategy (OpenAI-ES) with a regressed output layer to compress 8×8 pixel handwritten digits into a two-dimensional representation, outperforming Principal Component Analysis (PCA) and backpropagation trained ANN based AEs.

7.3 Thesis Conclusion

To summarise, EiM is an effective method of exploiting a nanomaterial substrate (or potentially any configurable medium) for *unconventional computing*. Both simulated and physical conductive networks were used to implement and investigate EiM processors.

This work initially focused on monolithic (single substrate) EiM devices and investigated the benefits of different conductive network/nanomaterial and EA properties. This provided significant insight into the capabilities of different configurable parameters (voltage stimuli, electrode reassignment, input and output weights, etc.) and selected material properties, explaining their success (or lack of success) found previously in literature.

The use of simulated conductive networks as a proxy for a physical nanomaterial enabled the efficient analysis of these fundamental questions, but also the investigation and advancement of such EiM devices. Several enhancements were proposed, and together they provided guidance on the future production of more flexible, better performing, and robust monolithic EiM processors.

A scalable approach to exploiting physical systems as neurons in a NN like structure was proposed. Different methods for training these novel iM-NNs for classification were examined. Finally, the ability to implement other typical ANN structures, such as an AE was proven, showing the potential of *in-materio* based systems to physically realise or mimic some typical ML techniques.

This work presents an approach to exploit systems with interesting physical dynamics, and leverage them as a computational resource. Such systems could become low power, high speed, *unconventional computing* assets in the future.

7.4 Further Work

This thesis has helped progress the EiM paradigm, but there remains scope for continued work within this area. In the following, I offer some thoughts and recommendations for future avenues of research.

Hidden Layer and Physical Neuron Size

In §6.3 it was observed that for the same Hidden Layer (HL) size (i.e., number of physical neurons in parallel), larger physical neurons, with more nodes/electrodes, performed more successfully when trained as in-Materio Extreme Learning Machines (iM-ELMs). However, to make informed decisions about iM-NN HL and physical neuron (i.e., nanomaterial substrate) sizes, further research is needed.

As mentioned, the work presented in this thesis provides some initial direction to this further work. Notably, when comparing ANNs with iM-NNs (or other iM-NNs leveraging different types of physical neurons), only considering HL size was sometimes inappropriate. This was demonstrated in §6.5 where physical neurons were found to introduce more optimisable parameters (i.e., weights and biases) than their artificial counterparts, inflating the search space. Instead, considering systems with a similar number of parameters was considered to be a fairer comparison. Determining the interaction and benefit between scaling either the HL or physical neuron substrate sizes will allow for a more systematic approach to future physical neuron-based NN design.

Sparse Systems and Emergent Behaviour

An *in-materio* focused approach to future research is encouraged, which maintains efforts to exploit a material's (or medium's) inherent complex properties alongside other improvements. While ML concepts have helped progress the EiM paradigm in this work, a cautious attitude should be taken to ensure that *in-materio* substrates and physical neurons are being appropriately exploited. As discussed in §6.2.1, ML models such as ANN are sometimes considered to be over-parameterised and instead attempts to reduce the number of optimisable parameters and exploit emergent be-

haviour might be advantageous [1]. Using sparse layers is one way of drastically reducing system parameters, but maintaining similar performance [2]. Indeed, implementing a sparse evolutionary training procedure might be easier in a custom *in-materio* system than in a typical ANN matrix-multiplication workflow.

Novel *in-materio* substrates

There remains significant scope to develop new *in-materio* device substrates. These might allow operation at significantly increased speeds or power efficiency. However, as discussed in §4.3.4 and §6.4.4, the benefits of *static* material and *dynamic* material should be considered and selected based on the target computational task.

For example, as researchers seek to replicate brain like behaviour by implementing *neuromorphic computing* [3], it seemed inevitable that physical biological brain matter would be used. Indeed, examples include an *in-vitro* Reservoir-Neuro system [4] and other Biological Reservoir Computing (RC) [5]. So called “Organoid Intelligence” [6] is an emerging field which includes developments using human stem cell-derived brain organoids to replicate critical molecular and cellular aspects of learning, memory and possibly aspects of cognition in vitro. This includes technologies that could enable novel biocomputing models via stimulus-response training. This presents an opportunity with potential overlap with EiM processors, perhaps operating brain cell cultures similarly to a CAP.

Pre-trained *in-materio* devices and Kernels

The iM-ELMs developed in §6.3 were found to perform well as basic classifiers, where the input layer and physical neuron stimuli were randomly initialised and exploited. These systems assume that the input data will be projected to a new, useful representation at the HL outputs, which are then combined by a regressed output layer. However, some physical neurons might be ill configured for this purpose, or an input layer may fail to deliver meaningful physical neuron input signals.

Instead, some degree of pre-training could be performed to ensure that an EiM processor or iM-NN successfully creates a new, spatially diverse representation of the input data. Novelty Search (NS) [7] is one such method which replaces the objective

function in a standard evolutionary search process with a function that rewards novelty rather than a performance-based fitness metric [8]. Such a concept could be used to pre-train an EiM processor’s output states, or an iM-NN’s HL’s outputs, to ensure a dataset is projected to a new novel representation. These systems could then be used similarly to a kernel [9].

Complex Body Analysis and Exploitation

Here, the potential self-use of a system to extract useful information is highlighted. For example, RC can be used to operate an origami robot [10]. This is perhaps more interesting when considered in the context of soft body robots, where the physical body itself is similarly used as a reservoir [11]. Soft robots are often under-actuated (where the number of the actuation points are less than the degrees of freedom), and when actuated they often generate diverse and complex body dynamics – which are high-dimensional, nonlinear, and contain short-term memory. These properties create challenges when it comes to control, but present an opportunity for implementing Physical RC.

It is suggested that by considering a dynamic system itself as a Physical RC it might (i) enable complex information to be gathered and inferred from the system, perhaps more efficiently than conventional digital sensing techniques; and (ii) the dynamic system might be exploited as a computational resource, while still functioning for its initial purpose.

Notably, these systems could be realised without digital electronics involved whatsoever. For example, acoustic logic gates could be used to analyse a robot’s movements [12], allowing us to avoid analogue to digital conversion, side-stepping limitations associated with discretisation.

Bibliography

- [1] N. Ganesh, “Rebooting Neuromorphic Hardware Design – A Complexity Engineering Approach,” *arXiv:2005.00522 [cs]*, Sep. 2020. [Online]. Available: <http://arxiv.org/abs/2005.00522>
- [2] D. C. Mocanu, E. Mocanu, P. Stone, P. H. Nguyen, M. Gibescu, and A. Liotta, “Scalable training of artificial neural networks with adaptive sparse connectivity inspired by network science,” *Nature Communications*, vol. 9, no. 1, p. 2383, Jun. 2018. [Online]. Available: <https://www.nature.com/articles/s41467-018-04316-3>
- [3] S. Bains, “The business of building brains,” *Nature Electronics*, vol. 3, no. 7, pp. 348–351, Jul. 2020. [Online]. Available: <https://www.nature.com/articles/s41928-020-0449-1>
- [4] P. Aaser, M. Knudsen, O. H. Ramstad, R. Wijdeven, S. Nichele, I. Sandvig, G. Tufte, U. Bauer, Ø. Halaas, S. Hendseth, A. Sandvig, and V. Valderhaug, “Towards making a cyborg: A closed-loop reservoir-neuro system,” in *ECAL*, 2017.
- [5] 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. [Online]. Available: <http://www.sciencedirect.com/science/article/pii/S0893608019300784>
- [6] L. Smirnova, B. S. Caffo, D. H. Gracias, Q. Huang, I. E. Morales Pantoja, B. Tang, D. J. Zack, C. A. Berlinicke, J. L. Boyd, T. D. Harris, E. C. Johnson, B. J. Kagan, J. Kahn, A. R. Muotri, B. L. Paulhamus, J. C. Schwamborn, J. Plotkin, A. S. Szalay, J. T. Vogelstein, P. F. Worley, and T. Hartung, “Organoid intelligence (OI): The new frontier in biocomputing and intelligence-in-a-dish,” *Frontiers in Science*, vol. 1, 2023. [Online]. Available: <https://www.frontiersin.org/articles/10.3389/fsci.2023.1017235>
- [7] J. Lehman and K. O. Stanley, “Abandoning Objectives: Evolution Through the Search for Novelty Alone,” *Evolutionary Computation*, vol. 19, no. 2, pp. 189–223, Jun. 2011.
- [8] A. Marrero, E. Segredo, C. León, and E. Hart, “A Novelty-Search Approach to Filling an Instance-Space with Diverse and Discriminatory Instances for the Knapsack Problem,” in *Parallel Problem Solving from Nature – PPSN XVII*, ser. Lecture Notes in Computer Science, G. Rudolph, A. V. Kononova, H. Aguirre, P. Kerschke, G. Ochoa, and T. Tušar, Eds. Cham: Springer International Publishing, 2022, pp. 223–236.
- [9] T. Hofmann, B. Schölkopf, and A. J. Smola, “Kernel methods in machine learning,” *The Annals of Statistics*, vol. 36, no. 3, pp. 1171–1220, Jun. 2008. [Online]. Available:

<https://projecteuclid.org/journals/annals-of-statistics/volume-36/issue-3/Kernel-methods-in-machine-learning/10.1214/0090536070000000677.full>

- [10] P. Bhovad and S. Li, “Physical reservoir computing with origami and its application to robotic crawling,” *Scientific Reports*, vol. 11, no. 1, p. 13002, Jun. 2021. [Online]. Available: <http://www.nature.com/articles/s41598-021-92257-1>
- [11] K. Nakajima, “Physical reservoir computing—an introductory perspective,” *Japanese Journal of Applied Physics*, vol. 59, no. 6, p. 060501, May 2020. [Online]. Available: <https://doi.org/10.35848/1347-4065/ab8d4f>
- [12] A. Parsa, D. Wang, C. S. O’Hern, M. D. Shattuck, R. Kramer-Bottiglio, and J. Bongard, “Evolving Programmable Computational Metamaterials,” in *Proceedings of the Genetic and Evolutionary Computation Conference*, Jul. 2022, pp. 122–129. [Online]. Available: <http://arxiv.org/abs/2204.08651>

Appendix A

Additional Theoretical Background

A.1 Complexity Measures

The complexity of the classification datasets use din this thesis is considered in §3.6.1. This included the use of python *proplexity* package [1], which computes a number of complexity measures commonly found in literature [2, 3, 1].

These complexity measures used fall into several groups, and are detailed as follows [1]:

- **Feature Based** measures that characterize how informative the available features are to separate the classes:
 - **f1** Maximum Fisher’s discriminant ratio, which describes the overlap of feature values in each class.
 - **f1v** Directional vector maximum Fisher’s discriminant ratio, computes projection that maximizes class separation by directional Fisher’s criterion.
 - **f2** Volume of overlapping region, describes the overlap of the feature values within the positive and negative classes.
 - **f3** Maximum individual feature efficiency, describes the efficiency of each feature in the separation of classes.
 - **f4** Collective feature efficiency, describes the features’ synergy.

- **Linearity** measures that try to quantify whether the classes can be linearly separated:
 - **l1** Sum of the error distance by linear programming.
 - **l2** Error rate of linear classifier, described by the error rate of the Linear SVM classifier within the dataset.
 - **l3** Non-linearity of linear classifier, described by the classifier's error rate on synthesized points of the dataset.
- **Neighbourhood** measures that characterize the presence and density of same or different classes in local neighbourhoods, often using a Nearest Neighbour (NN) Classifier:
 - **n1** Fraction of borderline points.
 - **n2** Ratio of intra/extra class NN distance, depends on the distances of each problem instance to its nearest neighbour of the same class and the distance to the nearest neighbour of a different class.
 - **n3** Error rate of NN classifier.
 - **n4** Calculates the Non-linearity of NN classifier.
 - **t1** Fraction of hyperspheres covering data, defined by the number of hyperspheres needed to cover the data, divided by the number of instances.
 - **lsc** Local set average cardinality, depends on the distances between instances and the distances to the instances' nearest enemies – the nearest sample of the opposite class.
- **Network** measures that extract structural information from the dataset by modelling it as a (epsilon-Nearest Neighbour) graph:
 - **density** Density metric, which calculates the number of edges in the final graph divided by the total possible number of edges.
 - **cls coef** Clustering Coefficient metric,

- **hubs** Hubs metric, which scores each sample by the number of connections to neighbours, weighted by the number of connections the neighbours have
- **Dimensionality** measures that evaluate data sparsity based on the number of samples relative to the data dimensionality:
 - **t2** Average number of features per dimension.
 - **t3** Average number of Principal Component Analysis (PCA) dimensions per points.
 - **t4** Ratio of the PCA dimension to the original dimension, which describes the proportion of relevant dimensions in the dataset.
- **Class Imbalance** measures that consider the ratio of the numbers of examples between classes:
 - **c1** Entropy of Class Proportions.
 - **c2** Imbalance Ratio.

Bibliography

- [1] J. Komorniczak and P. Ksieniewicz, “Problexity—An open-source Python library for supervised learning problem complexity assessment,” *Neurocomputing*, vol. 521, pp. 126–136, Feb. 2023. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S0925231222014461>
- [2] T. K. Ho and M. Basu, “Complexity measures of supervised classification problems,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 24, no. 3, pp. 289–300, Mar. 2002.
- [3] A. C. Lorena, L. P. F. Garcia, J. Lehmann, M. C. P. Souto, and T. K. Ho, “How Complex is your classification problem? A survey on measuring classification complexity,” Dec. 2020. [Online]. Available: <http://arxiv.org/abs/1808.03591>

Appendix B

Physical System Validation

B.1 SWCNT-PMMA Experiments

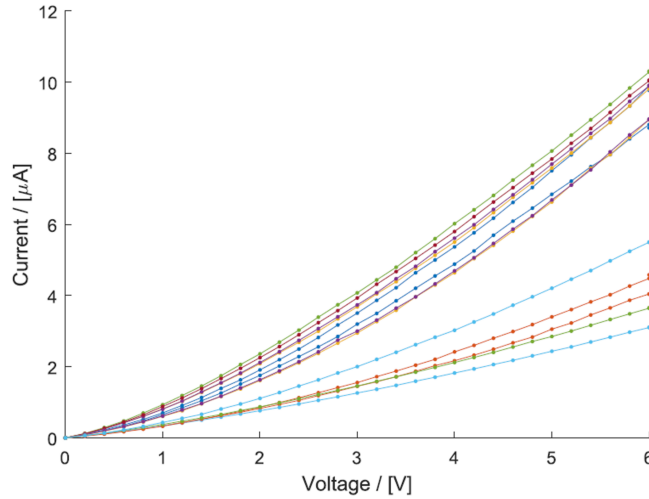


Figure B.1: Example inter-node IV characteristics between one node and 10 others on the fabricated microelectrode array with deposited 1.1wt% Single Walled Carbon Nanotubes (SWCNTs) suspended in Poly(butyl methacrylate) (PBMA).

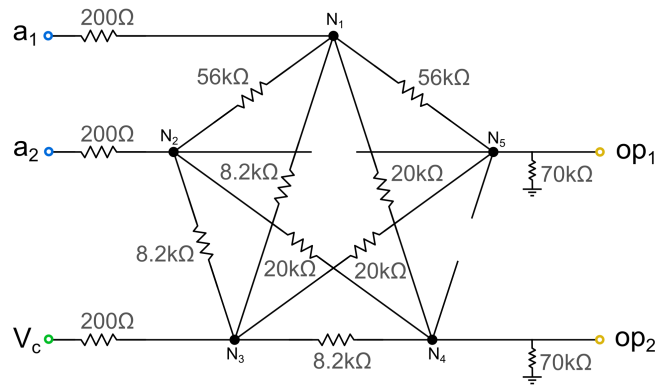
Table B.1: Summary fitting parameters of Eq.(3.4.6) to experimental data for internode characteristics of a 1.1wt% SWCNT/PBMA blend.

Paramater		Minimum	Maximum	Mean	Std
1.1%	a	3.3×10^{-8}	1.7×10^{-7}	1.2×10^{-7}	4.8×10^{-8}
	b	2.8×10^{-7}	9.6×10^{-7}	5.6×10^{-7}	2.4×10^{-8}

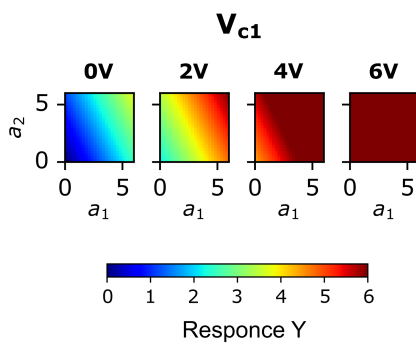
B.2 RRN and DRN System Experiments

B.2.1 RRN

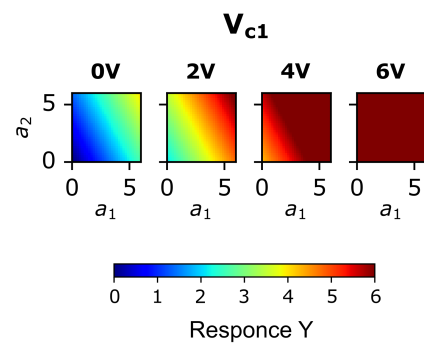
A simple example resistor based circuit network was built to compare experimental results with Simulation Program with Integrated Circuit Emphasis (SPICE) simulated results, as shown in Fig. B.2. The voltages at the two input nodes (a_1 & a_2) were swept in steps of $0.2V$. Various configuration voltages were considered, no other algorithm configuration parameters were utilised. The real and simulated circuits were found to produce near identical responses.



(a) Constructed example Resistor Random Network (RRN) network



(b) Experimental Results

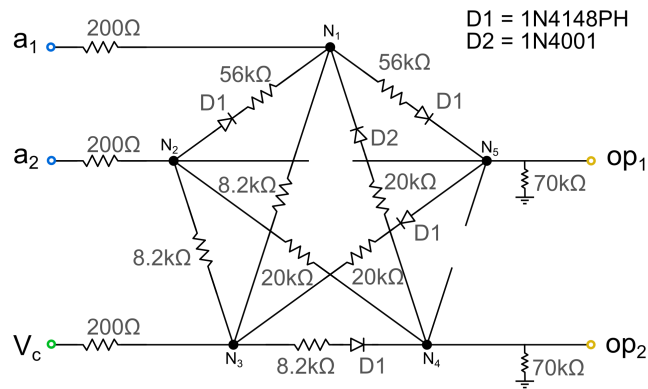


(c) Simulated Results

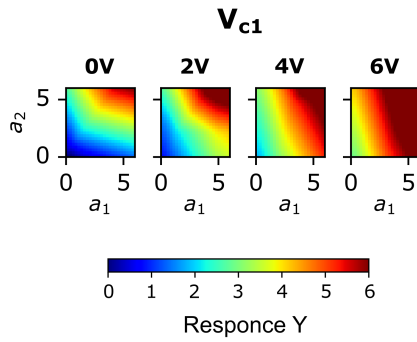
Figure B.2: Experimental and simulated response (Y) surface plots for an example Recurrent Neural Network (RNN) network.

B.2.2 DRN

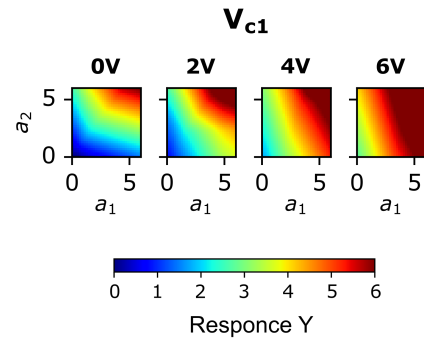
A simple example resistor and diode based circuit network was built to compare experimental results with SPICE simulated results, as shown in Fig. B.3; both 1N4148PH and 1N4001 diodes were used. The voltages at the two input nodes (a_1 & a_2) were swept in steps of $0.2V$. Various configuration voltages were considered, no other algorithm configuration parameters were utilised. The real and simulated circuits were found to produce near identical responses.



(a) Constructed example Diode Random Network (DRN) network



(b) Experimental Results



(c) Simulated Results

Figure B.3: Experimental and simulated response (Y) surface plots for an example DRN network.

B.3 Lambda Diode Network System Experiments

A Lambda Diode Network (LDN) was leveraged as a physical neuron to implement an in-Materio Neural Network as described in Chapter 6. To visualise its behaviour, a 2D sweep of voltage inputs was performed for several configuration voltages, as discussed in §6.4.1. The accumulated residuals (i.e., difference between the desired/set voltages and the actual/read voltage) from this experiment, which used a Raspberry Pi and Hardware Interface in a lab, are plotted in Fig. B.4.

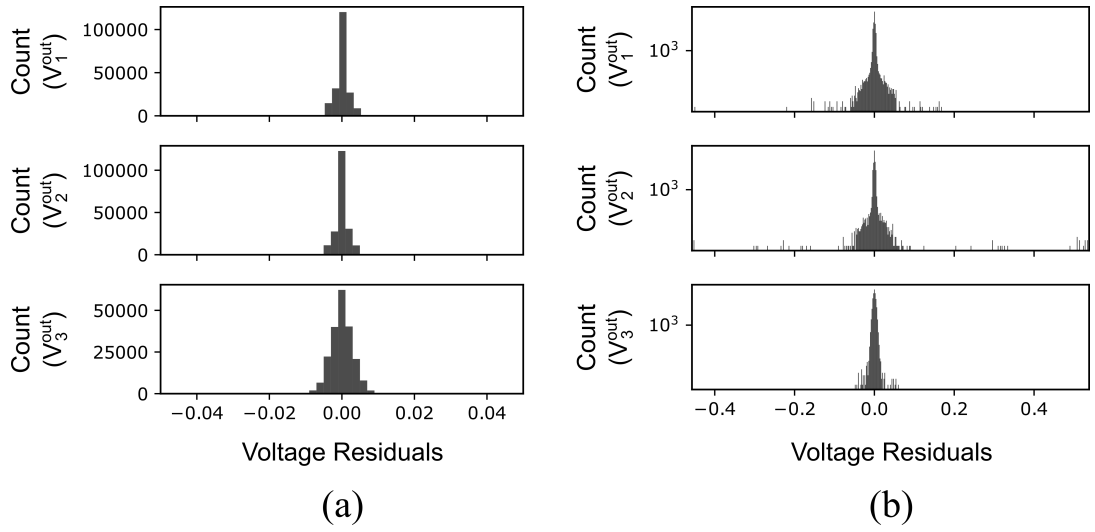


Figure B.4: Histogram of the output voltage residuals using a (a) linear and (b) log y-axis. Bin width = 0.002.

Appendix C

Additional Advanced Monolithic EiM Results

This appendix supports work presented in Chapter 4. This includes an investigation into an appropriate sample size of randomly generated ‘material processors’ and the number of repetitions required to mitigate randomness in convergence. This appendix also includes additional results which consider the effect of modifying the Decision Vector, as discussed in §4.3.2.

C.1 Repetition Reliability

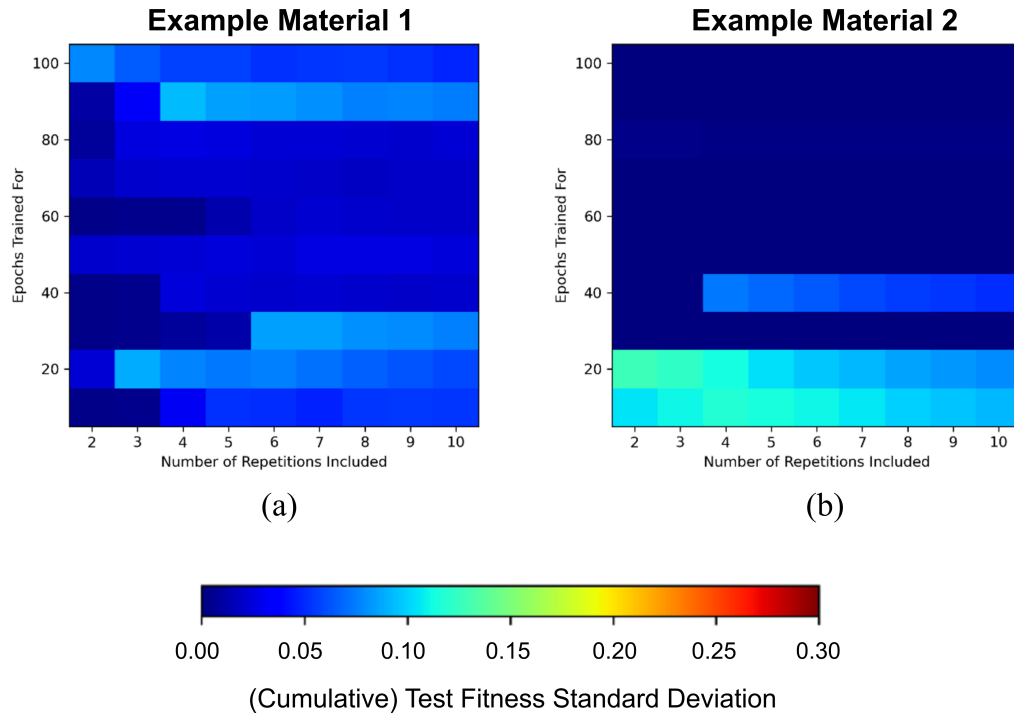
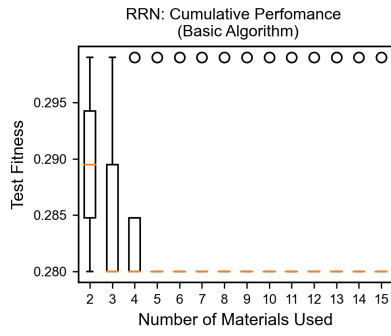
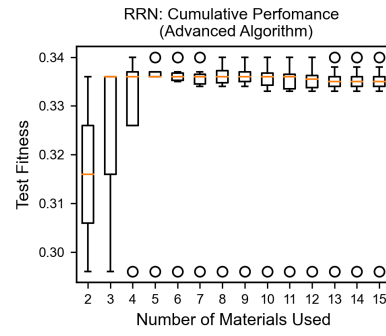


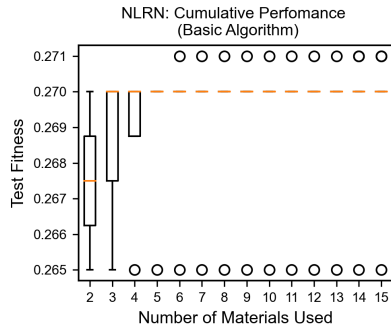
Figure C.1: Surface plot of the test fitness standard deviation as we either train for longer or include more repetitions. These results used the advanced Differential Evolution (DE) algorithm (with shuffle gene, input and output weights) to classify the con2DDS. It shows the effect of the cumulative standard deviation of the evolved final test fitness as more algorithm repetitions are introduced. This was done for two randomly generated Diode Random Network (DRN) material processors and shows that the standard deviation is low and settles after four to five repetitions of the DE algorithm.



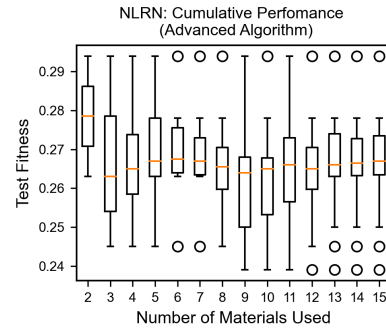
(a) Resistor Random Network (RRN) Basic Algo



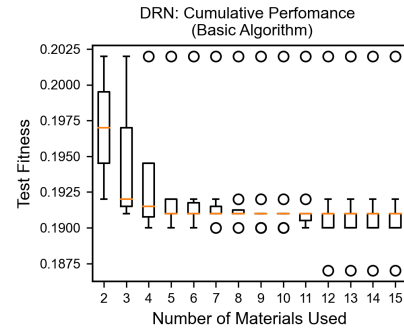
(b) RRN Advanced Algo



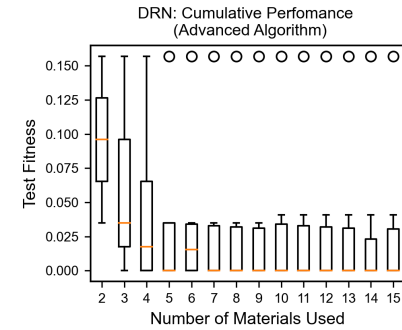
(c) Non-Linear Random Network (NLRN) Basic Algo



(d) NLRN Advanced Algo



(e) DRN Basic Algo



(f) DRN Advanced Algo

Figure C.2: Box & whisker plots showing the effect of including more randomly generated ‘material processors’ into the final result. Each of the processor types (RRN, NLRN & DRN) were solved using the basic and advanced algorithms over 50 iterations. This was then repeated for a new material so a box plot of the cumulative test fitness’s could be plotted. This was then repeated for another newly randomly generated material (and so on) such that the effect of including more material processors could be visualised. Note that introducing multiple repetitions of the DE algorithm on each individual material also improves stability.

C.2 con2DDS Results

For clarity, the results presented in Fig.4.9 displayed only the different individual additional configuration parameters (i.e., ‘shuffle’, input weights or output weights) in conjunction with configuration voltages, and the use of all configuration parameters at once. Here, the remaining combinations are shown in Fig.C.3, and a full table of results is given in Table C.1.

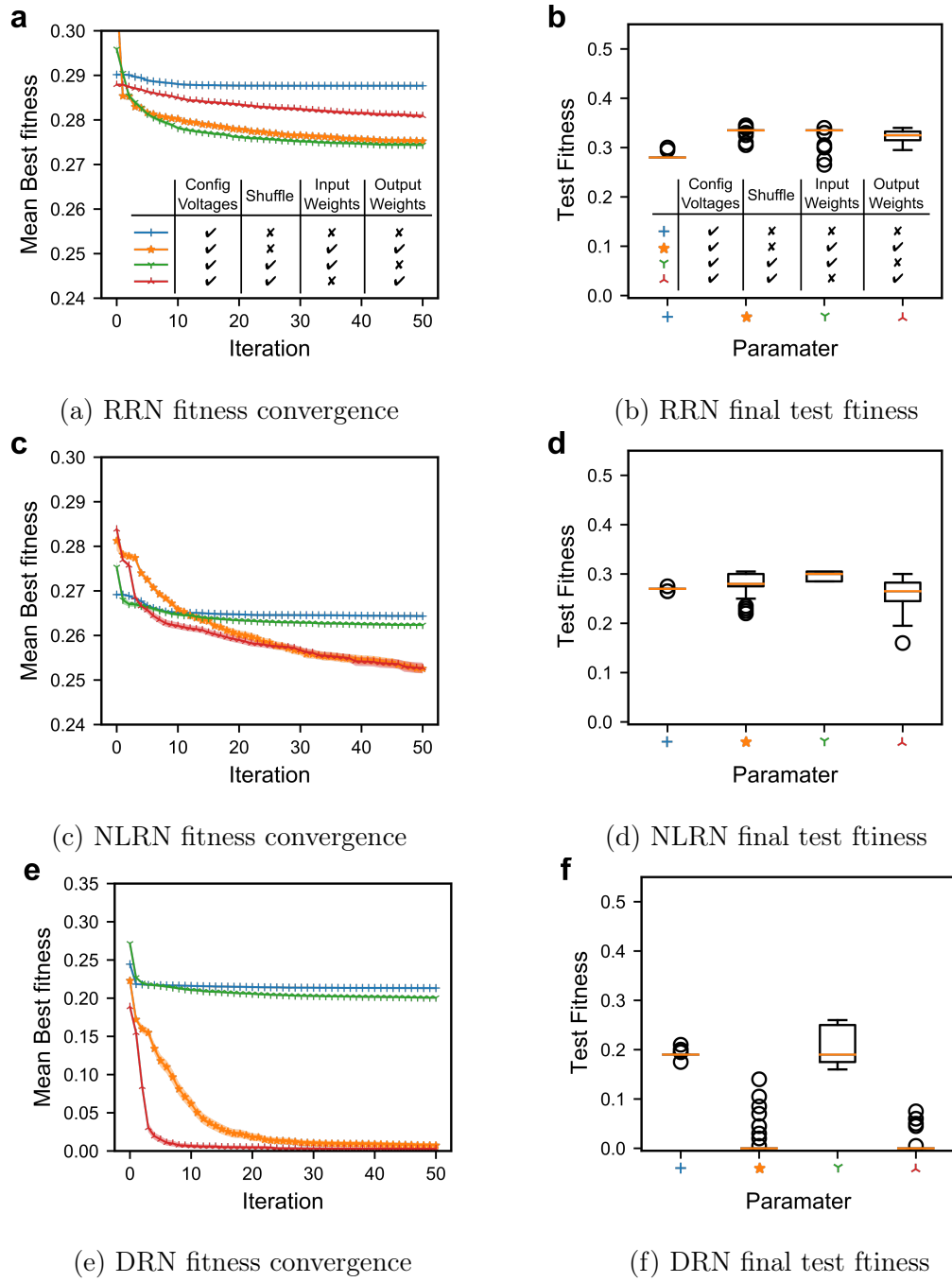


Figure C.3: The mean test fitness convergence and final test fitness box and whisker plots for the RRN, NLRN, and DRN.

Table C.1: Classification error (Φ_{error}) results from varying utilising different configuration parameters when classifying the con2DDS on the corresponding type of material processor (averaged from 15 material networks, each with 5 DE repetitions).

Material Model	Evolutionary Parameters				Training			Test		
	Config Voltage	Shuffle Gene	Input Weights	Output Weights	$\bar{\Phi}$	std(Φ)	Φ^*	$\bar{\Phi}$	std(Φ)	Φ^*
RRN	1	0	0	0	0.350	0.005	0.341	0.375	0.011	0.350
	1	0	1	0	0.313	0.003	0.309	0.337	0.016	0.300
	1	0	0	1	0.315	0.006	0.309	0.340	0.018	0.312
	1	0	1	1	0.314	0.005	0.309	0.337	0.016	0.287
	1	1	0	0	0.342	0.005	0.331	0.361	0.013	0.350
	1	1	1	0	0.313	0.003	0.309	0.334	0.011	0.312
	1	1	0	1	0.312	0.003	0.309	0.337	0.012	0.312
	1	1	1	1	0.313	0.004	0.309	0.337	0.014	0.287
NLRN	1	0	0	0	0.320	0.005	0.312	0.333	0.014	0.300
	1	0	1	0	0.309	0.003	0.300	0.303	0.021	0.275
	1	0	0	1	0.297	0.023	0.206	0.297	0.031	0.188
	1	0	1	1	0.299	0.018	0.219	0.294	0.023	0.225
	1	1	0	0	0.315	0.005	0.303	0.330	0.013	0.300
	1	1	1	0	0.308	0.004	0.300	0.303	0.020	0.275
	1	1	0	1	0.293	0.023	0.203	0.295	0.030	0.200
	1	1	1	1	0.299	0.023	0.194	0.292	0.025	0.212
DRN	1	0	0	0	0.375	0.051	0.281	0.385	0.054	0.287
	1	0	1	0	0.305	0.040	0.206	0.316	0.047	0.212
	1	0	0	1	0.223	0.059	0.119	0.222	0.072	0.087
	1	0	1	1	0.191	0.050	0.097	0.203	0.057	0.075
	1	1	0	0	0.292	0.041	0.244	0.307	0.042	0.250
	1	1	1	0	0.259	0.046	0.197	0.270	0.052	0.212
	1	1	0	1	0.154	0.054	0.037	0.147	0.068	0.013
	1	1	1	1	0.169	0.051	0.037	0.168	0.067	0.025

C.3 2DDS Results

Additional results using the advanced Evolution in-Materio (EiM) algorithm to classify the linearly separable 2DDS were carried out. This included all the combinations of additional decision parameters (i.e., ‘shuffle’, input weights or output weights) in conjunction with configuration voltages. Convergence results are shown in Fig.C.4, and a full table of results is given in Table C.2.

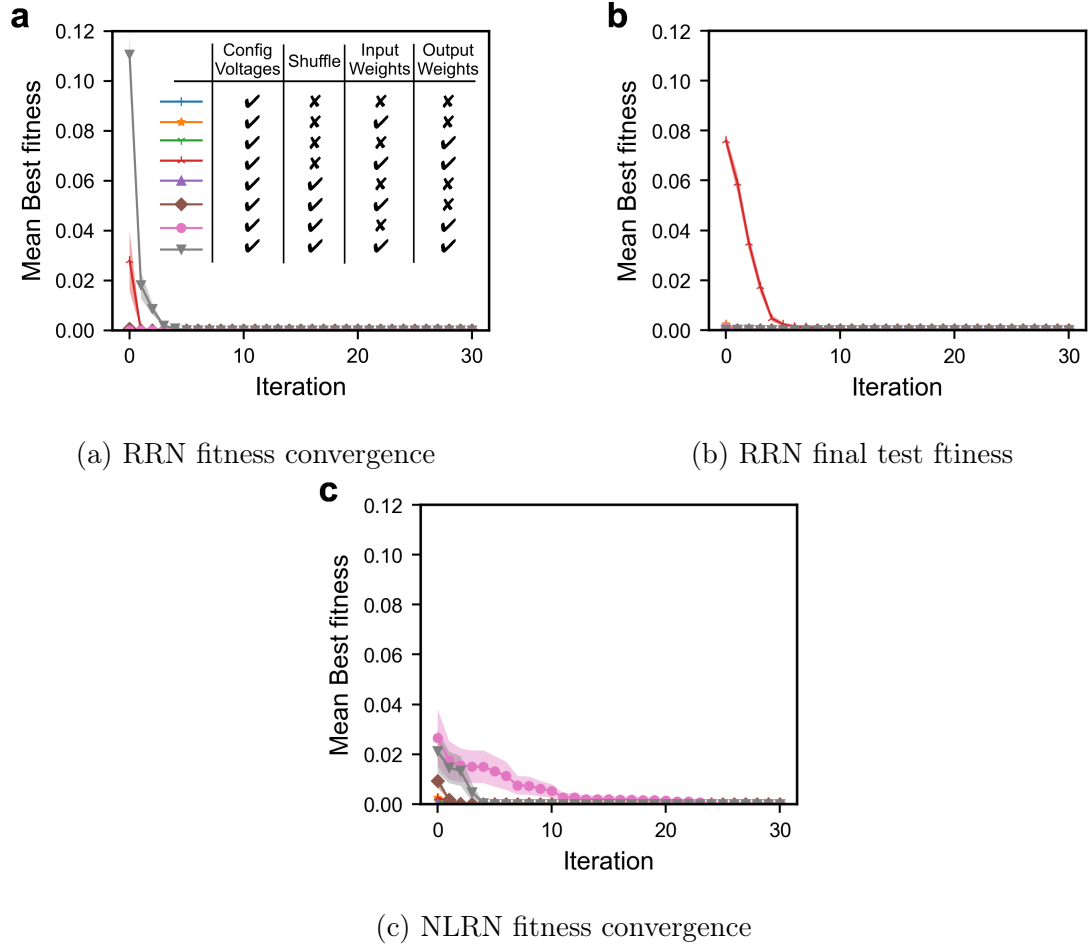


Figure C.4: The mean test fitness convergence for the different conductive network-based EiM processors classifying the 2DDS.

Table C.2: Classification error (Φ_{error}) results from varying utilising different configuration parameters when classifying the 2DDS on the corresponding type of material processor (averaged from 15 material networks, each with 5 DE repetitions).

Material Model	Evolutionary Parameters				Training			Test		
	Config Voltage	Shuffle Gene	Input Weights	Output Weights	$\bar{\Phi}$	std(Φ)	Φ^*	$\bar{\Phi}$	std(Φ)	Φ^*
RRN	1	0	0	0	0.000	0.000	0.000	0.000	0.000	0.000
	1	0	1	0	0.000	0.000	0.000	0.000	0.000	0.000
	1	0	0	1	0.000	0.000	0.000	0.000	0.000	0.000
	1	0	1	1	0.000	0.000	0.000	0.000	0.000	0.000
	1	1	0	0	0.000	0.000	0.000	0.000	0.000	0.000
	1	1	1	0	0.000	0.000	0.000	0.000	0.000	0.000
	1	1	0	1	0.000	0.000	0.000	0.000	0.000	0.000
	1	1	1	1	0.000	0.000	0.000	0.000	0.000	0.000
NLRN	1	0	0	0	0.000	0.000	0.000	0.000	0.000	0.000
	1	0	1	0	0.000	0.000	0.000	0.000	0.000	0.000
	1	0	0	1	0.000	0.000	0.000	0.000	0.000	0.000
	1	0	1	1	0.000	0.000	0.000	0.000	0.000	0.000
	1	1	0	0	0.000	0.000	0.000	0.000	0.000	0.000
	1	1	1	0	0.000	0.000	0.000	0.000	0.000	0.000
	1	1	0	1	0.000	0.000	0.000	0.000	0.000	0.000
	1	1	1	1	0.000	0.000	0.000	0.000	0.000	0.000
DRN	1	0	0	0	0.005	0.009	0.000	0.003	0.007	0.000
	1	0	1	0	0.002	0.006	0.000	0.002	0.005	0.000
	1	0	0	1	0.003	0.006	0.000	0.002	0.007	0.000
	1	0	1	1	0.001	0.003	0.000	0.001	0.002	0.000
	1	1	0	0	0.000	0.000	0.000	0.000	0.000	0.000
	1	1	1	0	0.000	0.002	0.000	0.000	0.001	0.000
	1	1	0	1	0.000	0.001	0.000	0.000	0.001	0.000
	1	1	1	1	0.000	0.000	0.000	0.000	0.000	0.000

C.4 flipped2DDS Results

Additional results using the advanced EiM algorithm to classify the linearly separable 2DDS were carried out. This included all the combinations of additional decision parameters (i.e., ‘shuffle’, input weights or output weights) in conjunction with configuration voltages. Convergence results are shown in Fig.C.5, and a full table of results is given in Table C.3.

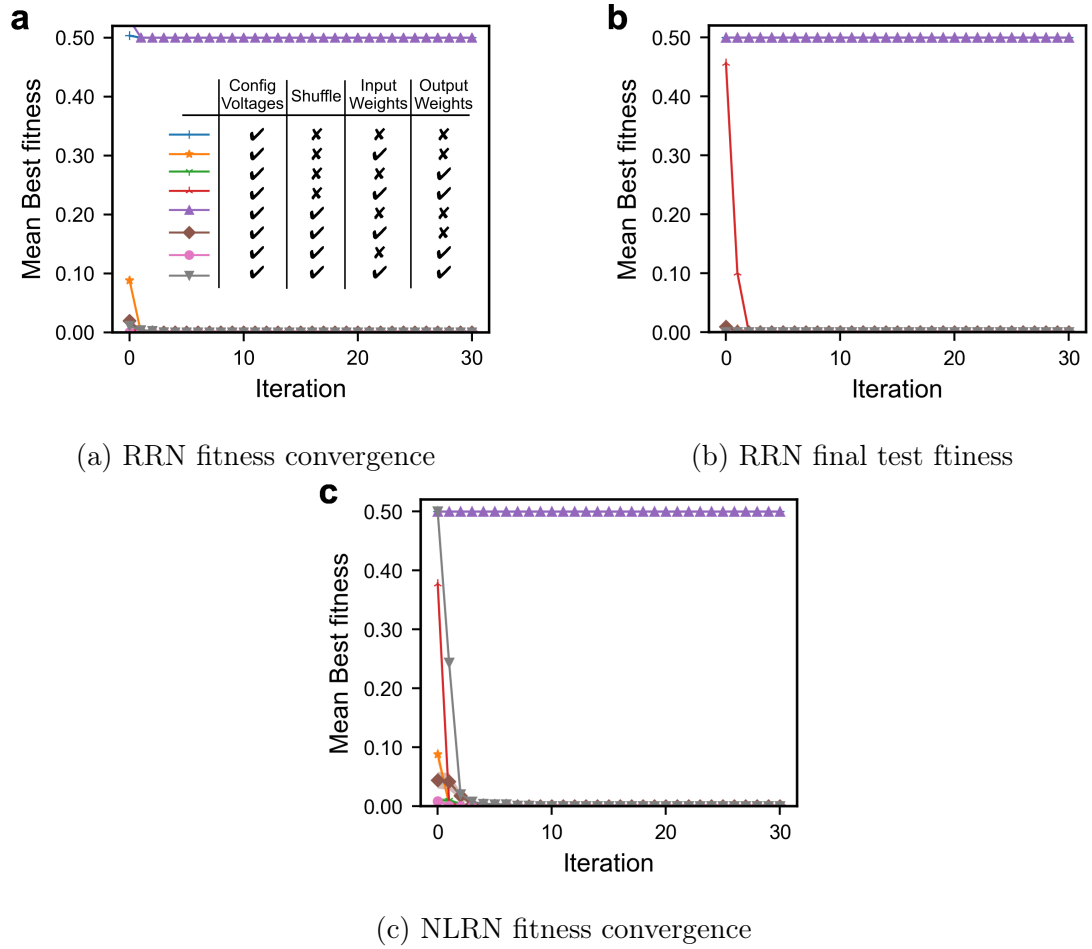


Figure C.5: The mean test fitness convergence for the different conductive network-based EiM processors classifying the flipped 2DDS.

Table C.3: Classification error (Φ_{error}) results from varying utilising different configuration parameters when classifying the flipped2DDS on the corresponding type of material processor (averaged from 15 material networks, each with 5 DE repetitions).

Material Model	Evolutionary Parameters				Training			Test		
	Config Voltage	Shuffle Gene	Input Weights	Output Weights	$\bar{\Phi}$	std(Φ)	Φ^*	$\bar{\Phi}$	std(Φ)	Φ^*
RRN	1	0	0	0	0.506	0.014	0.500	0.503	0.010	0.500
	1	0	1	0	0.000	0.000	0.000	0.000	0.000	0.000
	1	0	0	1	0.000	0.000	0.000	0.000	0.000	0.000
	1	0	1	1	0.000	0.000	0.000	0.000	0.000	0.000
	1	1	0	0	0.500	0.000	0.500	0.500	0.000	0.500
	1	1	1	0	0.000	0.000	0.000	0.000	0.000	0.000
	1	1	0	1	0.000	0.000	0.000	0.000	0.000	0.000
	1	1	1	1	0.000	0.000	0.000	0.000	0.000	0.000
NLRN	1	0	0	0	0.500	0.000	0.500	0.500	0.000	0.500
	1	0	1	0	0.000	0.000	0.000	0.000	0.000	0.000
	1	0	0	1	0.000	0.000	0.000	0.000	0.000	0.000
	1	0	1	1	0.000	0.000	0.000	0.000	0.001	0.000
	1	1	0	0	0.500	0.000	0.500	0.500	0.000	0.500
	1	1	1	0	0.000	0.000	0.000	0.000	0.000	0.000
	1	1	0	1	0.000	0.000	0.000	0.000	0.001	0.000
	1	1	1	1	0.000	0.000	0.000	0.000	0.000	0.000
DRN	1	0	0	0	0.500	0.001	0.500	0.500	0.000	0.500
	1	0	1	0	0.002	0.005	0.000	0.002	0.004	0.000
	1	0	0	1	0.003	0.007	0.000	0.003	0.010	0.000
	1	0	1	1	0.001	0.004	0.000	0.001	0.005	0.000
	1	1	0	0	0.500	0.000	0.500	0.500	0.000	0.500
	1	1	1	0	0.000	0.000	0.000	0.000	0.000	0.000
	1	1	0	1	0.000	0.001	0.000	0.000	0.001	0.000
	1	1	1	1	0.000	0.000	0.000	0.000	0.000	0.000

Appendix D

Physical iM-NN Neuroevolution Hyperparameter Investigation

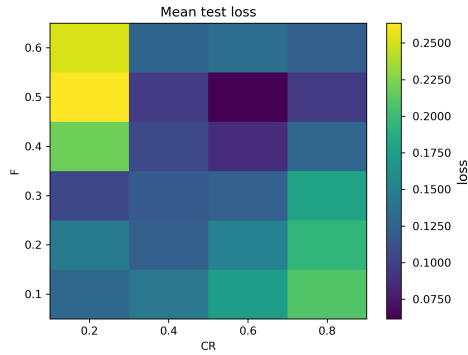
Before carrying out experimentation for the paper, some initial investigation was required to determine acceptable hyperparameter values for the three algorithms under consideration. The Differential Evolution (DE), OpenAI Evolutionary Strategy (OpenAI-ES) and Covariance Matrix Adaptation Evolution Strategy (CMA-ES) algorithms all utilise different hyperparameters.

To investigate these, in-Materio Neural Network (iM-NN) containing two physical hidden layer neurons, were used to classify a half moon 2d dataset (hm2DDS). The hm2DDS contains two attributes, two classes, and 1000 data instances (half allocated as class 1, and half as class 2), as described in §3.6.1; the algorithms are required to exploit non-linearities in the physical neurons to classify this dataset. The hm2DDS is split 70% – 30% to create a training and test data subset respectively, used to train and objectively measure performance.

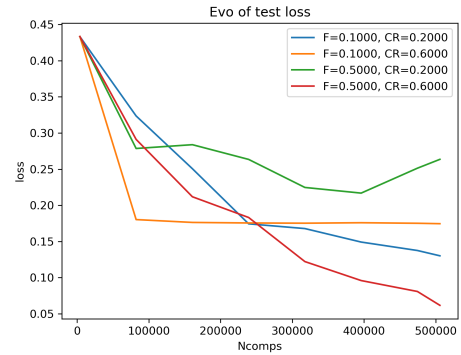
D.1 Differential Evolution

A DE/best/1/bin algorithm is used with a reflection bounds constraint. This has three hyperparameters which include the mutation factor F , a crossover CR and population size λ . Here, a reasonable population size of $\lambda = 1 \times d$ is selected, where d is the number of dimensions (i.e., evolvable system parameters) and use a batch size of $bs = 0.1 * K^{train}$ where K^{train} is the number of training data instances (i.e., 700).

A 2d hyperparameter sweep was carried out to consider the performance of the algorithm using different combinations of F and CR . For each combination, the algorithm was repeated three times for a fixed budget of $0.5 \times 10^6 N_{comps}$, and a surface plot of the mean final test fitnesses are shown in Fig. D.1a. These results show that $F = 0.5$ and $CR = 0.6$ are good hyperparameter values to be taken forward within the §6.4 experimentation.



(a) Final mean test fitness



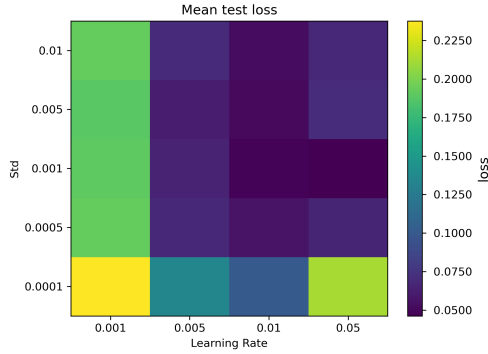
(b) Example mean test fitness convergence

Figure D.1: DE hyperparameter investigation results solving the hm2DDS for a fixed budget of 500000 N_{comps} , with $\lambda = d$.

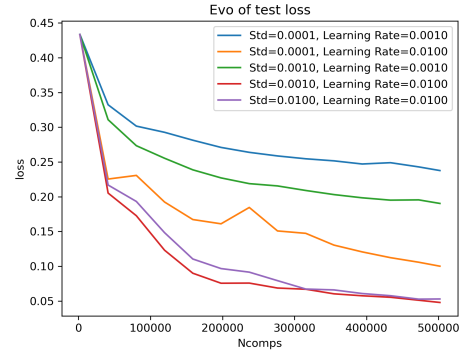
D.2 OpenAI ES

The OpenAI-ES is used with Adam optimiser and a ‘clip’ boundary constraint. Hyperparameters include Gaussian noise σ , learning rate α and pseudo-population λ ; but also additional hyperparameters introduced by Adam optimiser: $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\varepsilon = 10^{-8}$. Here, a reasonable population size of $\lambda = 1 \times d$ is selected, where d is the number of dimensions (i.e., evolvable system parameters) and use a batch size of $bs = 0.05 * K^{train}$ where K^{train} is the number of training data instances (i.e., 700).

A 2d hyperparameter sweep was carried out to consider the performance of the algorithm using different combinations of σ and α . For each combination, the algorithm was repeated three times for a fixed budget of $0.5 \times 10^6 N_{comps}$, and a surface plot of the mean final test fitnesses are shown in Fig. D.2a. These results show that $\sigma = 0.001$ and $\alpha = 0.01$ are reasonable hyperparameter values to be taken forward within §6.4 experimentation.



(a) Final mean test fitness



(b) Example mean test fitness convergence

Figure D.2: OpenAI-ES hyperparameter investigation results solving the hm2DDS for a fixed budget of $500000 N_{comps}$, with $\lambda = d$.

D.3 Covariance Matrix Adaptation ES

CMA-ES hyperparameters include initial step size (i.e., initial standard deviation of noise) σ and population size λ . The CMA-ES was used with the pop size automatically determined according to $\lambda = 4 + 3 \log(d)$. Therefore, only the σ hyperparameter values were investigated using a batch size of $bs = 0.05 * K^{train}$ where K^{train} is the number of training data instances (i.e., 700).

A 1d hyperparameter sweep was carried out to consider the performance of the algorithm using different combinations of σ . For each combination, the algorithm was repeated three times for a fixed budget of $0.5 \times 10^6 N_{comps}$, and a surface plot of the mean final test fitnesses are shown in Fig. D.2a. These results show that the initial step size does not impact performance too much, so a step size of $\sigma = 0.15$ was selected to be used in §6.4 experimentation (not that the initial step size is selected with respect to normalised boundaries $\in [0, 1]$ for the evolvable parameters).

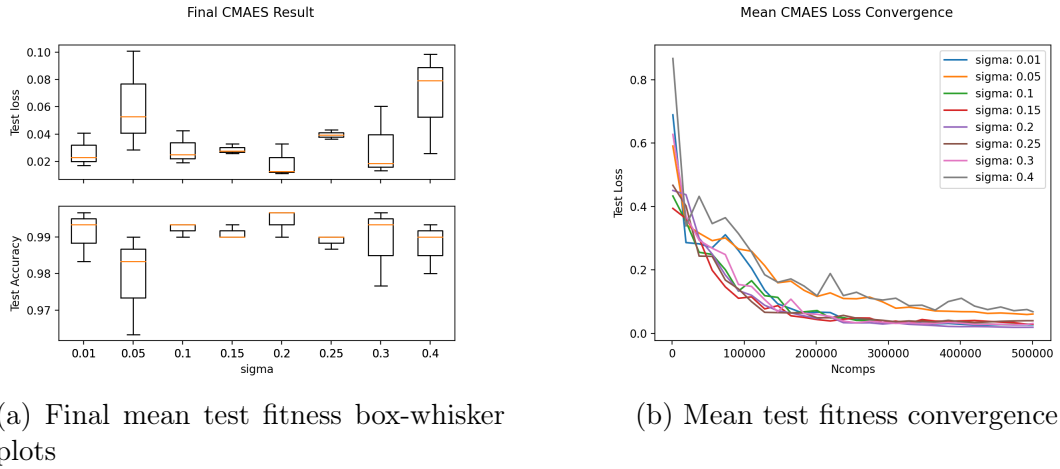


Figure D.3: CMA-ES hyperparameter investigation results solving the hm2DDS for a fixed budget of $500000 N_{comps}$.