Reservoir Computing as a model for *in-materio* computing

Matthew Dale, Julian F. Miller, and Susan Stepney

***SS: add notes like this for comments etc.

***JFM: add notes like this for comments etc.

***MD: last updated/checked 17/9/15

Abstract Research into substrate-based computing has shown that materials that contain rich properties can be exploited to solve computational problems. Modelling the computational processes occurring in these systems is a difficult task and understanding what part of the embodied system is doing the computation is still fairly ill-defined. This document discusses the prospects of using *Reservoir Computing* as a model for *in-materio* computing, introducing new training techniques (taken from Reservoir Computing) that could overcome training difficulties found in the current *Evolution-in-Materio* technique. Support for this hypothesis will be shown in examples of reservoir computing applied to other unconventional systems.

Key words: Material Computation, Evolution-in-Materio, Reservoir Computing, Unconventional Computing

1 Introduction

Biological organisms vastly outperform classical/conventional computing paradigms in many respects, from possessing inherent fault-tolerance to constructing highly parallel machines. Much of this is achieved by exploiting physicality and by sharing and distributing computational effort throughout the spatial system. As such, they

Matthew Dale
University of York, England, e-mail: md596@york.ac.uk

Name of Second Author ???
Name, Address of Institute e-mail: name@email.address

exploit physical interactions through feedback with the real-world, utilising features such as their own morphology.

Many of these systems comprise of relatively simple elements that emerge and coalesce into more complex, but robust, structural layers across different scales. Such grounding properties (and many more) have enabled these complex systems to thrive and evolve, adapting and co-evolving in real-time with their local ecosystem.

In modern research, many attempt to mimic the computational properties and behaviours of biological systems on conventional machines. It is argued here, that such attempts can themselves be flawed. Many of these experiments attempt to imitate the performance of a embodied system in a non-embodied abstraction, in a process that requires some transformation to a symbolic representation. In essence, we debate that such transformations can detract from many of the physical aspects that make these systems so powerful.

The classical/conventional computing architecture, although expertly refined over time, poses some fundamental inefficiencies. For example, classical computers require the transformation between high-level languages to low-level machine code - a process that requires layers of conversion through a compiler - making it computationally costly, slow and highly susceptible to faults and errors. These systems typically succumb to many issues in potential speed, from both an inability to deal with concurrent computations and the bottleneck created by the transfer of data between separate memory and processing entities. Because of these architectural weaknesses other intertwined discrepancies arise, such as an increase in power consumption, in system size and top-down design complexity.

Unconventional computing tries to address some of these limitations by attempting to provide alternative architectures and systems that typically exploit the underlying physics and many-scale interactions of the real-world. Many forms of unconventional systems have been explored in recent years; (**% insert examples %**)

For the purpose of this chapter we will discuss research into configurable *in-materio* systems (typically on the nanoscale), with the intention of identifying a computational model that could suit and conform to many *in-materio* systems.

The chapter begins with an introduction to the concepts of material computation with a brief mention of the criticality hypothesis (§2). Next, the field of Evolution-*in-materio* as a current investigation into the theory of material computation is reviewed, discussing its methodology and current research (§3). Reservoir Computing and its potential as an *in-materio* model is examined in (§4) and (§5). To further aid in identifying ”good” reservoirs some evaluation techniques are highlighted in (§6). Lastly, an argument is made in (§7) to the possible benefits reservoir computing could have on the design of computational substrates.

2 Material Computation

This section discusses some of the principle concepts of material computation, such as some informal definitions, the role critical dynamics has in complexity and where it may fit within other computing paradigms. Material computation itself is a conceptual term in which some computational process or mechanism maybe extracted from a sufficiently complex dynamical system in the form of a physical material substrate.

The definition provided here is derived from the axiom that all matter contains physical information and that systems of collective matter can dynamically modulate and redistribute information to change state, therefore, implied here as to performing some form of computation. A simple example of this would be the physical process in which the state or phase of matter changes in relation to energy, which could (loosely) in some sense be viewed as a computational mechanism.

However, Stepney *et al.* [98] provides a more contextual and appropriate definition of material computation as; "computation directly by physical and chemical processes of a complex substrate, with little or no abstraction to a virtual machine". From this definition we can more easily categorise and summarise material computation as an analogue process that utilises physical constructs, the tangible media itself and meta-processing to do computation. This "physicality" may be defined and observed as the structural topology, characteristic behaviours and information processing associated with the many-scale interactions occurring in that system.

As a notion, material computation is still bordering on its developmental/conceptual stage with early experiments supporting, or working in tandem with, current digital technology to form hybrid - and potentially very powerful - machines. In this respect, a configurable substrate can be used to, as suggested by Stepney *et al.*, transfer some of the computational burden from the digital system to the material. As such, the material can endow the system with many of the properties and advantages of analogue systems, such as speed and concurrency, in a device where memory and processing are not separable. To achieve this, we attempt to exploit processes and behavioural phenomena that naturally occur; properties that are governed directly by the underlying physics and chemistries of the substrate.

Exploiting computation directly from materials offers many potential advantages over classical systems where the computation performed does not depend so much on the details of the materials used. As a paradigm, substrate computing potentially offers vast amounts of computational power by capitalising on the massive parallelism and natural constraints of these rich systems. Such properties are suggested to have the potential to provide solutions "for free", or at least computationally cheaper, and provide a rich explorable state space, aligning computation to particular trajectories [96].

Much of the current interest in material computation is to abstract a model (or models) of computation from what the substrate does naturally. It has been proposed that the first step to producing a potential unified theory of material computation, i.e. a theory that better understands what computation is and how it occurs in materials, should take place in "primitive" (un-evolved) substrates, where the general principles are in plain sight [96]. After this, material computation could emerge with some supportive reasoning to a better understanding of computation in biological substrates.

2.1 The effects of criticality on complexity

A large body of literature discusses the hypothesis that there is a critical state in which a system can exhibit maximal computational power, and therefore where maximal complexity can be acquired, labelled a region near (or at) the *edge of chaos* [55]. This theory is raised here as it may have a direct relationship to material computation whereby a material can only exhibit "richness", and therefore be exploitable, by operating close to or within this region.

The *edge of chaos* represents the transitional border between ordered dynamics and chaotic behaviour, where perturbations to the system quickly fade and settle, or significantly affect long-term stability and become unpredictable. This critical landscape can be observed by looking at the systems trajectory in the phase space by monitoring the convergence towards or away from a steady state, and thus highlighting a systems sensitivity to initial conditions. Both of these distinct behaviours are said to be necessary to gain maximal complexity, using ordered behaviour to maintain memory and some chaotic behaviour to enable processing.

Langton [55] famously observed the effects/advantages of systems working in this transitional region using cellular automata. At a critical point, Langton observed that a cellular automaton could optimally perform computations, imitating complex life-like behaviour. Before this, the work of Packard [78] observed another unique property; that genetic algorithms tend to evolve populations in these critical regions, suggesting that adaptability was therefore optimised close to the edge of chaos. Similar conclusions on the significance of this region are proposed and demonstrated in neural networks, where vast computational power and capability in this region is described through network connectivity. Bertschinger and Natschläger [10] demonstrated these relationships in input-driven networks by accurately determining the position of the critical line with respect to structural parameters.

Moving away from abstract systems, it has also been suggested that *living* neural networks also support the "criticality hypothesis". Beggs [8] discusses this notion by looking at how the power-law distribution of neuronal avalanche sizes (a cascade of bursts of activity) suggests operation near a critical point. Beggs further explains, that the implications of these avalanche size distributions implies that information transmission, information storage, computational power and stability could all simultaneously optimise at the edge of chaos.

2.2 Configuration and structure

Traditional programs and algorithms represent idealised mathematical objects irrespective of their underlying hardware. In a physical system (say a biological system) computation is embodied, behaviour may not, or cannot, be completely captured by a closed mathematical model. As such, trying to alter these embodied systems requires different programming techniques. The difficulty involved in "programming" physical devices such as an exploitable medium is considerably harder. The programming and manipulation of materials requires (to some extent) a complex understanding of the properties and interactions within that system. We therefore require either some convoluted top-down "programming" approach (in the traditional sense) or an alternative mechanism (e.g. through training, learning or evolution). Whichever method is applied, this "program" would ideally alter the details of system in a controlled manner, most likely using controlling fields that affect structure and dynamics.

Structure plays a pivotal role in the complexity, behaviour and programmability of a material. The structure may ultimately define the computation and potentially adhere exclusively to the task. In a practical sense, a material would ideally require some form of self-assembly, or self-organisation, to create bottom-up structures on the molecular scale. Traditional top-down design is not only impractical where exact arrangements are not possible, but also typically implies some form of uniformity - which may be detrimental when large varying dynamics are wanted.

3 Evolution-in-materio

Evolution-*in-materio* (EIM) was first termed by Miller & Downing [69] as a means by which a physical system, i.e. a complex material, could be manipulated by computer controlled evolution (CCE) to perform useful computation. The idea of using unconstrained evolution as a search method in physical media is deep rooted in the field of Evolvable Hardware (EH) [31, 33]. Most evolved configurations in EH lead to digital products or components later embedded into physical artefacts. For example, evolving simulated models and optimisation programs, or, designing a physical system that can be manufactured after evolution; like antennas [60], robots [59, 81] or chemical systems made of oil droplets [32]. Natural evolution on the other hand, is a fully *embodied* (intrinsic) process where physical systems die, reproduce/self-replicate, co-evolve with the environment and other organisms, becoming many physical instantiations and tweaked copies of the thing being evolved.

Miller argues that Evolution-*in-materio* sits between full embodiment and the realised evolved EH solutions described above [70]. In this form, physical artefacts are configured (or conceptually created) during the process and assessed/controlled by simulated Darwinian evolution. This can also be seen in some EH systems, but, typically such systems are limited to constrained silicon hardware, e.g. electronic circuits evolved on Field Programmable Gate Arrays (FPGA) [39]. In this level of embodiment, EIM relies on a hybrid analogue/digital architecture where the evolutionary process (encapsulated on a digital computer) controls the writing/reading of physical signals to/from an analogue material. As such, the directed search tries to exploit the dynamics of the material by evaluating the performance of individual test configurations. Physical realisations are therefore embodied in the search process but evaluated externally. A system that operates in this manner is theoretically very powerful, allowing the manipulation of physical properties which are hitherto unknown.

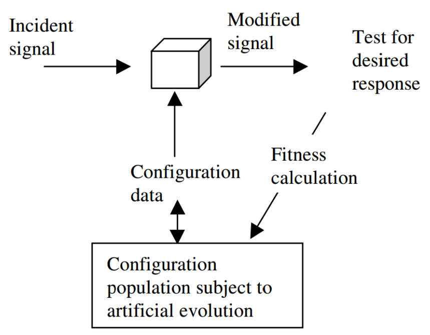
Incident signal
Modified signal
Test for desired response
Fitness calculation
Configuration data
Configuration population subject to artificial evolution

Fig. 1 Configurable Analogue Processor (CAP): using an evolutionary guided search the material is reconfigured to solve some computational task through applied input signals [69]

An early example of the EIM methodology can be found in Thompson's work with FPGAs [102]. Thompson attempted to evolve a frequency discriminator by allowing evolution to reconfigure circuit elements on the FPGA. In the process, he discovered that evolution in-fact used subtle electrical variations in the underlying material to form a solution. It was only made evident when evolved configurations no longer solved the problem when moved around the FPGA and when areas not directly involved somehow contributed to the overall operation. Thompson's work led to an explosion of interest, some of which can be seen in [33] and [70]. Miller & Downing labelled this work as a "starting point" for the exploration of intrinsic evolution as a means by which to exploit the natural properties of materials for computation [69].

3.1 Computation in Liquid Crystal

Miller & Downing discussed several interesting materials that could possess the desired characteristics needed for both computation and evolution. These include, liquid crystal, conducting and electro-active polymers, voltage controlled colloids, nanoparticle suspensions and irradiated or damaged semiconductors [69]. To exploit these materials they envisaged a device that could alter the materials function through configuration parameters (discrete signals), using what they referred to as a configurable analogue processor (CAP) (see Fig. 1).

Liquid Crystal (LC) was highlighted as the first candidate as it boasts a number of advantages for readily applying the theory. LC contains several key features including, wide availability, addressable using digital voltages, exhibits emergent behaviour, has a unique mesomorphic structure between ordered and disordered, and can relax to a initial base state.

Harding & Miller later adopted Liquid crystal as a basis material and constructed a bespoke platform to solve multiple computational problems [35]. The hardware used housed a liquid crystal display (LCD) and an array of dynamically selectable input/output connections to both the LCD and external measurement devices. Harding demonstrated liquid crystal as an efficient evolvable material where relatively small numbers of generations could produce effective solutions. Over the course of their investigation the LC system was applied to three separate tasks; tone discrimination [35], creating logic gates [37] and a real-time robot controller [36].

Harding & Miller found that a *rich* substrate of liquid crystal supplied many more exploitable properties compared to conventional silicon hardware (i.e. Thompson's FPGA). This in turn increased the diversity of solutions, thus increasing its evolvability. Harding & Miller's work demonstrated the advantages of emergent design by configuring intractable characteristic properties with no knowledge of their existence. But, the experiments also raised other fundamental questions on applying techniques from the Evolution-*in-materio* paradigm. For example, the length of time needed to "program" materials, i.e. how long does the search need to be and what constraints are there when transitioning into the physical world? what are the difficulties in defining the boundaries of the system under evolution, i.e. what is actually doing the computation? and what effects come with non-isolated systems embedded in a physical environment? is the system/evolution utilising environmental conditions and sources of noise? and what are the affects on system/solution reproducibility? what are the consequences of varying conditions on replicating the solution? Many of these questions and more are discussed in-depth in [70].

3.2 NASCENCE project: Carbon nanotube substrates

As part of the European FP7-ICT research project called NASCENCE[1] (NAnoSCale Engineering for Novel Computation using Evolution) new materials were later considered, along with a new bespoke hardware platform [13]. The present hardware iteration, known as the Mecobo board [65], forms another hybrid hardware architecture to integrate digital computers with experimental materials. The system interfaces with materials placed on micro-electrode arrays (MEA) using a similar premise to Harding & Miller's liquid crystal system; the CAP.

The substrates currently under analysis consist of Single-Walled Carbon Nanotubes (SWCNT) mixed with either a polymer or liquid crystal. Another substrate consisting of randomly-dispersed gold nanoparticles is also under investigation, but will be ignored here as it requires temperatures less than one kelvin to function. The polymer/LC mixtures disperse the nanotubes into random static networks, forming varying connection topologies and conductive pathways, possibly forming something akin to a random electrical circuit. Carbon nanotubes are used as they can exhibit either metallic or semi-conducting behaviour and contain other unique properties (e.g. ballistic conduction, thermal conductivity, self-assembly via Van der Waals

[1] Link to NASCENCE Homepage: http://nascence.no/

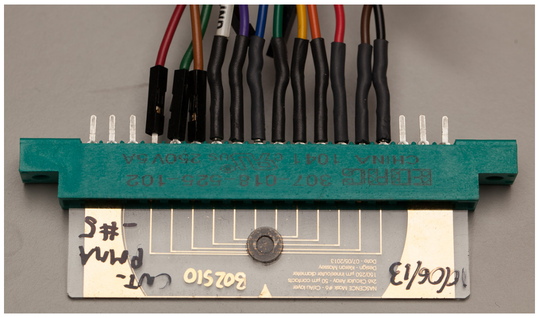

Fig. 2 Micro-electrode array used in the NASCENCE project to contain, stimulate and record activity in a carbon nanotube-based substrate. Computer controlled evolution is used to select active electrodes and mode (i.e. record or stimulate)

forces etc.), whilst the mixing material is said to create isolating regions, forming an insulator between elements and creating network structure.

A number of recent investigations have demonstrated the capabilities of SWCNT/ polymer mixtures, in particular the Poly-methyl metha-crylate (PMMA) substrate, as a potentially rich, evolvable and ubiquitous substrate. These investigations include, solving classification and optimisation problems such as frequency classification [72], classifying various data instances [21, 74], solving small-city cases of the travelling salesman problem [20] and applied to the (NP-hard) bin packing problem [73]. Early evidence highlights and supports the plausibility and potential of the methodology, but, in some respects it still lacks competitive results and again succumbs to some of the issues stated in §3.1.

A few features are worth noting for the above materials; PBMA appears to show greater stability than PMMA, the electrical percolation threshold occurs around a concentration of 1% (wrt. polymer weight) thus forming long-conductive pathways. After this, adding more nanotubes provides little computational advantages as a suitable network appears to already exist. Although, higher concentrations do demonstrate more non-linear current-voltage (I-V) behaviour in comparison. Less than 1%, the nanotube networks become fairly sparse and are argued to have reduced computational performance (and potentially more linear properties) [68].

Investigations into SWCNT/liquid crystal mixtures has shown some promise, for example, non-linear I-V behaviour appears to be more prominent. It has also been demonstrated that conductivity and orientation can be changed by an in-plane electric field. But, LC has been shown to experience a longer configuration time, interms of LC molecule and SWCNT alignment, due to LC molecules being smaller than SWCNT ribbons, and relaxation times. Other issues include; Long-term stability and reconfigurability, and the exact role of the liquid crystal in nanotube alignment [109].

New engineering technologies, adaptations in the search method (or fitness criteria), changes in hardware (e.g. no. of electrodes, different pitch sizes, etc.) and new materials could all have a significant effect on the field. The key components to the success of EIM lay in improvements to the fabrication of materials and the interfacing system used for stimulation and observation. This early work has also only highlighted one of many means of "programming" a material via evolution (i.e. through discrete voltage inputs). Other controlling fields, or even a combination of fields, could be utilised to manipulate/configure different materials - hopefully further separating the distinction between configuration and input signals. The ideal scenario for this field would be to pave the way for cheap, small, easily reconfigured and manufactured, multi- or single-purpose standalone computational devices.

4 Reservoir Computing

4.1 What is Reservoir Computing?

Reservoir Computing is the unification of three, individually conceived, methods for creating and training artificial recurrent neural networks (RNN); Echo State Networks (ESN) [40], Liquid State Machines (LSM) [67] and the Backpropagation-Decorrelation (BPDC) on-line learning rule [93].

To build some context we shall first discuss some basic concepts. A typical RNN model consists of a system of three key layers; an input layer, a hidden layer (the core network) and an output layer (see Fig. 3). The hidden layer contains processing elements (neurons) that are interconnected through weighted synapses (connection weights). The input and output layers are connected to the hidden layer, again through weighted strengths. Variations on the types of connectivity, e.g. feedback from the output to hidden layer or input layer to output layer, depends on the task and method. For simplicity lets imagine a simple input-to-hidden-to-output system encompassing a recurrent network in the hidden layer.

When driven by an input neurons activate, propagating information through the network to other neurons through varying connection strengths. The presence of cyclic connections can produce self-sustained activations, preserving a dynamic memory in the networks internal state. Networks such as this have been shown to be theoretically very powerful and are described as both (in some cases) Turing equivalent [49] and good universal approximators of dynamical systems [28]. But making the most of RNNs comes at a price, as they suffer from many training difficulties, such as the computational expense of updating large networks, bifurcation points and sometimes falling into inescapable local optima when using gradient-descent.

Reservoir Computing offers an alternative training technique, it reduces the computational cost and removes the problem of degenerative gradient information which leads to poor convergence. But, as we will discuss, the concepts of reservoirs go beyond traditional neural networks and encompass (to some extent) more general dynamical systems. Evidence of this can be seen in the following implementations; in electronic circuits [88, 85], a bucket of water [26], Gene Regulation Networks (GRN) of E.coli bacteria [48, 22], deoxyribozyme oscillators (referred to as "DNA

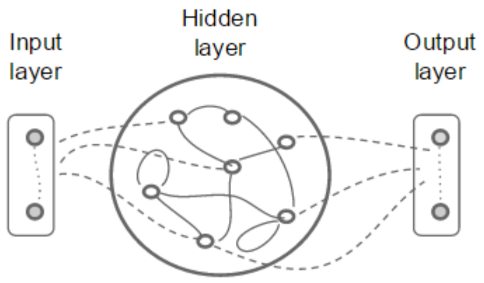

Fig. 3 A typical three-layer recurrent neural network. The input and output layer are connected to the hidden layer via weighted connections. The connections between neurons in the hidden layer are also weighted.

reservoir computing") [30] and a cats primary visual cortex [75]. Many of these systems are "black-box" systems in nature, holding the necessary and desirable dynamical properties to perform reservoir computation.

In this section we discuss some of the different types of reservoirs and some of the methods that have emerged from early RC experiments. We then delve into the relationship between *kernels* and reservoirs, and the importance of *criticality* in creating good reservoirs. Lastly, we give an overview of the functionality of reservoirs and some of the training and pre-training mechanisms available.

4.2 Reservoir types

There are many "flavours" of reservoirs originating from two separate research fields of machine learning and computational neuroscience. The first focusses on training dynamical systems for temporal learning tasks using artificial recurrent neural networks. Whereas the latter, aims at realistically modelling the computational properties of neural microcircuits. A short summary of the two main branches of RC will be mentioned here, including reservoirs in unconventional hardware. For more types and variations see [63].

The *Echo State Network* (ESN) is a discrete-time, neural network constructed from a sparse, random collection of analogue neurons. The typical neuron model employed uses the sum of its weighted inputs, applied to a sigmoid function (generally a hyperbolic-tangent), to give the neuron state $x(n)$. The state activations $x(n)$ of these neurons are termed as echo states [40], i.e. echoes of the input history. To propagate and hold this history the network is said to require the *echo state property*, or more generally speaking, a fading memory. The property itself is said to be provided by the characteristic dynamics of the system.

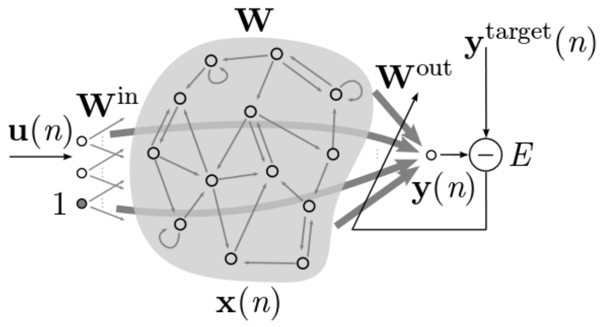

Fig. 4 Echo State Network: A random, static, recurrent network of sigmoidal neurons. The input driven system projects $u(n)$ into the reservoir network. Each node possesses a one-to-one weighted connection to all inputs (via W^{in}), outputs (via W^{out}) and other nodes (via W). The extra input (noted by 1) is provided as a bias W_{bias}. Training occurs on the output weights W^{out} (in relation to reservoir states $x(n)$) by reducing the error between $y(n)$ and $y^{target}(n)$. [61]

In ESNs, different scaling parameters, and in particular the *spectral radius* $\rho(.)$, influences these dynamics. These parameters fundamentally alter and control the amount of memory and non-linearity present in the system. The $\rho(.)$ parameter is used to scale the W matrix so that the largest absolute eigenvalue satisfies $\rho(W) < 1$ (typically, but not always. See [63]). Within this region the echo state property is said to be assured.

Another variant of the ESN model is the leaky-integrator neuron model. A neuron that possesses some form of memory of previous activations. These neurons contain a leaking rate, or decay parameter α, which can control the speed of the reservoirs update dynamics (see eq. (1)).

As Jaeger [40] describes, each neuron acts like a digital low-pass filter enabling a discrete network to approximate the dynamics of a continuous network (variations and uses can be seen in [40, 61, 63]. Dynamical systems have a natural time-scale, understanding the time-scale on which the input is changing compared to the time-scale of the system dynamics can be difficult. The leaky parameter α helps control and mediate any differences in input time-scales.

$$x(n) = (1-\alpha)x(n-1) + \alpha f(W_{in}u(n) + Wx(n-1)) \tag{1}$$

The *Liquid State Machine* (LSM) model came forth as a method for defining the computational properties and power of neural microcircuits, described as; “an alternative to Turing and attractor-based models in dynamical systems” [66]. The LSM model therefore represents a competitive model for describing computations in biological networks of neurons. The LSM attempts to model cortical micro-columns in the neocortex, structured in cortical layers of randomly created spiking neurons based on a spatial embedding. Among other things, it has been described as a possible process used by mammalian brains in speech recognition [25] and has been verified for cortical microcircuits in the primary visual cortex [75] and the primary auditory cortex [50].

Networks based on LSM use continuous streams of data (spike trains) to achieve real-time computations. Maass [66] has argued that classical models cannot handle real-time data streams based on spike trains. Unlike ESNs, they are generally more adaptive systems, supporting additional advanced readout features such as parallel perceptrons [67]. Although, in many cases a linear readout is often preferred.

Investigations using Liquid State Machines has highlighted the potential for more abstract unconventional applications. For example, pattern recognition using a physical medium (water) [26] and imitating LSMs in Escherichia Coli [48].

Unconventional Hardware: Single non-linear dynamical node with delayed feedback Recent experimental applications of reservoir computing in optoelectronics and photonics [5, 56, 79] have demonstrated a new way of constructing a pseudo-reservoir system. Using *delay systems* theory, a system can imitate the characteristics of a recurrent network without being one. Delay systems represent a class of dynamical systems which incorporate non-linear systems with some form of delayed feedback and/or delayed coupling.

The key feature of this new RC flavour is to replace a physical network of nodes (often large in size) with a single non-linear node and a delay line. The delay system mimics a large interconnected network by creating a topology of *virtual* nodes in the delay line. This is achieved by applying time-multiplexing techniques on the input, i.e. through a combination of sample-and-hold operations mixed with an additional input mask. The sample-and-hold operation creates a stream $I(t)$ which defines the state update determined by the delay τ in the feedback loop. An additional mask M is created to represent coupled weights between stream $I(t)$ and the virtual nodes. The matrix M was initially randomly created at the schemes fruition, motivated by the random connectivity in reservoirs, only later were optimised masks proposed in [6].

The number of virtual nodes in the system is defined by N equidistant points separated in time ($\theta = \tau/N$) along the delay interval τ. The resulting time-multiplexed input sequence becomes $J(t) = MxI(t)$ which is then fed into the non-linear node (see Fig. 5). Once the system has updated after time τ, the output nodes access the states in the delay line using $\sum_{i=1}^{N} w_i x(t - \frac{N}{\tau}(N-i)$. For more detail on the masking process go to the supplementary information provided for [5] and [6].

The single non-linear node scheme has many interesting implications for designing hardware reservoir computers. A clear advantage of the scheme is that the overall architecture needed is very simple, but, using a delay loop implies a serial process (in contrast to the RC parallel feeding of nodes). Therefore, the speed of information processing depends on the state update given by time τ. A few suggestions have been given to compensate for such discrepancies and to increase computational capability, e.g. adding additional delay lines (increasing memory capacity) and finding an optimal number of nodes that can reasonably be implemented physically - adding more nodes will decrease τ, but having too many nodes is physically impractical given the hardware involved. Future work using this technique looks

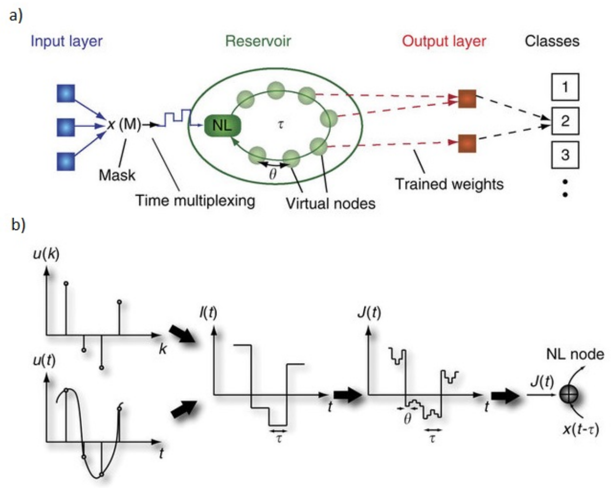

Fig. 5 a) Single non-linear node with delayed feedback. A new system derived from delay systems theory where virtual nodes are created in the delay loop τ to represent the hidden layer. The number of virtual nodes N is separated by time θ given by $\theta = \tau/N$, b) time-multiplexing of the input stream $I(t)$ (sampled from discrete $u(k)$ or continuous time $u(t)$) and a randomly created mask M that creates the input sequence $J(t)$. $J(t)$ is then added to the delayed state of the reservoir $x(t-\tau)$ before being fed back into the non-linear node [5]

very promising and could produce some ultra-high-speed computing solutions for specific tasks.

4.3 Reservoirs and kernels

A reservoir can be interpreted as possessing kernel-like properties. A *kernel* acts as a pre-processor, embedding input data into a vector space known as a *feature* space. It is understood in many statistical machine learning methods that combining this feature space with a simple linear discriminant algorithm can enable the learning of complex non-linear functions.

In kernel methods this is achieved by projecting the input space $u(n)$ into a high-dimensional (possibly infinite-dimensional) feature space $x(n)$ *without* paying the price of its explicit computation - referred to as the *kernel trick*. A kernel can therefore be expressed as the expansion function; $x(u(n))$. However, there are two significant differences between reservoirs and kernels; reservoirs ***do*** explicitly compute the input transformation, i.e. does not possess the kernel trick, and kernels are typically ill-equipped to handle temporal signals. To tackle temporal tasks the learned function $x(.)$ requires some form of memory of previous inputs. Reservoirs solve this by utilising the networks recurrent topology, which creates memory by retaining previous state activations. The final expansion function of the reservoir can therefore be represented as $x(n) = x(x(n-1), u(n))$.

Reservoirs use this pre-processing technique to map temporal features of the input into a spatially defined feature map (the network). The desired features can then be extracted, or combined, in a linear fashion to create the output $y(n)$:

$$y(n) = W^{out} x[u(n)] \tag{2}$$

where $W^{out} \in \mathbb{R}^{N_y \times N_x}$. This enables reservoirs to tackle many temporal and dynamic real-world tasks not possible using simple non-temporal kernels. Coincidently, 2 also implies the system contains a clear separation between the reservoir and the linear readout, separating the training procedure from the hidden layer, i.e. only W^{out} is trained. As such, the kernel representation offers a much faster and better converging mechanism compared to other RNN models, as it does not suffer from vanishing gradients. This representation also classes reservoirs as very powerful adaptive filters. For more information on kernels see [89].

To design an optimal reservoir one should find a good trade-off between 1) the transformation of the input that optimally boosts the linear classifiers capability - later referred to as the "*quality*" of the kernel - and 2) a sufficiently-long (fading) memory based on the input history. These two properties often conflict, to pertain a useful memory requires ordered behaviour and a rich transformation requires dynamic behaviour. Legenstein & Maass [57] have shown that optimal reservoirs tend to experience the best trade-offs at a critical point near the "edge of chaos".

4.4 Reservoirs and criticality

Dynamic networks are said to exhibit emergent criticality and self-organising properties [99]. A novel example of this can be seen [9], where a self-organising structure of carbon nanotubes evolves to produce maximum entropy given a strong applied electric field. Moreover, it has been observed that dynamic networks can self-organise into critical regions where they can perform interesting computations. These systems are characterised by motion in the phase space described as *trajectories* or state transitions. A trajectory may converge towards (i.e. be attracted to) a stable or unstable steady state; an attractor. Attractors vary from *point* - a point in the state space that attracts trajectories into its basin - to *strange* and chaotic - attracting trajectories, but inside diverge exponentially. These systems are very robust, small perturbations in the trajectory will tend to converge towards the same attractor. But, both external inputs and parameter changes in the system can drastically alter the shape of the phase space. Such changes may perturb a trajectory, moving or "clamping" the system between different attractor basins. As a result, clamping may even create, remove or change existing attractors and thus alter the initial phase space created by the natural dynamics of the system [97]. The overall dynamics of the system can therefore, to some extent, be controlled.

As was discussed in §2, certain critical regions have been identified to display interesting properties for computation (e.g. maximum complexity and performance). Although the reasons for maximised performance is not entirely understood, a series of quantitative measures have come forth to observe the effects of phase transition on computational capability.

Early measurements of dynamics in reservoir-like systems can be found in [10] where, using a similar technique to Derrida & Pomeau [24], dynamic behaviour is measured using the Hamming distances between two output states. By observing a growth in the distance between states it can be determined that chaotic behaviour is present, similarly, a decrease in distance would therefore indicate more ordered behaviour. This concept is similar to computing the characteristic Lyapunov Exponent for a dynamical system, with both measures analysing the sensitivity to differences in initial conditions.

Bertschinger & Natschläger highlight two fundamental properties required for these networks; a *fading memory* and a "network mediated" separation. A fading memory is indicative of an ordered phase with some dynamics, hence why it forgets. The same property is found in both Liquid State Machines and Echo State Networks, referred to as the "echo state property" in the latter. This allows the readout function to use information from recent inputs and derive functions of those inputs from the network state. The last property, network mediated separation, is deemed fundamentally important for input time-series networks, with similarities to the separation property in LSMs (see [67]). The property requires - ideally a large - diversity in network states to be the result of differences in inputs alone, i.e. characteristic features can be identified in the input and any changes in state should not directly be a result of chaotic dynamics which could produce the same phenomena. As such, this property enables a readout function to respond effectively to any variation in the inputs.

Legenstein & Maass [58] elaborate further on these properties, proposing two critical elements that characterise the computational capabilities of a complex dynamical system (cortical neural microcircuits in this case). These new measures, or properties, are proposed because Lyapunov Exponents are only useful for analysing a systems dynamics, not necessarily helpful in predicting good parameter regions that create high computational performance. These are referred to as the *kernel-quality* and the *generalisation-capability*. The kernel-quality refers similarly to the linear separation property found in kernels, as discussed in §4.3. An empirical measure of this property is achieved by examining the complexity of functions that can be carried out on the inputs that boost the classification power of a subsequent linear layer. The second element/property, simply quantifies a systems capability to generalise any learned behaviour to a new input.

Boedecker *et al.* [12] extends these ideas to ESNs, but, instead creates a general framework for direct and localised measurements for each neuron. Boedecker *et al.* further delve into measurements indicating the memory of each neuron and the transfer of information between each neuron. This work probably more importantly highlights some interesting and relevant points for all systems; a network does not necessarily need to be at the edge of chaos to do computation, the measured region where a system is at the edge of chaos is not universal for all tasks, and a critical state may maximise computational capabilities, but, such criticality may also be unnecessary or detrimental to certain tasks.

4.5 Training reservoirs

This section describes the training procedure and some of the available techniques one can use to train both the linear outputs and the reservoir itself. The two methods for training linear readouts described here are; "off-line" *batch-mode* training - using simple linear or ridge regression techniques (done once all reservoir states are collected into matrix X for training length T). And, "on-line" training - often gradient descent-based, Recursive Least Squares algorithm (a useful extensive investigation of RLS-type training is shown in [52]). The later subsections describe some *pre*-processing training techniques that can be used on the reservoir itself, methods that have been identified as useful in creating tailored/optimal reservoirs.

There are many training techniques available in this diverse field, to reduce any confusion we will just discuss methods perceived as "classical" training methods. For more examples of training techniques see [43, 61, 63].

Reservoirs are traditionally trained in a supervised manner where the temporal input $u(n)$ and coupled target output $y^{target}(n)$ are provided. Given a desired output the system can learn input-output behaviour by minimising the error (3) between system output and desired output. To evaluate if the learned behaviour generalises accordingly, new input data is tested and the error between the two are again compared.

$$E(y, y^{target}) = \sqrt{\frac{1}{T} \frac{\sum_{t=1}^{T} (y(n) - y^{target}(n))^2}{\sigma^2(y^{target}(n))}}, \tag{3}$$

Reservoir computers are conceptually viewed as recurrent neural networks incorporating the three-layered topology of N_x hidden nodes (neurons), N_u inputs and N_y outputs. As discussed in §4.2, variations on how these are implemented are also possible, but, essentially the system still adheres to the same structural layers.

The general update state equations for most systems are defined in (4) and (5) for, discrete time $n = 1, ..., T$, internal state $x(n)$ and output $y(n)$:

$$x(n) = f(W^{in}u(n) + Wx(n-1) + W^{fb}y(n-1) + W^{bias}), \tag{4}$$

$$y(n) = f^{out}(W^{out}[x(n); u(n)]), \tag{5}$$

The function f is commonly represented by a sigmoid, typically a hyperbolic tangent. In echo state networks this represents a basic $tanh$ neuron, but varies depending on the application. In other networks, f can be designed to form linear nodes, threshold logic gates or spiking neurons. In regards to the output $y(n)$, f^out may also be a non-linear function (sigmoid) but tends to be identity in most cases.

The weight matrices; input weights $W^{in} \in \mathbb{R}^{N_u \times N_x}$, reservoir/hidden layer weights $W \in \mathbb{R}^{N_x \times N_x}$ and feedback weights (from the output to the reservoir, if needed) $W^{fb} \in \mathbb{R}^{N_x \times N_y}$ are all drawn randomly from a uniform distribution at creation and remain static. The output weight matrix $W^{out} \in \mathbb{R}^{N_y \times (N_x + N_u)}$ includes weights for the inputs as they act as additional states (hence the concatenation of $[x(n); u(n)]$). Typically, the W matrix forms a sparse network with many of the weights set to zero, the other W^{in} and W^{fb} matrices can either be dense or sparse. Additional scaling parameters might also be applied to the matrices to govern properties such as non-linearity, stability and global dynamics. Techniques that can optimise/adapt each matrix (on-line or pre-processing) will be discussed later in the section.

The bias W^{bias} can be used to counteract training issues and large weights by adding noise, acting as a *regularization* parameter, or to push the *tanh* neuron to a particular state, creating a smoothing effect.

Applying feedback W^{fb} can be useful, or, detrimental to certain tasks. Some tasks might not be learnt to a reasonable degree without feedback, or, certain systems may require dynamics beyond what is supplied by the driven input to construct a suitable output. Adding feedback comes with its own risks, feedback will ultimately change the stability of the system and requires adaptations in the training procedure. It is often advised only to use feedbacks when necessary. For more information see [61].

The *off-line* technique is completed in one training cycle T after the system has computed all states for the given inputs. It provides a very fast training mechanism as it essentially computes a linear model given by the known output Y, collected reservoir states X and desired output Y^{target}:

$$Y = W^{out} X, \tag{6}$$

The collected state matrix $X \in \mathbb{R}^{N_x x T}$ is created when the input $u(n)$ is ran through the reservoir states $x(n)$. To avoid initial transients created by an initial zero state $x(0)$, a section at the beginning of the training sequence is discarded in the state matrix X. Essentially, the system goes through a "warming-up" process where states bounce around rather than returning to the states equilibrium output, i.e. the system is too chaotic to retrieve any useful information about the input.

Given (6), we can find the optimal weights that minimise the error between $y(n)$ and $y^{target}(n)$ by solving the overdetermined system:

$$Y^{target} = W^{out} X, \tag{7}$$

equation (7) can be solved for W^{out} using linear regression. The simplest method is to use Ordinary Least-Squares (8), but typically this method succumbs to stability issues when inverting (XX^T).

$$W^{out} = Y^{target} X^T (XX^T)^{-1}, \tag{8}$$

Lukoševičius [61] recommends using either ridge regression (regression with Tikhonov regularisation β) (9) or the Moore-Penrose pseudo-inverse (10). Ridge regression is a stable and effective solution and is generally advised. Adding a regularisation parameter counteracts the problems of producing very large output weights, which often indicates very sensitive and unstable solutions.

$$W^{out} = Y^{target} X^T (XX^T + \beta I)^{-1}, \tag{9}$$

where I is the identity matrix and β the regularisation parameter. Setting $\beta = 0$ gives the same method for solving linear regression in (8). It is therefore recommended to use a logarithmic scale for selecting β where it never reaches zero [63].

The pseudo-inverse is applied in some cases typically because of its ease to implement in certain programming environments (e.g. MATLAB), but comes at a price. The pseudo-inverse method is computationally expensive for large matrices of X and typically overdetermined. However, in most cases the network is made up of relatively small matrices and over-fitting depends on the difficulty of the task.

$$W^{out} = Y^{target} X^{+}, \tag{10}$$

4.5.1 On-line training

Some tasks require an on-line training method that adapts with time, minimising the error at each time step. This implicitly turns W^{out} into an adaptive linear combiner. The Recursive Least Squares (RLS) algorithm (11) is more commonly applied as it overcomes the severely impaired convergence performance of the Least Means Square (LMS) algorithm [42].

$$E(y, y^{target}, n) = \frac{1}{N_y} \sum_{i=1}^{N_y} \sum_{j=1}^{n} \lambda^{n-j} (y_i(j) - y_i^{target}(j))^2, \tag{11}$$

Using RLS comes at a cost, the number of weights is quadratic instead of linear and can still be numerically unstable in some cases. Other powerful on-line methods may be useful to a practitioner, particularly in the presence of feedback connections, such as Backpropagation-Decorrelation (BPDC) [93]).

For clarity, the RLS training procedure is described here, derived from [23]. First, set the error forgetting parameter λ close to but less than one - the forgetting parameter controls the contribution of previous samples. Next, initialise the autocorrelation matrix $\rho(0) = I/\delta$, with δ being a small constant and I the identity matrix. At each time step compute the output weights using the following steps:

Step 1: Calculate the reservoir state $x(n)$ and output signal $y(n)$ for input $u(n)$.
Step 2: Calculate the error between target output $y^{target}(n)$ and system output given previous output weights:

$$e(n) = y^{target}(n) - W^{out}(n-1)x(n), \tag{12}$$

Step 3: Update the gain vector $K(n)$:

$$K(n) = \frac{\rho(n-1)x(n)}{\lambda + x^T(n)\rho(n-1)x(n)}, \tag{13}$$

Step 4: Update the autocorrelation matrix $\rho(n)$:

$$\rho(n) = \frac{1}{\lambda}[\rho(n-1) - K(n)x^T(n)\rho(n-1)], \tag{14}$$

Step 5: Assign new output weights $W^{out}(n)$ using (12) and (13):

$$W^{out}(n) = W^{out}(n-1) + K(n)e(n), \tag{15}$$

Again, for more readout training methods including feedback training (such as FORCE training), supervised, unsupervised, reinforcement learning, etc. consult the excellent reviews and practical aids of [63] and [61].

4.5.2 Adaptation and pre-training

Adaptive reservoirs - that evolve with time - are inspired by natural adaptation in biological neurons. These adaptive processes are the result of persistent changes in a neurons electrical properties, governed by unsupervised *local* adaptation rules often referred to as *Intrinsic Plasticity* (IP). These rules represent a homeostatic mechanism in which neurons self-modify their intrinsic activity (i.e. excitability). Using such learning rules has shown to increase robustness and performance when pre-training reservoirs [86, 94]. For an overview of recent investigations including both local and global adaptation schemes consult [62] and [63].

In the context of this work, we are primarily concerned with the role of deterministic pre-training (although adaptation will undoubtedly play a role in the future). In classical RC, all reservoirs are generated randomly, it is therefore true to say that the performance of each reservoir will vary upon creation. As has already been stated, reservoir computing boasts its training performance on the separation between the reservoir and readout. The readout training, at its core, is quite inexpensive allowing the possibility of other forms of reservoir *pre-training*, i.e. generating reservoirs deterministically for each task. Even a crude experiment such as selecting a reservoir which produces the smallest error from a pool of randomly-created reservoirs highlights the advantages of pre-training. With this in mind, evolutionary algorithms propose themselves as an intuitive search strategy for pre-training.

Investigations using evolutionary optimisation for pre-training are again well documented. Many strategies have been attempted, including evolving topologies (i.e. network size), weight matrices (such as W^{in}, W, W^{fb}), global parameters (e.g. spectral radius), connection density, adapting slopes of the activation function $f(.)$ and even training w^{out} when no target signal is available. Other interesting methods include *EvoLino* - evolving hidden connections to gradient-based long-short term memory (LSTM) RNNs [82] and using *Neuro-Evolution of Augmented Topologies* (NEAT) as a meta-search algorithm for evolving ESNs [18] (the "related work" section also discusses other neuro-evolution methods for constructing ESNs). All of these methods have shown great potential, highlighting the performance increases and optimisations reservoir pre-training can create for specific tasks. Pre-training and adaptation appears to be one of many key branches under investigation in the field of reservoir computing.

5 Modelling materials with Reservoir Computing

Any high-dimensional dynamical system with an observable state $x(n)$, as a result of input $u(n)$, could form the basis of a reservoir according to [63]. This implies that any material that can exhibit sufficient dynamics and a fading memory could therefore, theoretically, be adopted as a reservoir.

Given that reservoir computing is based on artificial recurrent neural networks, the unambiguous route to follow would be to design hardware substrates modelled on simplified neural network-like structures, e.g. large coupled networks of non-linear elements with varying densities of connectivity. Using this structural model, semi-isolated regions of activity may form exploitable meta-states for the trainable readout. Various implementations of Hardware-based artificial neural networks have existed for some time; see [71] for a review of HNNs. More recently, there has been increased popularity towards applying memristive components to neuromorphic circuits, see [53] (a review CMOS/memristor hybrids) and [90].

Kulkarni & Teuscher [54], among others, have examined and demonstrated reservoir networks built from memristor devices. Memristors appear to be ideal components for building reservoirs, they exhibit both non-linear properties and maintain a memory of previous inputs. In the Kulkarni & Teuscher experiment, simulated circuits are randomly-created from networks of memristors, then a subsequent readout layer is trained using a genetic algorithm to solve some computational task. Other simulated memristor reservoirs include, simple-cycle reservoirs [14] i.e. memristive networks that form nodes instead of analogue neurons (see [80]), training more realistic volatile memristor models [17] and variation-tolerant reservoirs [15].

Actually fabricating, random, highly-interconnected networks and devices from nanoscale switching elements is a challenging task. In a practical sense, at these incredibly small scales, features of self-assembly and self-organisation are essential - characteristics that might only be achievable through unconventional methods and materials. Konkoli & Wendin [51] provide a brief review of some non-CMOS devices and discuss the suitability of the RC model on such unconventional devices. These devices include, molecular electronic networks, memristor networks and other substrates outlined as part of the SYMONE project [110]. At present, the SYMONE project is investigating networks of organic transistors (NOMFETs) and self-assembled networks of gold nanoparticles that could feature functionalised memristive junctions [51].

From what the authors are aware, only two examples have demonstrated that could classify as physical, *in-materio* reservoir computing systems; Silicon-based photonic chips (based on nanophotonic crystal cavities) [27, 105] and Atomic Switch Networks (ASNs) [92, 100, 101].

The *photonics chip* primarily exploits the propagation of light through silicon. Inside these chips exist photonic crystals that remove the propagation of certain frequencies of light - known as the band gap. Adding a line defect to a crystal produces a photonic crystal waveguide, effectively a process by which light is forced between the defect. Cavities are then created along the line defect to create an optical "resonator" which traps light, increasing the power inside the resonator. These resonators then form a optical reservoir which can be trained and manipulated using different types of readouts, e.g. [105] actually creates a linear system and the non-linearity is added at the readout (through the inherent non-linearity of the measuring equipment). Methods such as this propose an intriguing optical alternative to hardware-based reservoir computing.

The *Atomic switch network* approach focuses on the electrical and chemical properties of a material. These networks attempt to mimic the vast complexity, emergent dynamics and connectivity of the brain. Under this method, highly-interconnected networks are constructed by bottom-up self-assembly of silver nanowires. Through a triggered electro-chemical reaction, coated copper seed nucleation sites spawn large quantities of silver nanowires of various lengths (from nano- to millimetre scale). As a result, large random networks are formed creating crossbar-like junctions between nanowires, and when exposed to gaseous sulphur create $Ag|Ag_2S|Ag$ nanoscale metal-insulator-metal (MIM) junctions. Applying external activation (a bias voltage) to these junctions, a temporary resistance drop is created leading to functional memristive junctions called Atomic Switches. Applying this construction and functionalisation process the ASN method offers some unique properties, such as scalability and practicality in creating highly-complex nanoscale substrates.

The emergent behaviour and dynamics of ASNs can be observed through fluctuations in network conductivity, a result of spontaneous switching between discrete metastable resistance states, where locally excited regions cause cascading changes in resistance throughout the system. As such, the non-linear responses to resistive switching are reported to result in higher harmonic generation (HHG) - also suggested as a useful measure for quantitatively evaluating reservoir dynamics [92].

A clear advantage of the ASN technique is that it allows some degree of regulation in fabrication and further control through "resistance control" training [100]. For example, varying the parameters of the nucleation sites can alter the network structure and therefore the substrates dynamics. The relative size, numbers and pitch of copper seeds can determine the length of wires, forming distant connections or confining spatial regions to dendrite-like structures, thus ultimately defining the density of connections [92].

ASNs appear to be natural candidates for reservoir computing, producing a high-dimensional recurrent network that does not require the manipulation of individual elements. In the present literature, ASNs have only been applied to one derived RC task; the waveform generation task [92, 100]. This is a simple analogue task which measures a reservoirs ability to construct a desired output waveform from an independent input waveform using network generated harmonics. For example, can a trained reservoir given an input sine-wave, construct either a sawtooth, square-wave or any other periodic function at the output (essentially a Fourier Series task using harmonic analysis). These initial experiments have proven ASNs to be capable reservoirs and has also highlighted HHG as a potential metric for evaluating reservoir properties.

6 Evaluating the characteristics of reservoirs

Creating a random reservoir (in simulation at least) is fairly straightforward but designing one with the right properties, using the large parameter space available, is a difficult task. In many cases parameter selection is often done by hand, through trial and error, and with expert knowledge of the wanted characteristics. So how can we better understand and evaluate reservoirs? one idea is to simplify its construction. In doing so, one could provide a more theoretical understanding of what makes reservoirs useful/successful. [80] explores this idea by contemplating three issues; 1) what is the minimal complexity of topology and parameters that produce comparable performance to standard models? 2) what degree of randomness is needed to construct competitive reservoirs? and 3) how do completely deterministic reservoirs compare? These are good questions for understanding underlying RC principles, but may be impractical to investigate given an already created (maybe static) physical substrate. Instead, we desire more experimental quantitative measures that individually describe the reservoir and its qualities as an efficient kernel.

Determining, or evaluating, reservoir quality/performance can be achieved in two ways; either through direct measurement of performance on a given task, or, by accumulatively assessing individual properties of the reservoir. Coincidently, using the latter method provides a mechanism in which performance could be partially predicted for any task. As [76] explains, a good reservoir that scores well in all properties may be able to facilitate the process of “learning transfer”. This implies the reservoir itself can be trained to some objective function that will increase its capability without seeing any output task. The objective function, in the mentioned case, measures how well a reservoir (an LSM) separates different classes of inputs into different reservoir states. As such, it was shown that improving separability and instilling an adaptive balance between chaotic and ordered behaviour (through changes in structure) a reservoir can increase its performance across different artificial problems.

In this section we will discuss some of the proposed measures found in the reservoir computing literature, accompanied by a variety of different benchmark tasks used to assess performance. Such measures and benchmarks are suggested here as an effective method for evaluating the potential of substrate-based reservoirs.

6.1 Kernel quality and separation

6.1.1 Kernel quality

Kernel quality evaluates a reservoirs ability to produce diverse and complex mappings (functions) of the input stream u that can consequently increase the classification performance of a linear decision-hyperplane [57].

Kernel quality, also known as the *linear separation property*, was first introduced by Legenstein & Maass [57], along with an accompanying metric referred to as the *generalisation capability* of a reservoir. The two properties are closely coupled and can be measured using roughly a similar ranking mechanism. The first, kernel quality, measures a reservoirs ability to produce diverse reservoir states given significantly different input streams. The second, measures the reservoirs capability to generalise given similar input streams with respect to a target function. Both measurements can be carried out using the method stated in [16], by computing the rank r of the $n \times m$ matrix M, with the two methods only differing in the selection of m input streams $u_i, ..., u_m$, i.e. input streams being largely different or of similar type/class.

The rank is assessed as follows; Given the input stream u_i the reservoir state vector x_{u_i} of length n is collected to form each column in the Matrix M. The rank r of each matrix is then estimated by Singular Value Decomposition (SVD). Büsing *et al.* [16] explains that a good reservoir should possess a high kernel quality and a low generalisation rank, and also identifies an insightful correlation in the measurement to the reservoirs dynamics. For example, reservoirs in ordered and chaotic regimes produce opposite values in both measurements, i.e. ordered behaviour produces low ranking values in both and high values in a chaotic regime. This same connection has also been observed in [19] where the Lyapunov Exponent and the kernel quality strongly correlate.

6.1.2 Class separation

Class separation is a similar metric that corresponds directly to different classes of input stimuli. Demonstrations of class separation can be found in [19, 29, 76]. Separation is measured as the average distance between resulting states, once again, given the assumption that significantly different inputs should generate significantly different reservoir states. To calculate separation requires the division of the input and state vectors into discrete classes, although, [29] provides an alternative measure characterised on the original assumption. For example, given two different input vectors $u_j(n)$ and $u_{\hat{j}}(n)$ the euclidean distance between inputs should be large and positive, as described by D:

$$D = \|u_j(n) - u_{\hat{j}}(n)\|, \tag{16}$$

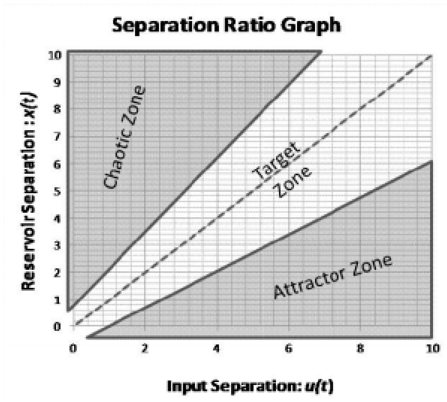

Fig. 6 Separation Ratio Graph [29]. Graphical representation of the phase transition between chaos and order. Systems in the target zone are said to possess both a good separation property and ideal dynamic behaviour to produce optimal reservoirs.

if the reservoir exhibits a good separation property the reservoir states ($x_j(n)$ and $x_{\hat{j}}(n)$) should increase in distance, or be equal to the original distance:

$$D \geq \|x_j(n) - x_{\hat{j}}(n)\|, \tag{17}$$

which can be represented as the ratio (18), provided a significant distance between the inputs:

$$\frac{\|x_j(n) - x_{\hat{j}}(n)\|}{\|u_j(n) - u_{\hat{j}}(n)\|} \geq 1, \tag{18}$$

This simplified measure has been extended by [29] into *Separation Ratio Graphs* to produce a visual representation of separation and the phase transition of correlated dynamic behaviour (see Fig. 6).

Konkoli & Wendin [51] offer another comparable method for identifying reservoir quality in memristor networks. This metric is again based on the assumption quality can be measured by observing the reservoirs ability to generate different dynamic states at the output. In this case, it is observed by measuring the dissimilarity between output states and a linear combination of the inputs, i.e. determining if the non-linear frequency response of a network cannot be approximated by a linear mixture of delayed inputs. Dissimilarity is measured in the Fourier space (w) between outputs $o(n)$ and a linear combination of the time shifted inputs $z(n)$, given by:

$$\delta = \frac{\|(o(w) - z(w))\|}{\|((o(w))\|} \tag{19}$$

A large dissimilarity (large δ) is ideal in a reservoir as it describes a complex projection of the input. A small δ on the other hand may simply describe a linear propagation of the input, highlighting the absence of richness in the reservoir.

6.2 Quantifying dynamics

Producing a reservoir with rich dynamics is evidently desirable but how can we quantify dynamical behaviour? how can we pin-point the critical region at the edge of chaos? In the literature, three to four prominent themes in classifying and measuring dynamics have arose, including observation of trajectories and attractor behaviour, internal scaling of input-driven activity, memory and retention, and higher-harmonic generation - particularly in reservoirs comprised of resistive switches.

6.2.1 *Spectral radius*

In Echo State Networks, the spectral radius $\rho(.)$ (the largest absolute eigenvalue) of the internal weight matrix W is used to determine and control reservoir dynamics [40]. The parameter ρ globally scales the internal weights moving the system between different regimes, altering the non-linearity and impulse response of the reservoir. Effectively, the scaling parameter alters the internal time-scales of the system, providing the *echo state property*.

Parameter selection of the spectral radius is typically centred around a value of one; smaller than one is attributed to a stable regime (a conducive fading response to input stimuli), if larger than one, a system will be typically unstable causing undesirable interference to new inputs.

6.2.2 *Lyapunov Exponents*

A popular measurement for criticality, or chaotic behaviour, in a reservoir is the empirical estimation of the Lyapunov Exponents (LE) for a dynamical system [10, 19, 29, 58, 106]. To calculate the Lyapunov Exponent, and quantify a systems phase, we measure the divergence between two close trajectories due to some small perturbation. For example, if an orbit is close to an attractor, in an ordered phase, small changes should dissipate over time. In a chaotic phase, an applied change will diverge exponentially from an orbit, persisting or increasing over time.

Gibbons [29] provides a simple approximation of the largest (Maximal) Lyapunov Exponent - as the largest tends to dominate. To do this, we measure the Euclidean distance between the perturbed states $d_j(n)$, where $x_{\hat{j}}(n)$ is the nearest neighbour to $x_j(n)$, then compute the resultant distance relative to the initial conditions $d_j(0)$:

$$\lambda(n) = g\sum_{k=1}^{N} \ln(\frac{d_j(n)}{d_j(0)}) = g\sum_{k=1}^{N} \ln(\frac{\|x_j(n)-x_{\hat{j}}(n)\|}{\|u_j(n)-u_{\hat{j}}(n)\|}) \qquad (20)$$

the initial state $d_j(0)$ is derived from [29] as a substitution of the distance between nearest neighbour inputs $u_j(n)$ and $u_{\hat{j}}(n)$. In equation (20), g is simply used as a gain parameter.

Various formats and interpretations exist in calculating different Lyapunov Exponents due to different approximation methods (see [47, 58, 106]). For example, Verstraeten *et al.* [106] examines the local Lyapunov spectrum and the Jacobian of the reservoir, suggesting performance can be better predicted by the maximum of the minimal Lyapunov Exponent. Boedecker *et al.* [12] takes a similar approach to Gibbons, estimating the mean exponential rate of divergence of the trajectories for an n-dimensional phase space. In this case, the maximal exponent (λ) is most prominent compared to the other n-Lyapunov characteristic exponents measured:

$$\lambda = \lim_{k \to \infty} \frac{1}{k} \ln\left(\frac{d_k}{d_0}\right) \tag{21}$$

In equation (21), and in most cases, a system with $\lambda \approx 0$ constitutes somewhere near the *edge of chaos*. A chaotic system is said to be present at $\lambda > 0$ and an ordered, or, sub-critical system at $\lambda < 0$.

Another suggested measure of criticality is the measure of instantaneous entropy of reservoir states, defined as the *average state entropy* (ASE) in [77]. Applying Rényi quadratic entropy, one can measure the distribution of instantaneous amplitudes in reservoir states, providing some measure of the "richness" of dynamics. The entropy measurement is associated with an expectation that increased diversity of amplitudes, i.e. increased spread of amplitudes away from some concentrated point, will increase the readouts ability construct the desired response.

6.2.3 *Memory Capacity*

Measuring the short-term memory capacity of a reservoir was first outlined by Jaeger [41] as a quantitative measurement to observe the echo state property (fading memory). To determine the memory capacity of a reservoir we simply measure how many delayed versions of the input $u(n-k)$ can the outputs recall or recover with precision. As Jaeger describes, using the equation in (22), we can measure memory capacity by how much variance of the delayed input can be recovered summed over all delays [41]. This is carried out by training individual output units to recall the input at time k with the maximum capacity MC of an N node reservoir typically bounded by its size, i.e. $MC \leq N$.

$$MC = \sum_{k=1}^{\infty} MC_k = \sum_{k=1}^{\infty} \frac{cov^2(u(n-k), y(n))}{\sigma^2(u(n))\sigma^2(y(n))} \tag{22}$$

This measurement has direct connotations to the dynamic behaviour of a system, i.e. it can be helpful in identifying the boundaries between static structure - which provides memory - and the point of complex dynamics that gives us processing. As such, one might expect a chaotic system to lose information regarding previous inputs at a faster rate and a more ordered regime to increase (to some extent) input retention.

6.2.4 *Harmonic Generation*

The generation of higher harmonics in Atomic Switch Networks has been identified as a technique for examining emergent behaviour and network connectivity [92, 100]. In ASNs, higher harmonic generation (HHG) is attributed to the non-linear frequency response of the system due to both input amplitude and memristive "hard" switching behaviour. To examine HHG, one can simply plot the frequency response of the system, which can be used to identify connectivity and system dynamics, i.e. changes in network response (an on-set of HHG) is typically due to an increase in "hard" switching memristive connections past a percolation threshold [92]).

6.3 *Evaluation through benchmark tasks*

As a means of direct assessment, we can measure performance of reservoirs and their subsequent readouts by applying them to specific tasks. Reservoir computing (and Neural networks as a whole) possesses an abundance of benchmark tasks, from simple classification and time-series prediction to robot navigation [4, 23] and non-linear channel equalization [79, 80].

In this section we will discuss benchmark tasks that are the most prevalent in the reservoir computing literature.

6.3.1 *Waveform generation*

A simple task that requires a rich transformation of a temporal input waveform (a periodic signal) to create an entirely new waveform. This task is based upon the Sillin *et al.* [92] physical adaptation of Jaeger's [43] sine-wave generator task and is linked directly to Fourier series/analysis. The task is to train the system to produce three different waveforms given an input sine-wave. In both [92] and [100], this is achieved by applying a 10 Hz input sine-wave (at one electrode in the ASN) to produce a 10 Hz square-wave, sawtooth and a 20 Hz sine-wave at the output ($y(n)$) - via the combination of other weighted electrode readings (e.g. recorded states $x(n)$). The task itself is said to be an excellent precursor to testing potential reservoir substrates on more difficult temporal problems [92], as it highlights an abundance of temporal features (e.g. phase shifts, delays, harmonic generation, recurrence etc.).

A similar task also widely investigated is the continuous-time multiple superimposed oscillator (MSO) task. In this benchmark, the reservoirs role is to predict the evolution of and generate a superposition of two or more sinusoidal waves with different frequencies.

$$s(t) = sin(w_1 t) + sin(w_2 t) \tag{23}$$

where $w_1 = 0.2$ and $w_2 = 0.311$. The task has been demonstrated in photonics experiments [27] and other non-traditional reservoirs [95].

6.3.2 *Time-series prediction and generation*

Two prominent benchmark tasks in reservoir computing are Non-linear Auto-regressive Moving Average (NARMA) dynamical system modelling and the Mackey-Glass chaotic time-series prediction task.

The NARMA task originates from Atiya & Parlos work on training recurrent networks [7] and its goal is to evaluate a reservoirs ability to model an n-th order, highly non-linear dynamical system where the system state depends upon the incoming input as well as its own history. The challenging aspect of the NARMA task is that it contains long-term dependencies created by the n-th order time-lag. Typically, the problem is carried out on 10-th and 30-th ordered systems in the literature [5, 42, 43, 94].

A description of the 10-th ordered task is as follows; Given white noise $u(n)$ from a uniform distribution of interval [0, 0.5], the reservoir should predict an output $y(n)$ close to the target $y(n+1)$, calculated by:

$$y(n+1) = 0.3y(n) + 0.05y(n)\left(\sum_{i=0}^{9} y(n-i)\right) + 1.5u(n-9)u(n) + 0.1 \tag{24}$$

Mackey-Glass chaotic time-series prediction is another common benchmark, where the system is trained to predict one time-step into the future (for examples see [40, 45, 106]).

The system is described by the Mackey-Glass delay differential equation:

$$\dot{y}(n) = \alpha y(n-\tau)/(1+y(n-\tau)^{\beta}) - \gamma y(n) \tag{25}$$

As [40] explains, parameters for the MG task are typically set to $\alpha = 0.2$, $\beta = 10$ and $\gamma = 0.1$ with the parameter τ set to 17 to produce a mildly chaotic attractor - a system is said to have a chaotic attractor if $\tau > 16.8$.

Other time-series prediction benchmarks are summarised in [80], including predicting laser activations in the Santa Fe Laser dataset (originally used in [46]), predicted next output in the Hénon Map dataset, IPIX Radar and Sunspot series datasets.

6.3.3 *Classification tasks*

Simple classification problems are wide and varied in the field of machine learning, some of which can be seen in both the RC and EIM literature. For example, typical tasks for EIM are tone and frequency discriminators and Iris & Lenses dataset classification [34, 72, 74]. Examples in reservoir computing include, signal classification (discriminating between two waveforms) [79], various n-bit parity problems [10, 23, 83, 87] and other time-independent classification tasks stated in [3].

Possibly the most adopted classification task in RC involves the recognition of isolated digits from multiple speakers [56, 79, 80, 84, 86, 107, 108]. Taken from a subset of the T146 speech corpus dataset, the task uses a total of 500 speech fragments collected from five different participants listing the numbers zero to nine (repeated ten times). The reservoir interprets these speech fragments through a pre-processing filter in the form of a digital model of the human cochlea, linear classifiers are then trained to be sensitive to individual digits with a final classification being made on the temporal mean of the output. Performance on the task is measured using cross-validation by calculating the number of misclassified digits using the Word Error Rate (WER).

Reservoir computing is said to be very competitive to, or outperform, many of the state-of-the-art approaches on these tasks.

6.4 Material properties and considerations

In order to perform material computation we require some ability to manipulate and control certain aspects of physical structure and behaviour. To observe such effects requires a means of observation and measurement. In the two examples presented (i.e. ASNs and EIM), we have discussed one method by which this can be achieved, through the application and recording of electrical voltages implanted on a micro-electrode array. There are of course many other possible methods. For example, optical stimulus/measurement and other stimuli across the electromagnetic spectrum (e.g. optoelectronic and photonic reservoirs), image recognition for observation (e.g. a method also used in Fernando & Sojakka's bucket of water), control and observation through magnetic fields (e.g. manipulating ionised gases or observing nuclear magnetic resonance (NMR) [96]), chemical excitation and reaction (e.g. reaction-diffusion computers and slime mould [1, 2]) and many other possible combinations.

There are many physical properties and considerations that require discussion when talking about using any novel material for computation. In order to narrow the discussion, we will focus on four key factors that possess some relevance to substrate-based reservoirs; 1) a means by which to observe network connectivity and activity, 2) assuring non-linearity is present in the system, 3) methods for modelling activity and behaviour, and 4) the impact of environmental factors.

6.4.1 *Network connectivity and activity*

The computational capability of a material is often directly related to conductivity and the density of connections inside it. Variations in these concentrations can have an adverse or favourable effect on conductivity and task performance. For example, to optimise an ASN one can control the densities of silver nanowires [92]. Massey *et al.* [68] also identifies a similar relationship, where nanotube concentration directly alters the conductivity and computational performance of SWCNT/PBMA composites. But, how can we measure connectivity in materials and analyse distributed

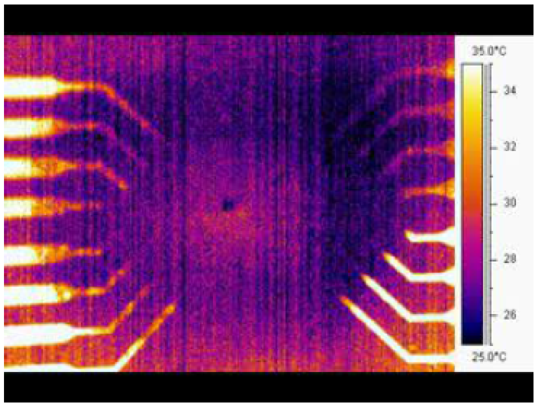

Fig. 7 Infrared image of carbon nanotube/polymer mixture. Infrared has been identified as a useful method for observing current flow and local regions of activity in both EIM and ASNs

activity? One possible method is demonstrated in [101], where ultra-sensitive infrared (IR) imaging is used help to identify network conductivity and behaviour in ASNs. IR imaging was used to observe and measure possible dominant conductive pathways and identify the activity of local regions to stimulus by thermal emission.

Other forms of network observation include, observation of power-law scaling in the power spectral density, microscopy and spectroscopy such as examining structure through an optical microscope and scanning electron microscopy, observing current flow by electron-beam-induced-current, and investigating structural properties using Fourier transform infrared and Raman spectroscopy (a method used to observe the interaction of molecules in liquid crystal/SWCNT composites in [109]).

6.4.2 *Non-linearity*

Non-linearity within a material can be measured through current-voltage (I-V) characteristics. In ASNs, non-linearity is observed by the presence of pinched hysteresis curves as a function of input amplitude (produced by applying slow voltage sweeps). This non-linear *I-V* characteristic is said to be the result of changing switch behaviour (in this case towards a "hard-switching" regime) and increased harmonic generation [92].

Similar *I-V* measurements have also been used in both SWCNT/PBMA and SWCNT/LC composites proposed in the NASCENCE project [68, 109].

6.4.3 *Modelling*

Simulating the internal properties of specific materials would be desirable, but typically impractical, creating an exact representation of one individual material would

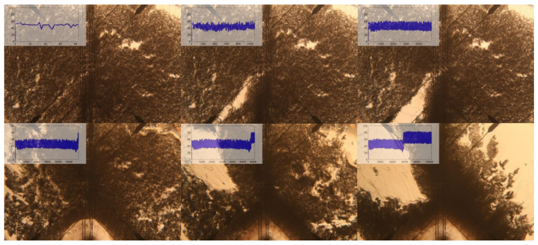

Fig. 8 Image of SWCNT/LC behaviour to voltage stimulus, observed through a microscope. Observation at this scale shows detail in the complexity of local interactions and carbon nanotube alignment to electric fields.

be a difficult task. Instead, we desire some abstraction, a model, of what the material is doing. Some methods are presented here, some to model specific features of the system and others as potential accompaniments to the reservoir model. For example, the modelling of electron transport using a Monte Carlo approach, percolation visualisation for electrode pathways, Voronoi diagrams for visualising electrode activity [91], Volterra series and Wiener series models, using Random Boolean Networks as a model, abstract neural networks for modelling global activity, NARMAX (non-linear autoregressive moving average model with exogenous inputs) modelling, and using Cellular Automata as a reservoir/substrate model [112, 113].

6.4.4 *Environment and noise*

The local environment, thermal noise, other noise related fluctuations and quantum effects are tangible concerns in the physical domain. Sensitive systems require good isolation and compensation techniques to reduce the effects of both internal and external noise. Effectively, the systems susceptibility to noise will determine its robustness and adaptability. In many cases, noise and unwanted variability could be filtered using conventional techniques, but requires some care in implementation (i.e. does not impede on the boundaries of the system or unduly affect what is doing the computation). Evolution-*in-materio* on the other hand has been found to exploit such external influences. Using an evolutionary search in physical systems has shown an uncanny ability to utilise strange characteristics that are clearly attributed to external influences. For example, Bird & Layzell's evolvable motherboard was directly influenced by local laboratory equipment in producing evolved oscillators [11].

7 Feedback design: Designing substrates to be good reservoirs

Beyond simply applying the reservoir model, could we utilise some of its highlighted features to aid in the design of *in*-materio systems?

Often the difficulty in designing and engineering computational substrates is not coherently or accurately defining/knowing what range of features are necessary, or even exploitable. The design process, in some respects, is fairly open and full of potential ”rabbit holes” and paradoxes. For example, if one were to use evolution to find computational exploits, we assume that some aspects of the physical system are unknown. Despite this, there are clearly some attributes that are desirable as a basis (for evolution at least), i.e. reconfigurable, input-driven, reproducible, some stability, etc. So the question can be rephrased; can we use known computational phenomena from reservoir computing to help reduce the possible search space of suitable materials? In essence, could the abstract model guide material design, and in return, serve to create a more realistic and efficient model. Therefore, creating a feedback-loop in design where new novel materials can be evaluated - potentially in fast manner - for computational capability and reservoir quality.

7.1 Multiple/Modular and Hierarchical reservoirs

Could multiple smaller reservoirs reproduce the overall dynamics of a larger network? As stated in [61], a powerful extension of ESN has come forth consisting of lots of small ESNs in parallel, where an averaged output has drastically improved performance, but, at a price in terms of memory capacity, i.e. less nodes typically equals less memory. Another approach would be to use multiple reservoirs to overcome hardware limitations, such as the number of electrodes available. These networked reservoirs could, as a by-product, provide increased robustness (further distributing computation) and added variability in states, i.e. promote a good overall separation property. Theoretically, such systems could be constructed from various materials each with different kernel properties and time-scales, allowing a global weighting system or training method (possibly backpropagation) to optimally choose which material to utilise for certain tasks. Moreover, this could lead to a more generic reservoir network suitable to multiple tasks.

Hierarchical reservoirs are said to represent an up-and-coming, possibly highly advantageous, avenue worth pursuing [63]. A hierarchical system attempts to overcome some of difficulties found in classical reservoirs such as scalability, learning complex intelligent tasks and working with multiple time-scales simultaneously. Some early examples of these architectures can be found in [44] where high-level reservoirs extract features from low-level ones using a “feature-voting” system. Another can be seen in a system comprised of decoupled sub-reservoirs with inhibitory connections [111] where the inhibitory connections predict the activation of sub-reservoirs. Other examples include, hierarchically clustered ESNs and the impact substructures have on stability [47], increased performance on speech recognition tasks [104] and acoustic modelling [103], and a hierarchical architecture used for hand-writing recognition to sort delivery parcels [64].

8 Conclusion

Reservoir computing can be interpreted as good stepping stone to modelling and training - to somewhat static in structure - *in-materio* systems. Reservoir computing is considered mostly on the basis that the systems themselves may not be able to undergo “on-line” training/adaptation (manually or autonomously). Thus, the material is perceived as an initially-configurable black-box, or a suitably rich kernel, removed from the final training process.

The reservoir method provides many advantages when dealing with unknown or intractable properties, but, it is also limited by the complexity of the task it is required to learn. For example, a good reservoir requires a high-dimensional expansion of the input that can be exploited by a subsequent readout layer. This implies that for more complex tasks the reservoir may require an exponential growth in exploitable features from an exceptionally large feature space, which itself is potentially impractical, or at least very difficult to extract useful features from. Harnessing these features then becomes a significant challenge when implemented in a physical device. Standard artificial neural networks try to overcome this problem by building the required non-linear features internally. This can only be achieved through internal training, which is traditionally achieved through backpropagation. To actually apply backpropagation in a physical system may be incredibly difficult, or near impossible, requiring a material to contain the right attributes for internal training and manipulation.

A recent example of a physical system that uses error-backpropagation as a training method is demonstrated in [38]. The experiment and implementation is carried out on both an acoustic system - using the propagation of sound waves between a speaker and microphone - and an electro-optical system - harnessing the reciprocal transmission of light through an optical circuit. As Such, this experiment poses a unique insight into backpropagation applied to physical systems and actually presents a potentially competitive alternative to digital neural networks.

The research areas discussed in this chapter, if combined, pose a very rich avenue to explore. Reservoir computing presents us with a convenient theoretical model that can be, primarily (of course others exist), trained through a subsequent readout layer. It also provides some indication of what properties are required to increase the performance of that layer, for example the right dynamic phase, good kernel projection, etc.

A recognised deficiency in the reservoir model is the undeniable fact that optimisation and pre-training can improve performance in reservoirs; a topic that requires some caution. For example, if applied, what then differentiates this method from other more traditional RNN training techniques? However, it is argued here that the pre-training formality separates it from the traditional, as it is a process by which a reservoir can be *pre-set* with some ideal properties to effectively increase the performance of the main training process.

As was discussed in §4.5, evolution as a form of pre-training is not new concept, but, in the physical domain it may provide a crucial and efficient technique for manipulating/configuring matter into suitable reservoirs, e.g. through structural alignment/deformation, or creating rich local regions of varying activity. As is described in the Evolution-*in-materio* doctrine, evolution may be the best practical technique to exploring and exploiting properties that are currently intractable or hitherto unknown, properties that could produce the most interesting and competent physical reservoirs.

Acknowledgements Matthew Dale is funded by a Defence Science and Technology Laboratory (DSTL) PhD studentship

References

1. A. Adamatzky. *Physarum machines: computers from slime mould*, volume 74. World Scientific, 2010.
2. A. Adamatzky, B. Costello, and T. Asai. *Reaction-diffusion computers*. Elsevier, 2005.
3. L. A. Alexandre, M. J. Embrechts, and J. Linton. Benchmarking reservoir computing on time-independent classification tasks. In *Neural Networks, 2009. IJCNN 2009. International Joint Conference on*, pages 89–93. IEEE, 2009.
4. E. A. Antonelo, B. Schrauwen, and J. Van Campenhout. Generative modeling of autonomous robots and their environments using reservoir computing. *Neural Processing Letters*, 26(3):233–249, 2007.
5. L. Appeltant, M. C. Soriano, G. Van der Sande, J. Danckaert, S. Massar, J. Dambre, B. Schrauwen, C. R. Mirasso, and I. Fischer. Information processing using a single dynamical node as complex system. *Nature communications*, 2:468, 2011.
6. L. Appeltant, G. Van der Sande, J. Danckaert, and I. Fischer. Constructing optimized binary masks for reservoir computing with delay systems. *Scientific reports*, 4, 2014. Article No. 3629.
7. A. F. Atiya and A. G. Parlos. New results on recurrent network training: unifying the algorithms and accelerating convergence. *Neural Networks, IEEE Transactions on*, 11(3):697–709, 2000.
8. J. M. Beggs. The criticality hypothesis: how local cortical networks might optimize information processing. *Philosophical Transactions of the Royal Society of London A: Mathematical, Physical and Engineering Sciences*, 366(1864):329–343, 2008.
9. A. Belkin, A. Hubler, and A. Bezryadin. Self-assembled wiggling nano-structures and the principle of maximum entropy production. *Scientific reports*, 5, 2015. Article No. 8323.
10. N. Bertschinger and T. Natschläger. Real-time computation at the edge of chaos in recurrent neural networks. *Neural computation*, 16(7):1413–1436, 2004.
11. J. Bird and P. Layzell. The evolved radio and its implications for modelling the evolution of novel sensors. In *Evolutionary Computation, 2002. CEC'02. Proceedings of the 2002 Congress on*, volume 2, pages 1836–1841. IEEE, 2002.
12. J. Boedecker, O. Obst, J. T. Lizier, N. M. Mayer, and M. Asada. Information processing in echo state networks at the edge of chaos. *Theory in Biosciences*, 131(3):205–213, 2012.
13. H. Broersma, F. Gomez, J. Miller, M. Petty, and G. Tufte. Nascence project: Nanoscale engineering for novel computation using evolution. *International journal of unconventional computing*, 8(4):313–317, 2012.
14. J. Bürger, A. Goudarzi, D. Stefanovic, and C. Teuscher. Composing a reservoir of memristive networks for real-time computing. *arXiv preprint arXiv:1504.02833*, 2015.
15. J. Burger and C. Teuscher. Variation-tolerant computing with memristive reservoirs. In *Nanoscale Architectures (NANOARCH), 2013 IEEE/ACM International Symposium on*, pages 1–6. IEEE, 2013.

16. L. Büsing, B. Schrauwen, and R. Legenstein. Connectivity, dynamics, and memory in reservoir computing with binary and analog neurons. *Neural computation*, 22(5):1272–1311, 2010.
17. J. P. Carbajal, J. Dambre, M. Hermans, and B. Schrauwen. Memristor models for machine learning. *Neural Computation*, 27(3):725–747, 2015.
18. K. C Chatzidimitriou and P. A. Mitkas. A neat way for evolving echo state networks. In *ECAI*, pages 909–914, 2010.
19. J. Chrol-Cannon and Y. Jin. On the correlation between reservoir metrics and performance for time series classification under the influence of synaptic plasticity. *PloS one*, 9(7):e101792, 2014.
20. K. D. Clegg, J. F. Miller, M. K. Massey, and M. Petty. Travelling salesman problem solved in materioby evolved carbon nanotube device. In *Parallel Problem Solving from Nature–PPSN XIII*, pages 692–701. Springer, 2014.
21. K. D. Clegg, J. F. Miller, M. K. Massey, and M. C. Petty. Practical issues for configuring carbon nanotube composite materials for computation. In *Evolvable Systems (ICES), 2014 IEEE International Conference on*, pages 61–68. IEEE, 2014.
22. X. Dai. Genetic regulatory systems modeled by recurrent neural network. In *Advances in Neural Networks-ISNN 2004*, pages 519–524. Springer, 2004.
23. S. Dasgupta, F. Wörgötter, and P. Manoonpong. Information theoretic self-organised adaptation in reservoirs for temporal memory tasks. In *Engineering Applications of Neural Networks*, pages 31–40. Springer, 2012.
24. B. Derrida and Y. Pomeau. Random networks of automata: a simple annealed approximation. *EPL (Europhysics Letters)*, 1(2):45, 1986.
25. P. F. Dominey. Complex sensory-motor sequence learning based on recurrent state representation and reinforcement learning. *Biological cybernetics*, 73(3):265–274, 1995.
26. C. Fernando and S. Sojakka. Pattern recognition in a bucket. In *Advances in artificial life*, pages 588–597. Springer, 2003.
27. Van Vaerenbergh T. Fiers, M., F. Wyffels, D. Verstraeten, Dambre J. Schrauwen, B., and P. Bienstman. Nanophotonic reservoir computing with photonic crystal cavities to generate periodic patterns. *Neural Networks and Learning Systems, IEEE Transactions on*, 25(2):344–355, 2014.
28. K. Funahashi and Y. Nakamura. Approximation of dynamical systems by continuous time recurrent neural networks. *Neural networks*, 6(6):801–806, 1993.
29. T. E. Gibbons. Unifying quality metrics for reservoir networks. In *Neural Networks (IJCNN), The 2010 International Joint Conference on*, pages 1–7. IEEE, 2010.
30. A. Goudarzi, M. R. Lakin, and D. Stefanovic. Dna reservoir computing: A novel molecular computing approach. In *DNA Computing and Molecular Programming*, pages 76–89. Springer, 2013.
31. G. W. Greenwood and A. M. Tyrrell. *Introduction to evolvable hardware: a practical guide for designing self-adaptive systems*, volume 5. John Wiley & Sons, 2006.
32. J. M. Gutierrez, T. Hinkley, J. Ward Taylor, K. Yanev, and L. Cronin. Evolution of oil droplets in a chemorobotic platform. *Nature communications*, 5, 2014.
33. P. C. Haddow and A. M. Tyrrell. Challenges of evolvable hardware: past, present and the path to a promising future. *Genetic Programming and Evolvable Machines*, 12(3):183–215, 2011.
34. S. Harding and J. F. Miller. Evolution in materio: A tone discriminator in liquid crystal. In *Evolutionary Computation, 2004. CEC2004. Congress on*, volume 2, pages 1800–1807. IEEE, 2004.
35. S. Harding and J. F. Miller. Evolution in materio: Initial experiments with liquid crystal. In *Evolvable Hardware, 2004. Proceedings. 2004 NASA/DoD Conference on*, pages 298–305. IEEE, 2004.
36. S. Harding and J. F. Miller. Evolution in materio: A real-time robot controller in liquid crystal. In *Evolvable Hardware, 2005. Proceedings. 2005 NASA/DoD Conference on*, pages 229–238. IEEE, 2005.

37. S. Harding and J. F. Miller. Evolution in materio: Evolving logic gates in liquid crystal. In *Proc. Eur. Conf. Artif. Life (ECAL 2005), Workshop on Unconventional Computing: From cellular automata to wetware*, pages 133–149. Beckington, UK, 2005.
38. M. Hermans, M. Burm, T. Van Vaerenbergh, J. Dambre, and P. Bienstman. Trainable hardware for dynamical computing using error backpropagation through physical media. *Nature communications*, 6, 2015.
39. T. Higuchi, M. Iwata, I. Kajitani, H. Yamada, B. Manderick, Y. Hirao, M. Murakawa, S. Yoshizawa, and T. Furuya. Evolvable hardware with genetic learning. In *Circuits and Systems, 1996. ISCAS'96., Connecting the World., 1996 IEEE International Symposium on*, volume 4, pages 29–32. IEEE, 1996.
40. H. Jaeger. The "echo state" approach to analysing and training recurrent neural networks-with an erratum note. *Bonn, Germany: German National Research Center for Information Technology GMD Technical Report*, 148:34, 2001.
41. H. Jaeger. *Short term memory in echo state networks*. GMD-Forschungszentrum Informationstechnik, 2001.
42. H. Jaeger. Adaptive nonlinear system identification with echo state networks. In *Advances in neural information processing systems*, pages 593–600, 2002.
43. H. Jaeger. *Tutorial on training recurrent neural networks, covering BPPT, RTRL, EKF and the "echo state network" approach*. GMD-Forschungszentrum Informationstechnik, 2002.
44. H. Jaeger. Discovering multiscale dynamical features with hierarchical echo state networks. *Technical report No. 9*, 2007.
45. H. Jaeger and H. Haas. Harnessing nonlinearity: Predicting chaotic systems and saving energy in wireless communication. *Science*, 304(5667):78–80, 2004.
46. H. Jaeger, M. Lukoševičius, D. Popovici, and U. Siewert. Optimization and applications of echo state networks with leaky-integrator neurons. *Neural Networks*, 20(3):335–352, 2007.
47. S. Jarvis, S. Rotter, and U. Egert. Extending stability through hierarchical clusters in echo state networks. *Frontiers in neuroinformatics*, 4, 2010.
48. B. Jones, D. Stekel, J. Rowe, and C. Fernando. Is there a liquid state machine in the bacterium escherichia coli? In *Artificial Life, 2007. ALIFE'07. IEEE Symposium on*, pages 187–191. IEEE, 2007.
49. J. Kilian and H. T. Siegelmann. The dynamic universality of sigmoidal neural networks. *Information and computation*, 128(1):48–56, 1996.
50. S. Klampfl, S.V. David, P. Yin, S.A. Shamma, and W. Maass. Integration of stimulus history in information conveyed by neurons in primary auditory cortex in response to tone sequences. In *39th Annual Conference of the Society for Neuroscience, Program*, volume 163, 2009.
51. Z. Konkoli and G. Wendin. On information processing with networks of nano-scale switching elements. *International Journal of Unconventional Computing*, 10(5-6):405–428, 2014.
52. A. U. Küçükemre. *Echo state networks for adaptive filtering*. PhD thesis, University of Applied Sciences, 2006.
53. D. Kudithipudi, C. Merkel, M. Soltiz, Garrett S R., and Robinson E P. Design of neuromorphic architectures with memristors. In *Network Science and Cybersecurity*, pages 93–103. Springer, 2014.
54. M. S. Kulkarni and C. Teuscher. Memristor-based reservoir computing. In *Nanoscale Architectures (NANOARCH), 2012 IEEE/ACM International Symposium on*, pages 226–232. IEEE, 2012.
55. C. G. Langton. Computation at the edge of chaos: phase transitions and emergent computation. *Physica D: Nonlinear Phenomena*, 42(1):12–37, 1990.
56. L. Larger, M. C. Soriano, D. Brunner, L. Appeltant, J. M. Gutiérrez, L. Pesquera, C. R. Mirasso, and I. Fischer. Photonic information processing beyond turing: an optoelectronic implementation of reservoir computing. *Optics express*, 20(3):3241–3249, 2012.
57. R. Legenstein and W. Maass. Edge of chaos and prediction of computational performance for neural circuit models. *Neural Networks*, 20(3):323–334, 2007.
58. R. Legenstein and W. Maass. What makes a dynamical system computationally powerful. *New directions in statistical signal processing: From systems to brain*, pages 127–154, 2007.

59. H. Lipson and J. B. Pollack. Automatic design and manufacture of robotic lifeforms. *Nature*, 406(6799):974–978, 2000.
60. J. D. Lohn, D. S. Linden, G. S Hornby, W. F. Kraus, and A. Rodriguez-Arroyo. Evolutionary design of an x-band antenna for nasa's space technology 5 mission. In *Evolvable Hardware, NASA/DoD Conference on*, pages 155–155. IEEE Computer Society, 2003.
61. M. Lukoševičius. A practical guide to applying echo state networks. In *Neural Networks: Tricks of the Trade*, pages 659–686. Springer, 2012.
62. M. Lukoševicius and H. Jaeger. Overview of reservoir recipes. Technical report, Tech. Rep. 11). Bremen: Jacobs University Bremen, 2007.
63. M. Lukoševičius and H. Jaeger. Reservoir computing approaches to recurrent neural network training. *Computer Science Review*, 3(3):127–149, 2009.
64. M. Lukoševičius, H. Jaeger, and B. Schrauwen. Reservoir computing trends. *KI-Künstliche Intelligenz*, 26(4):365–371, 2012.
65. 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*, pages 267–279. Springer, 2014.
66. W. Maass. Liquid state machines: motivation, theory, and applications. *Computability in context: computation and logic in the real world*, pages 275–296, 2010.
67. W. Maass, T. Natschläger, and H. Markram. Real-time computing without stable states: A new framework for neural computation based on perturbations. *Neural computation*, 14(11):2531–2560, 2002.
68. 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*, 117(13):134903, 2015.
69. J. F. Miller and K. Downing. Evolution in materio: Looking beyond the silicon box. In *Evolvable Hardware, 2002. Proceedings. NASA/DoD Conference on*, pages 167–176. IEEE, 2002.
70. J. F. Miller, S. Harding, and G. Tufte. Evolution-in-materio: evolving computation in materials. *Evolutionary Intelligence*, 7(1):49–67, 2014.
71. J. Misra and I. Saha. Artificial neural networks in hardware: A survey of two decades of progress. *Neurocomputing*, 74(1):239–255, 2010.
72. M. Mohid, J. F. Miller, S. Harding, G. Tufte, O. R. Lykkebo, M. K. Massey, and M. C. Petty. Evolution-in-materio: A frequency classifier using materials. In *Evolvable Systems (ICES), 2014 IEEE International Conference on*, pages 46–53. IEEE, 2014.
73. M. Mohid, J. F. Miller, S. Harding, G. Tufte, O. R. Lykkebo, M. K. Massey, and M. C. Petty. Evolution-in-materio: Solving bin packing problems using materials. In *Evolvable Systems (ICES), 2014 IEEE International Conference on*, pages 38–45. IEEE, 2014.
74. M. Mohid, J. F. Miller, S. Harding, G. Tufte, O. R. Lykkebø, M. K. Massey, and M. C. Petty. Evolution-in-materio: Solving machine learning classification problems using materials. In *Parallel Problem Solving from Nature–PPSN XIII*, pages 721–730. Springer, 2014.
75. D. Nikolić, S. Haeusler, W. Singer, and W. Maass. Temporal dynamics of information content carried by neurons in the primary visual cortex. In *Advances in neural information processing systems*, pages 1041–1048, 2006.
76. D. Norton and D. Ventura. Improving liquid state machines through iterative refinement of the reservoir. *Neurocomputing*, 73(16):2893–2904, 2010.
77. M. C. Ozturk, D. Xu, and José C Príncipe. Analysis and design of echo state networks. *Neural Computation*, 19(1):111–138, 2007.
78. N. H. Packard. *Adaptation toward the edge of chaos*. University of Illinois at Urbana-Champaign, Center for Complex Systems Research, 1988.
79. Y. Paquot, F. Duport, A. Smerieri, J. Dambre, B. Schrauwen, M. Haelterman, and S. Massar. Optoelectronic reservoir computing. *Scientific reports*, 2, 2012.
80. A. Rodan and P. Tino. Minimum complexity echo state network. *Neural Networks, IEEE Transactions on*, 22(1):131–144, 2011.

81. E. Samuelsen and K. Glette. Real-world reproduction of evolved robot morphologies: Automated categorization and evaluation. In *Applications of Evolutionary Computation*, pages 771–782. Springer, 2015.
82. J. Schmidhuber, D. Wierstra, M. Gagliolo, and F. Gomez. Training recurrent networks by evolino. *Neural computation*, 19(3):757–779, 2007.
83. B. Schrauwen, L. Büsing, and R. A. Legenstein. On computational power and the order-chaos phase transition in reservoir computing. In *Advances in Neural Information Processing Systems*, pages 1425–1432, 2008.
84. B. Schrauwen, J. Defour, D. Verstraeten, and J. Van Campenhout. The introduction of time-scales in reservoir computing, applied to isolated digits recognition. In *Artificial Neural Networks–ICANN 2007*, pages 471–479. Springer, 2007.
85. B. Schrauwen, M. DHaene, D. Verstraeten, and J. Van Campenhout. Compact hardware liquid state machines on fpga for real-time speech recognition. *Neural Networks*, 21(2):511–523, 2008.
86. B. Schrauwen, M. Wardermann, D. Verstraeten, J. J. Steil, and D. Stroobandt. Improving reservoirs using intrinsic plasticity. *Neurocomputing*, 71(7):1159–1171, 2008.
87. J. Schumacher, H. Toutounji, and G. Pipa. An analytical approach to single node delay-coupled reservoir computing. In *Artificial Neural Networks and Machine Learning–ICANN 2013*, pages 26–33. Springer, 2013.
88. F. Schürmann, K. Meier, and J. Schemmel. Edge of chaos computation in mixed-mode vlsi-a hard liquid. In *Advances in neural information processing systems*, pages 1201–1208, 2004.
89. J. Shawe-Taylor and N. Cristianini. *Kernel methods for pattern analysis*. Cambridge university press, 2004.
90. P. Sheridan, W. Ma, and W. Lu. Pattern recognition with memristor networks. In *Circuits and Systems (ISCAS), 2014 IEEE International Symposium on*, pages 1078–1081. IEEE, 2014.
91. H. O. Sillin. *Neuromorphic hardware: The investigation of atomic switch networks as complex physical systems*. PhD thesis, UNIVERSITY OF CALIFORNIA, LOS ANGELES, 2015.
92. H. O. Sillin, R. Aguilera, 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*, 24(38):384004, 2013.
93. J. J. Steil. Backpropagation-decorrelation: online recurrent learning with o (n) complexity. In *Neural Networks, 2004. Proceedings. 2004 IEEE International Joint Conference on*, volume 2, pages 843–848. IEEE, 2004.
94. J. J. Steil. Online reservoir adaptation by intrinsic plasticity for backpropagation–decorrelation and echo state learning. *Neural Networks*, 20(3):353–364, 2007.
95. J. J. Steil. Several ways to solve the mso problem. In *ESANN*, pages 489–494, 2007.
96. S. Stepney. The neglected pillar of material computation. *Physica D: Nonlinear Phenomena*, 237(9):1157–1164, 2008.
97. S. Stepney. Nonclassical computation: a dynamical systems perspective. In *in Handbook of Natural Computing*. Citeseer, 2011.
98. S. Stepney, S. Abramsky, A. Adamatzky, C. Johnson, and J. Timmis. Grand challenge 7: Journeys in non-classical computation. In *Visions of Computer Science, London, UK, September 2008*, pages 407–421. BCS, 2008.
99. S. Stepney, S. L. Braunstein, J. A. Clark, A. Tyrrell, A. Adamatzky, R. E. Smith, T. Addis, C. Johnson, J. Timmis, and P. Welch. Journeys in non-classical computation i: A grand challenge for computing research. *International Journal of Parallel, Emergent and Distributed Systems*, 20(1):5–19, 2005.
100. A. Z. Stieg, A. V. Avizienis, H. O. Sillin, R. Aguilera, H. Shieh, C. Martin-Olmos, E. J. Sandouk, M. Aono, and J. K. Gimzewski. Self-organization and emergence of dynamical structures in neuromorphic atomic switch networks. In *Memristor Networks*, pages 173–209. Springer, 2014.
101. A. Z. Stieg, A. V. Avizienis, H. O. Sillin, C. Martin-Olmos, M. Aono, and J. K. Gimzewski. Emergent criticality in complex turing b-type atomic switch networks. *Advanced Materials*, 24(2):286–293, 2012.

102. A. Thompson. An evolved circuit, intrinsic in silicon, entwined with physics. In *Evolvable Systems: From Biology to Hardware*, pages 390–405. Springer, 1997.
103. F. Triefenbach, A. Jalalvand, K. Demuynck, and J. Martens. Acoustic modeling with hierarchical reservoirs. *Audio, Speech, and Language Processing, IEEE Transactions on*, 21(11):2439–2450, 2013.
104. F. Triefenbach, A. Jalalvand, B. Schrauwen, and J. Martens. Phoneme recognition with large hierarchical reservoirs. In *Advances in neural information processing systems*, pages 2307–2315, 2010.
105. K. Vandoorne, P. Mechet, T. Van Vaerenbergh, M. Fiers, G. Morthier, D. Verstraeten, B. Schrauwen, J. Dambre, and P. Bienstman. Experimental demonstration of reservoir computing on a silicon photonics chip. *Nature communications*, 5, 2014.
106. D. Verstraeten and B. Schrauwen. On the quantification of dynamics in reservoir computing. In *Artificial Neural Networks–ICANN 2009*, pages 985–994. Springer, 2009.
107. D. Verstraeten, B. Schrauwen, M. dHaene, and D. Stroobandt. An experimental unification of reservoir computing methods. *Neural Networks*, 20(3):391–403, 2007.
108. D. Verstraeten, B. Schrauwen, D. Stroobandt, and J. Van Campenhout. Isolated word recognition with the liquid state machine: a case study. *Information Processing Letters*, 95(6):521–528, 2005.
109. 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*, 117(12):125303, 2015.
110. G. Wendin, D. Vuillaume, M. Calame, S. Yitzchaik, C. Gamrat, G. Cuniberti, and V. Beiu. Symone project: Synaptic molecular networks for bio-inspired information processing. *International Journal of Unconventional Computing*, 8(4):325–332, 2012.
111. Y. Xue, L. Yang, and S. Haykin. Decoupled echo state networks with lateral inhibition. *Neural Networks*, 20(3):365–376, 2007.
112. Ozgur Y. Reservoir computing using cellular automata. *CoRR*, abs/1410.0162, 2014.
113. Ozgur Y. Connectionist-symbolic machine intelligence using cellular automata based reservoir-hyperdimensional computing. *CoRR*, abs/1503.00851, 2015.