
Article:
Stepney, Susan orcid.org/0000-0003-3146-5401 (2024) Physical reservoir computing: a tutorial. Natural Computing. ISSN 1567-7818

https://doi.org/10.1007/s11047-024-09997-y

Natural Computing
https://doi.org/10.1007/s11047-024-09997-y

Physical reservoir computing: a tutorial

Susan Stepney[1]

Accepted: 28 September 2024

Abstract
This tutorial covers physical reservoir computing from a computer science perspective. It first defines what it means for a physical system to compute, rather than merely evolve under the laws of physics. It describes the underlying computational model, the Echo State Network (ESN), and also some variants designed to make physical implementation easier. It explains why the ESN model is particularly suitable for direct physical implementation. It then discusses the issues around choosing a suitable material substrate, and interfacing the inputs and outputs. It describes how to characterise a physical reservoir in terms of benchmark tasks, and task-independent measures. It covers optimising configuration parameters, exploring the space of potential configurations, and simulating the physical reservoir. It ends with a look at the future of physical reservoir computing as devices get more powerful, and are integrated into larger systems.

Keywords Reservoir computing · Physical computing · Echo State Network

1 Introduction

Artificial Neural Networks (NNs) for Machine Learning are pervasive. Feed forward NNs can be used for classification of complex data, and have a well-developed learning algorithm: back propagation. Recurrent Neural Networks (RNNs) gain memory through their recurrent connections, and are hence useful for time series prediction and classification. However, their standard training algorithm, back-propagation through time, is more computationally intensive.

Jaeger (2001) introduced an efficient training algorithm for random RNNs, one form[1] of reservoir computing (RC) called the Echo State Network (ESN). In an ESN the input and internal weights of the RNN are chosen randomly, and only the output weights are trained, in a one-shot non-iterative algorithm. This algorithm greatly increased the practicality of RNNs.

An RNN is a specific case of an open dynamical system, and the ESN training algorithm treats it as a black box, not changing its internal configuration. It was discovered that a physical dynamical system could replace the software RNN in its black box, and be exploited to perform physical reservoir computing.

1.1 Physical computing

All information is physical (Landauer 1991); all computing is physical. However, classical digital computing occurs at a very high level of abstraction, many virtual machine levels above the underlying physical material, and hence can be resource hungry. Physical computing refers to computing happening much closer to the 'metal': the implemented computational model more directly maps to the physical material, exploiting its natural physical properties more effectively.

This is how analogue computers work. The computer is designed to be a physical *analogue* of the problem being solved, typically by following the same differential equation. The computer is set to an initial state analogous to the problem's initial state, evolves under its own physical laws, and reaches a final state analogous to the problem's solution. The relevant computational model, covering all such computations, is the General Purpose Analog Computer

✉ Susan Stepney
susan.stepney@york.ac.uk

[1] Department of Computer Science, University of York, York YO10 5DD, UK

[1] For a discussion of other forms, and the origins of RC, see Jaeger (2021a).

Published online: 15 November 2024

(GPAC) (Shannon 1941), modelling the programmable Differential Analyser.

Physical Reservoir Computing (PRC) has a different computational model: that of RNNs. The aim is to find physical substrates that follow the model in a way that can be exploited for computation, and that enjoy the efficiency advantages of direct exploitation of dynamics. In addition, a physical reservoir does not need to be electronic: it can be made of many different kinds of material, that can be excited by many different modalities of input signal. This allows for a more natural coupling to environmental inputs.

Individual PRCs have been demonstrated in a wide range of materials, and performed successfully on small benchmark problems. However, they still face challenges of scaling up to more computational power, and discovery of real-world 'killer apps'. This tutorial aims to bring the development of PRCs to a wider computing audience, to allow further developments of this potentially powerful technology.

1.2 Tutorial overview

The structure of this tutorial is as follows. Section 2 defines what it means for a physical system to be a computer, rather than some other device. Section 3 discusses various computational models for PRC. Section 4 discusses choosing the material substrate that forms the core of the PRC. Sections 5 and 6 discuss input and output, including encoding, representation, and measurement. Section 7 discusses how to characterise a PRC, in terms of benchmarks and task-independent measures, and how to use these to find good configurations. Section 8 discusses uses of substrate simulation, and Sect. 9 discusses the future of PRC.

1.3 Further reading

This tutorial provides a computer scientist's view of PRC. In addition to this, there are many valuable resources about RC in general, and PRC in particular. Lukoševičius (2012) gives useful advice for configuring and training RCs. Stepney et al. (2018) has a collection of chapters on a variety of issues in physical computing in general; Nakajima and Fischer (2021) has a collection of chapters on specific PRCs. Nakajima (2020) provides an introduction to PRC. Tanaka et al. (2019) review the state of the art; Liang et al. (2024) review and evaluate recent PRC variants. Zolfagharinejad et al. (2024) provide a wider review of physical neuromorphic computing. Cucchi et al. (2022) is another PRC tutorial, from a more materials science and engineering perspective.

2 Physical computing and AR theory

Before we can describe physical *reservoir* computing, we need to address the question: when does a physical system compute? That is, when is it specifically and reliably performing a computation, as opposed to simply being a physical system? Attempts to answer this question have led to philosophical conundrums, from a rock that can compute any function (Chalmers 1996), to the whole universe being a computer (Lloyd 2005).

Such answers, whilst providing stimulating philosophical debates, are not much help to the practical engineer wishing to build useful devices. Abstraction/Representation theory (AR/T) (Horsman et al. 2014, 2017, 2018; Stepney and Kendon 2021) gives a definition of when a physical system is computing, as opposed to merely undergoing the physical processes of that system, or being experimented upon, that can be used to design and analyse unconventional physical computers.

AR theory distinguishes two kinds of spaces, an *abstract space* M of mathematico-logical and *computational* entities, and a *physical space* $\mathbf{P}$ of material objects and *computing* devices, along with a (necessarily informal) *representation relation* $\mathcal{R} : \mathbf{P} \to M$ that mediates between them. The reverse of the representation relation is the instantiation relation $\widetilde{\mathcal{R}} : M \to \mathbf{P}$, which holds when abstract entity $m_{\mathbf{p}}$ is instantiated as physical object $\mathbf{p}$. Whereas the representation relation is primitive, the instantiation relation is a derived relation, requiring characterisation of the physical system and its theories; see original references for full details.

Abstract computation takes abstract entities to abstract entities, $C : M \to M$. Physical evolution takes physical objects to physical objects, $\mathbf{H} : \mathbf{P} \to \mathbf{P}$. We link these

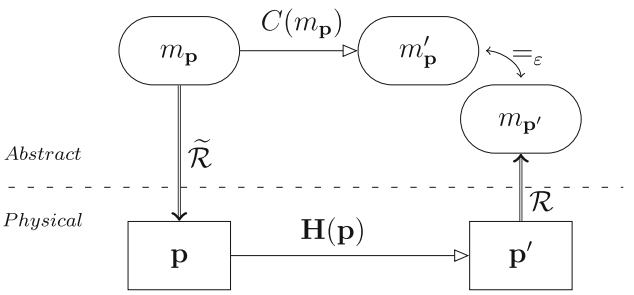

Fig. 1 Parallel evolution of an abstract entity and the physical object it represents: (top arrow) abstract computational dynamics $C(m_{\mathbf{p}})$ gives the resulting abstract state $m'_{\mathbf{p}}$; (down arrow) the abstract entity $m_{\mathbf{p}}$ is instantiated as physical object $\mathbf{p}$ through $\widetilde{\mathcal{R}}$: (bottom arrow) physical dynamics $\mathbf{H}(\mathbf{p})$ gives the resulting physical object $\mathbf{p}'$; (up arrow) $\mathcal{R}$ is used to represent $\mathbf{p}'$ as the abstract result $m_{\mathbf{p}'}$. If $m_{\mathbf{p}} =_\varepsilon m_{\mathbf{p}'}$, the diagram ε-commutes, and the physical device computes. (Adapted from Horsman et al. (2014).)

objects and entities through instantiation and representation (Fig. 1).

We now have two abstract objects: $m'_{\mathbf{p}}$, the result of the abstract computation, and $m_{\mathbf{p}'}$, the representation of the result of the physical evolution. For some (problem-dependent) error quantity ε and distance function $d()$, if $d(m_{\mathbf{p}'}, m'_{\mathbf{p}}) \leq \varepsilon$ (or, more briefly, $m_{\mathbf{p}'} =_{\varepsilon} m'_{\mathbf{p}}$), then we say that the diagram in Fig. 1 *ε-commutes*.

We may need to perform substantial experiments on, or theoretical analyses of, a system, both to determine the instantiation relation $\widetilde{\mathcal{R}}$, and to characterise the domain in which the diagram ε-commutes. But, given a well-characterised ε-commuting diagram over a known domain, we have a faithful abstract representation of the physical system, and we can use that physical system to mirror the abstract computation: we can use it to compute. The final abstract state can be known *either* by computation of the abstract representation of the system, $m_{\mathbf{p}} \mapsto m'_{\mathbf{p}}$, *or* by instantiating the abstract state in a physical device $\mathbf{p}$, allowing the physical evolution, $\mathbf{p} \mapsto \mathbf{p}'$, and then representing the resulting object abstractly as $m_{\mathbf{p}'}$. Given a well-characterised system, the results differ by less than the problem-dependent ε. In this case, the physical evolution is indeed *computing*.

Many devices claimed to be performing computation (either seriously, or for the point of philosophical debate) fail to meet the AR/T requirements because they lack any form of representation, the link between the abstract model and physical device (for example, the universe-as-computer case). Others fail because they need access to another computer to ensure that the diagram ε-commutes, typically by performing the entire desired computation *post hoc*, hidden in the output representation stage (for example, the rock-as-computer case). Yet others fail because the system is not well-characterised, and physical results that coincidentally match abstract ones are cherry-picked, with the others discarded.

The AR/T model allows for multiple substrates physically computing according to the same computational model (as is the case in multiple realisations of classical computing), and for a given substrate physically computing according to a variety of computational models (as in the case of classical computers implementing unconventional models). Here we focus on a single computational model, reservoir computing, potentially realised in a variety of substrates.

AR/T can be used as a framework to encompass the complete design and analysis of a physical computing system. There are three typical starting points:

- Substrate first, where the properties and behaviours of a given substrate are examined in detail, and a corresponding computation model is developed that has analogous (although more abstract) properties. For example, early neural network computational models were derived as highly abstracted models of brain and other neural functions; the Turing machine model was derived as a highly abstracted model of how 'human computers' perform calculations (Turing 1937).
- Computational model first, where a model is taken as given, and a substrate is searched for, or engineered, to conform to that model. For example, classical digital computers are engineered to conform to the von Neumann stored-program computer architecture; neuromorphic computers are engineered to conform to the neural model.
- Co-design (Stepney 2019), where model and substrate are iteratively designed and engineered together.

The topic of this tutorial, PRC, suggests a computational model first description. We describe PRC in terms of the AR/T framework according to the components of Fig. 1: the top arrow computational model (Sect. 3, reservoir computing), the bottom arrow physical substrate (Sect. 4, the device), the down arrow input (Sect. 5) and up arrow output (Sect. 6), and the requirement that the diagram ε-commutes (Sect. 7, characterisation).

3 Choosing a computational model

Physical computing is done with respect to a given computational model or programming paradigm. This defines the underlying form of the computation shown as the top arrow in Fig. 1. The computational model gives a semantics to the expression of the desired computation, allowing the derivation of the result. The chosen substrate must be able to implement this model on an appropriate domain.

3.1 Examples of computational models

There are many computational models available, such as classical Turing models, boolean logic and circuits, spin ices and cellular automata, artificial neural networks, a variety of quantum models, and more. A computational model does not have to be universal for it to be interesting to implement in a physical system. Apart from some quantum models, all these models can be implemented, or simulated, efficiently in a classical computer.

Here the focus is on the Echo State Network model of reservoir computing.

3.2 The Echo State Network model

The Echo State Network (ESN) model, introduced by Jaeger (2001), is a discrete time random Recurrent Neural Network (RNN) with a specific training algorithm. The recurrency gives RNNs memory, making them suitable for various time series analysis tasks, but they are hard to train using conventional techniques, which require computationally-intensive back-propagation through time. The ESN approach overcomes this by using an untrained random network (the 'reservoir'), with a fast training algorithm of the linear output layer weights only (Sect. 3.2.4).

This 'black box' approach to training ESNs, with no need to access or modify internal state parameters, makes them an interesting target for physical computing, using some material instantiation of the reservoir model.

In this section we review the basic ESN model.

3.2.1 The basic equations

The state update and observation equations are

$$\mathbf{x}(t+1) = f(\rho \mathbf{W}\mathbf{x}(t) + \mathbf{W}_u\,\mathbf{u}(t)) \tag{1}$$

$$\mathbf{v}(t+1) = \mathbf{W}_v\,\mathbf{x}(t) \tag{2}$$

where $\mathbf{u}(t), \mathbf{x}(t), \mathbf{v}(t)$ are the N_u-d input, N-d state, and N_v-d output vectors at time t; $\mathbf{W}_u$ is the $N \times N_u$ random input weight matrix; $\mathbf{W}$ is the $N \times N$ random state weight matrix, normalised to have a maximum absolute eigenvalue of one; ρ is the spectral radius modulating that normalisation; f is a non-linear thresholding function, typically $\tanh(.)$; $\mathbf{W}_v$ is the task-dependent trained $N_v \times N$ output weight matrix. Some models also allow the input to be a direct parameter in the output calculation.

Other authors use minor variants of this basic reservoir equation, such as $\mathbf{u}(t+1)$ in Eqn. 1, and/or $\mathbf{v}(t)$ in Eqn. 2. The form here is discussed in Stepney (2021), which considers the time taken to process inputs and outputs through their respective weight matrices.

3.2.2 The input terms

The N_u-d input vector $\mathbf{u}$ represents the input to the reservoir. In many examples and benchmarks in the literature, the input is scalar-valued, but it can be a vector representing multiple input channels. The input vector may be augmented with a constant bias term as an additional dimension; if so, even if the external input is scalar-valued, the resulting input is a vector with $N_u = 2$.

The $N \times N_u$ input weight matrix $\mathbf{W}_u$ is used to distribute the input across the state of the reservoir. In the ESN model, the weights of the input matrix $\mathbf{W}_u$ are chosen randomly, typically with a uniform distribution $W_{ij} \in U[-1, 1]$. They should then be scaled, or equivalently the input values should be scaled, such that the input has a sufficiently large effect to perturb the reservoir dynamics, but not so much that it saturates the system. If different input channels have different ranges, they may need to be scaled separately. The input scaling may need to be done in concert with the spectral radius ρ scaling of the state weight matrix (see Sect. 3.2.3).

3.2.3 The state terms

The N-d state vector $\mathbf{x}$ represents the states of the N nodes in the RNN. The $N \times N$ weight matrix $\mathbf{W}$ represents the weighted connections between the nodes.

In the ESN model, the weights of the internal state matrix $\mathbf{W}$ are also chosen randomly, typically with a uniform distribution $W_{ij} \in U[-1, 1]$. There may also be a density parameter, controlling how many (randomly selected) weights are non-zero. A density of one corresponds to a fully connected reservoir, lower densities correspond to more sparsely connected reservoirs.

Once the random state weight matrix has been formed, it is normalised to ensure the *echo state property* (ESP). The ESP or *fading memory property* ensures that the reservoir's state at large times is not dependent on details at early times (Jaeger 2001): there is no sensitive dependence on initial conditions. A sufficient condition to ensure the ESP is to normalise $\mathbf{W}$ such that its largest singular value, $\max(\sigma_i)$, is one. However, Jaeger (2001) notes that this sufficient condition is overly restrictive, and normalising $\mathbf{W}$ such that its largest absolute eigenvalue, $\max |\lambda_i|$ (that is, its spectral radius), is one is adequate to ensure the ESP in most cases. Hence normalising by the maximum absolute eigenvalue is traditional in the literature.

Different spectral radii are more appropriate for different tasks: the weight matrix is normalised, and the spectral radius parameter ρ applied explicitly. Jaeger (2007) notes that, although weight matrices with spectral radius greater than one do not have the ESP in isolation, they can still have the ESP if their input is large enough, and that "the larger the input amplitude, the further above unity the spectral radius may be while still obtaining the ESP". Hence the chosen value of ρ can affect the appropriate input scaling (Sect. 3.2.2).

The random weights provide a random projection of the input into a high dimensional space, such that the desired separation of different classes can be achieved with the trained linear output transformation $\mathbf{W}_v$. However, some projections are better than others, in a task dependent manner. Hence, it is often the case that a sampling over multiple random input and state weight matrices is performed, and the best performing one(s) selected for use. In some cases, a well-performing set of weights may even be searched for using, for example, an evolutionary algorithm (see Sect. 7.4). If the weights are optimised for a particular class of tasks, rather

than a specific task, then it is still the case that training for the specific task is fast, even if the pre-optimisation is not.

3.2.4 Training the output weights

The N_v-d output vector $\mathbf{v}$ represents the trained output from the RNN. In many examples and benchmarks in the literature, the output is scalar-valued.

The $N_v \times N$ trained output weight matrix $\mathbf{W}_v$ is used to project the high dimensional reservoir state into the output value.

Reservoir training is supervised training. Given a reservoir and a stream of length S of inputs $\mathbf{u}(t), t \in 1..S$ with a corresponding stream of target outputs $\hat{\mathbf{v}}(t), t \in 1..S$, the aim of training is to find the output weight matrix $\mathbf{W}_v$ that minimises the mean square error (Sect. 7.1) between the observed outputs $\mathbf{v}(t)$ and the target outputs. This training can be thought of as 'programming' the random ESN to perform a particular task, whereas adjusting the internal parameters (Sect. 3.2.3) to values suitable for a particular task is a form of overall reservoir configuration.

There are two main ways to calculate $\mathbf{W}_v$ (Lukoševičius, 2012, §27.4), shown in Algorithm 1. The first is a direct calculation using the Moore-Penrose pseudoinverse of the $N \times S$ matrix formed from S observations of the state during training; this is accurate but can be computationally expensive for large N and S, and requires $S \gg N$ to avoid overfitting. The second uses ridge regression with a regularisation parameter β; a larger β can be used to reduce the size of the components of $\mathbf{W}_v$ at the expense of a larger error. The value for β is task and data dependent, but it can be found relatively efficiently without the need to run multiple experiments: simply evaluate $\mathbf{W}_v$ over a range of values of β in Alg.1, line 13. In addition to discussion of these algorithms, Lukoševičius (2012) has many excellent tips for training reservoirs effectively, some of which are specific to ESNs, some of which are also relevant to PRCs.

Algorithm 1 training (Lukoševičius, 2012, §27.4)

1: $S :=$ length of training input $\gg N$
2: run reservoir with washout stream
3: **for** $t \in 1..S$ **do**
4: step reservoir with input $\mathbf{u}(t)$
5: $\mathbf{x}(t) :=$ reservoir state at time t
6: **end for**
7: $\mathbf{X} := [\mathbf{x}(1)\,\mathbf{x}(2)\,\ldots\,\mathbf{x}(S)]$ ▷ matrix of states
8: $\hat{\mathbf{V}} := [\hat{\mathbf{v}}(1)\,\hat{\mathbf{v}}(2)\,\ldots\,\hat{\mathbf{v}}(S)]$
9: **if** pseudoinverse **then**
10: $\mathbf{W}_v := \hat{\mathbf{V}}\mathbf{X}^+$
11: **else** ridge regression
12: $\beta :=$ regularisation parameter
13: $\mathbf{W}_v = \hat{\mathbf{V}}\mathbf{X}^\mathsf{T}(\mathbf{X}\mathbf{X}^\mathsf{T} + \beta\mathbf{I}_N)^{-1}$
14: **end if**

Once the reservoir has been trained, it should be evaluated on independent test data. See Sect. 7.1 for definitions of measures used to report the performance. This evaluation provides the evidence that the system works sufficiently well for the required computation: it provides a measure of ε in the ε-commuting square (Fig. 1).

The key point of this training algorithm is that it does not iteratively update the values of internal weights. RNNs are expensive to train with iterative back-propagation, because the recurrence needs unwinding with back-propagation through time. Jaeger (2001)'s key insight was that a single-shot learning of the readout weights is sufficient. Extreme Learning machines (Huang et al. 2006) use a similar single-shot learning approach, in the context of feedforward NNs.

It is this training approach, treating the reservoir as an unmodified black-box, that makes this model particularly suitable for physical computing: the ESN black box may be replaced by a suitable physical device. There is no need to modify internal parameters, and no need for a differentiable model of the device.

One note of caution: Jaeger et al. (2007, Fig. 4) demonstrate that a good prediction may not hold indefinitely; in the case demonstrated, after many *thousands* of timesteps, the prediction diverges. This is typically many times longer than reservoirs are trained or tested, but becomes an issue if they are to be deployed.

3.3 Augmenting the basic ESN model

In this section we review some standard ways to augment the basic ESN model.

3.3.1 Leaky nodes

The basic model, Eqn.1, is often augmented with a leak term, which allows some of a node's current state to 'leak' into its next state:

$$\mathbf{x}(t+1) = (1-\alpha)\mathbf{x}(t) + \alpha f(\rho\mathbf{W}\mathbf{x}(t) + \mathbf{W}_u\mathbf{u}(t)) \tag{3}$$

where $\alpha \in [0, 1]$ is the leak parameter. This parameter can be used to provide an extra timescale, in addition to the discrete 'clock tick', to the model.

3.3.2 Feedback

The basic model can be augmented with a feedback term:

$$\mathbf{x}(t+1) = f(\rho\mathbf{W}\mathbf{x}(t) + \mathbf{W}_u\mathbf{u}(t) + \mathbf{W}_{fb}\mathbf{v}(t)) \tag{4}$$

where $\mathbf{W}_{fb}$ is a $N \times N_v$ random untrained feedback weight matrix that feeds back the trained output into the reservoir state.

A reservoir driven by only output feedback, rather than external input, is called *autonomous*. Caluwaerts et al. (2013) use this approach to generate autonomous gait patterns in a tensegrity system.

3.3.3 Different neuronal topologies

The classic ESN has a randomly connected topology: $\mathbf{W}$ is random. Different neuron connection topologies, for example, ring topologies (Rodan and Tiňo 2011) and lattice topologies (Dale et al. 2021b) also show good reservoir performance. This demonstrates that the requirements for RC do not depend sensitively on the precise form of the internal weight matrix. Suárez et al. (2024) provide a toolkit for instantiating RC networks with biological connectome data.

3.3.4 Breaking symmetry

Using tanh as the non-linear function f in Eqn.1 has a limitation, because tanh is an odd function: $\tanh(x) = -\tanh(-x)$. This can be an issue in some cases where the task does not have this odd symmetry.

The standard way to break the symmetry is to add a bias term to the argument of the non-linear function:

$$\mathbf{x}(t+1) = f(\rho \mathbf{W}\mathbf{x}(t) + \mathbf{W}_u\,\mathbf{u}(t) + \mathbf{b}) \tag{5}$$

where $\mathbf{b}$ is an N-dimensional bias vector. This can be implemented by extending the input vector $\mathbf{u}(t)$ with an extra constant value, $\mathbf{u}_b(t) = (\mathbf{u}(t); 1)$, and using an $N \times (N_u + 1)$ input weight matrix $\mathbf{W}_{ub}$:

$$\mathbf{x}(t+1) = f(\rho \mathbf{W}\mathbf{x}(t) + \mathbf{W}_{ub}\,\mathbf{u}_b(t)) \tag{6}$$

Pathak et al. (2018) break symmetry on the output by adding a non-linear term, as (in the notation used here; compare Eqn.2)

$$\mathbf{v}(t+1) = \mathbf{W}_{v1}\mathbf{x}(t) + \mathbf{W}_{v2}(\mathbf{x}(t) \odot \mathbf{x}(t)) \tag{7}$$

where $\mathbf{x} \odot \mathbf{y}$ denotes the Hadamard, or elementwise, product of vectors $\mathbf{x}$ and $\mathbf{y}$.

3.3.5 Suitability for physical implementation

We have the following: (i) the ESN performance is insensitive to neuronal topology; (ii) training does not access any internal parameters of the reservoir; (iii) Eqn.1 has the form of a discrete time open dynamical system with a particular transfer function.

These make the ESN model suitable for PRC. The (simulated, neuronal) dynamical system can be replaced by a material system that has an appropriate dynamic response when driven with an input signal. The material is a black box reservoir: its internal structure need not have any analogue of neurons; all that is needed is for the PRC's own dynamics to be sufficiently non-linear and dissipative to satisfy the overall requirements of the ESN model.

Thus the ESN model provides an appropriate *computational abstraction*, without needing an implementation of its detailed *internal structure*.

3.4 Overcoming state observation limitations

The size of the output weight matrix $\mathbf{W}_v$ is $N_v \times N$. This implies that the entire reservoir state, all N degrees of freedom, is accessible and observable, to be operated on by this matrix. In a physical system, this is not necessarily so: only a few observations of the substrate's state, N_{obs}, may be observable each timestep. In such a case, the output weight matrix has size $N_v \times N_{obs}$. Since several aspects of the computational power of the reservoir are limited by N_{obs} (Sect. 7.3), this should be as large as possible. If there are physical limitations to the number of observations, there are several approaches to increasing the effective N_{obs}, which we review in this section.

3.4.1 Delayed output reservoir

In the extreme case, it may be possible to take only a single reading from the PRC at each timestep, giving $N_{obs} = 1$.

Chen et al. (2022) use a technique from the time-delay embedding literature to generate a k-d 'state vector' from a time series of k 1-d scalar observations, suitable for their use in training a neural-ODE model of a PRC.

In the notation used here, the process takes a series of scalar state observations $x(t)$, and constructs a derived k-d 'state vector' $\mathbf{x}(t)$ as:

$$\mathbf{x}(t) = (x(t), x(t+1), \ldots, x(t+k-1))^{\mathsf{T}} \tag{8}$$

3.4.2 Delay feedback reservoir

Time multiplexing model

Appeltant et al. (2011) introduce the delay feedback reservoir computer (DFRC), which has only a single non-linear node with a single input and output. The output, after a time delay, is fed back and added to the external input. The approach allows for an increased size of observable state, at the expense of higher frequency inputs and outputs.

In standard ESNs, the input is 'space-multiplexed' through the random input weight matrix across the degrees of freedom of the reservoir. In delay feedback reservoirs, the input is *time multiplexed* through a random input mask

of size N: the input is multiplied by the terms in the mask to give a sequence of inputs, $u(t) \rightarrow (u_m(t), u_m(t+1/N), \ldots, u_m(t+(N-1)/N)$, where $u_m(t+k/N) = mask(k) \times u(t)$. This sequence is fed into the reservoir at a frequency N times that of the input frequency. This conceptually adds N 'virtual nodes' along the delay path, leading to an N-node ring topology of the underlying reservoir.

In the literature, the input is typically a scalar quantity u, and the mask is a 1d vector of N weights. The approach can be extended to an N_u-d vector input $\mathbf{u}$, with a matrix mask of size $N \times N_u$. The mask is simply the input weight matrix reinterpreted as a time multiplexer, rather than a space multiplexer as in the ESN case.

The state is read at the end of the delay line at the same frequency as the masked input, leading to N scalar values per original timestep. The system's observed scalar state at each of these N sub-timesteps is assembled into a single N-d vector state at each original timestep: $\mathbf{x}(t) = (x(t), x(t+1/N), \ldots, x(t+(N-1)/N)^{\mathsf{T}}$. The output weight matrix is trained in the usual way on this state vector.

The time delay is readily implemented by a long optical fibre, and so this approach is particularly suitable for optical systems. However, the time delay can also be implemented in other modalities, making this approach a favoured one in PRC where the non-linear material has restricted input and output capabilities. The downside is that the inputs and outputs need to be fed into and read from the reservoir at a frequency N times higher than the external input frequency. For large N, this may lead to implementation challenges.

Mackey–Glass non-linear node

Appeltant et al. (2011) use a Mackey–Glass system[2] for their non-linear node. This model is derived from a delay differential equation introduced to model certain physiological systems (Mackey and Glass 1977). The paper examines various models to characterise illness, including growth rates of white blood cells in leukaemia, through changes to the variables that lead to these systems turning chaotic. The particular model used as the basis for the non-linear node is (Mackey and Glass, 1977, Eqn.4b):

$$\frac{dP}{dt} = \frac{\beta\theta^n P(t-\tau)}{\theta^n + P(t-\tau)^n} - \gamma P(t) \tag{9}$$

where $P(t) > 0$ is the concentration of circulating blood cells at time t; τ is the time delay between initiating blood cell production and the mature blood cells being released; $\beta > 0$ is the base level production rate of cells; $\theta > 0$ is the baseline concentration; $n > 0$ is a real-valued non-linearity parameter; $\gamma > 0$ is the decay rate of cells.

Normalising the concentration with respect to the baseline, $x(t) := P(t)/\theta$, gives the more usual form of the Mackey–Glass chaotic equation (Glass and Mackey, 2010, Eqn.1):

$$\frac{dx}{dt} = \frac{\beta x(t-\tau)}{1 + x(t-\tau)^n} - \gamma x(t) \tag{10}$$

Here x is a dimensionless normalised quantity, expressed in 'units' of the parameter θ. θ is still a parameter of the model, but is now implicit in the equation: it provides the scale for the dependent variable x.

There are three timescales in Eqn.10: $1/\beta$ (production, or growth, timescale); $1/\gamma$ (decay timescale), and τ (time delay). Some authors further normalise these with respect to the decay timescale, to have two independent timescales: γ/β and $\gamma\tau$. However, when using this model in a physical real-time system, not performing this further normalisation makes it simpler to map the timescale parameters to physical timescales.

Appeltant et al. (2011) use the time-normalised model as the basis for their non-linear node, and add an input term. Performing an analogous modification on the non-time normalised Eqn.10 gives

$$\frac{dx}{dt} = \frac{\beta(x(t-\tau) + \alpha u(t))}{1 + (x(t-\tau) + \alpha u(t))^n} - \gamma x(t) \tag{11}$$

where u is a dimensionless input, with the same normalisation as x, and α is an input scaling parameter, to bring the magnitude of the normalised input suitably in line with that of the state (see Sect. 3.2.2).

The original Mackey–Glass model (Eqn.9) has a positive-valued dependent variable P. In its adaptation for DFRC (Eqn.11), care needs to be taken if the input u can be negative, and make the term $x(t-\tau) + \alpha u(t)$ negative. If this is the case, the non-linearity parameter n should be restricted to integer values.

The Mackey–Glass system is a particularly common model for simulated delay feedback reservoirs, as it has several useful parameters: in addition to the various timescale parameters, the non-linearity parameter n moves the autonomous system between periodic and chaotic behaviours. Furthermore, it is readily implemented in hardware with just a few analogue electronic components, providing a readily traversable route from theoretical model to physical implementation.

3.4.3 Rotating neurons

The DFRC architecture is an interesting variant on reservoir computing, demonstrating that a full recurrent network of non-linear nodes is not necessary: a single source of

[2] This delay differential equation is also used in a different context in RC as a benchmark task (see Sect. 7.2.1). These different usages should not be confused.

non-linearity, and a single recurrent delay line, is all that is needed. However, it does have some implementation issues, such as smoothly integrating the feedback in the same medium as the rest of the reservoir system, and the increase in input and output frequency by a factor of N needed for time-multiplexing.

Liang et al. (2022) introduce the 'rotating neuron' architecture, designed to overcome some of these implementation issues by more readily permitting an all-analogue implementation (although potentially introducing different implementation issues). This has N non-linear nodes connected in a cycle. A rotor feeds a differently weighted input to each neuron: the rotor rotates by $1/N$ each timestep, so at the next timestep each input weight is used for the next node. A similar rotor setup is used for the output. The authors demonstrate this is mathematically equivalent to a simple cyclic reservoir, which itself is of comparable power to a fully connected reservoir (Rodan and Tiňo 2011).

3.4.4 Discussion

What these variant approaches demonstrate is that the specific underlying architecture of an ESN, that of being a random recurrent neural network, is not critical to the functioning or power of the reservoir. Different topologies and different output styles can be exploited to make the reservoir easier to physically construct, and to increase its computational capacity. The engineered substrate need not correspond faithfully to the internal structure of the ESN computational mode. Nor need it correspond faithfully to any of these variants: they could be modified, extended, or combined as engineering constraints and task requirements dictate.

4 Choosing a material substrate

As mentioned above, AR/T can be used in a 'substrate first' manner for designing and analysing physical computing, but here we are starting with the computational model, reservoir computing, then evaluating substrates in this context. This is the bottom arrow of Fig. 1.

4.1 Everything reservoirs

From a materials science perspective, there is an embarrassment of riches, because nearly every substrate 'reservoirs' to some degree or other. A wide range of substrates are described in the literature, including analogue electronic circuits, atomic switch systems, carbon nanotubes, magnetic metamaterials, memristors, MEMS devices, octopus arms, opto-electronics, photonics, spintronic systems, tensegrity structures, and more (Allwood et al. 2023; Caluwaerts et al. 2013; Dale et al. 2021a; Dion et al. 2018; Grollier et al. 2020; Jensen and Tufte 2017; Nakajima et al. 2013; Nowshin et al. 2020; Tanaka et al. 2019; Nakajima and Fischer 2021).

The reason nearly every substrate 'reservoirs', despite most substrates having nothing approaching an internal neuronal structure, is to do with the black box nature of the reservoir, and the form of its dynamics, eqns.1 and 2.

The black box nature means that the substrate does not need to mirror a random RNN: it merely needs to have a high-dimensional state that can be measured (either directly, or exploiting delayed outputs, see Sect. 3.4.1), that undergoes an analogous dynamics when excited by an input, and that has the fading memory property. Equation 1, an RNN, is just a special case of a (discrete time) non-linear dynamical system.

Most substrates are non-linear dynamical systems, and these dynamics can be directly exploited computationally. They are also dissipative, and hence have fading memory. Given the wide range of potential substrates, this tutorial cannot be a cookbook of how to build a specific PRC; instead, it focusses on the general design questions that need to be addressed in different ways for different systems.

4.2 Not everything reservoirs well

From an engineering perspective, the choice of a suitable substrate for a given class of application requires consideration of how the dynamical system can be engineered into a system that is a good reservoir, and interfaced effectively. The main issues with finding an appropriate substrate concern:

1. Matching its natural timescale with that of the input (if being used in real-time), so that its memory fades, but not too fast (Sect. 4.3);
2. Ensuring that it is robust to environmental effects and noise (Sect. 4.4);
3. Achieving sufficient computational capacity for the application (Sect. 4.5);
4. Exciting it with the inputs (Sect. 5);
5. Measuring states and getting the outputs (Sect. 6);
6. Configuring it for different applications, or tuning its performance for a given application (Sect. 7).

4.3 Timescales

For a PRC interfacing to a real world input, there are two timescales of interest: that of the input, and that of the substrate.

The timescale of the input is partly given by the phenomenon producing it, and partly by any preprocessing performed. The timescale of the substrate is given by its

natural dynamics, and depends on the underlying physical system. Response frequencies of different systems range over many orders of magnitude, from gigahertz or terahertz in photonic and magnetic systems to sub-hertz in tensegrity and chemical systems.

If the natural dynamics of the substrate is much faster than the input, then it will have relaxed and have no memory of the previous input by the time the next arrives. Alternatively, if its natural dynamics is much slower, it will not be able to respond substantially to an input before the next arrives. The timescales need to be commensurate.

4.4 Robustness and noise

In PRC, the physical properties of the substrate are being exploited directly. These properties may change with environmental changes, such as temperature and surrounding RF noise. The substrate may need to be calibrated to current environmental conditions (or use its own computational properties to correct for them), and may need to be isolated from environmental noise that cannot be compensated for in the device. In particular, the system needs to be isolated from any interface, measurement, and experimental devices that might themselves be reacting physically to the inputs, and thereby contributing an unwanted extra dynamics to the system.

Even if the temperature of the device is kept constant, the substrate itself will have internal thermal noise. There may also be other forms of internal noise. Multiple measurements of the substrate state, $\mathbf{x}(t)$, may be needed to obtain an average response.

An additional form of noise is measurement noise. The state of the substrate is measurable only with limited precision, and the measurements may vary due to noise of the measurement device.

4.5 Computational capacity and scaling

The computational capacity of a reservoir is governed by the number of observable nodes (Sect. 7.3): the input is projected into a high dimensional space, and sufficient dimensions must be observable so that a linear output layer can separate different classes of inputs.

Having a larger device may allow more inputs to be provided, and outputs to be read, in parallel, although with some kinds of devices, these numbers might scale with perimeter rather than area of material. Even if a substrate supports a large number of state observations, it might not realise its full potential capacity, due to intrinsic dynamical limitations. This may be a function of the configuration parameters (Sect. 7), or the physical configuration of the material. For example, if the dynamics is dominated by edge effects, a rectangular rather than square device of the same area might perform better (Dale et al. 2021c); different thicknesses of substrate might perform differently (Dale et al. 2024).

4.6 Physical reservoirs with neuronal structure

As noted in Sect. 3.3.5, the material substrate of a PRC need not have a neuronal structure. However, some architectures do have such a structure, explicitly wiring together nonlinear components into a network, thereby giving access to the internal weights $\mathbf{W}$. The ESN model uses *random* weights, but, as noted at the end of Sect. 3.2.3, these weights may need to be modified to improve performance. It can be similarly worth training the connection weights in a PRC that has a neuronal architecture.

Training the weights of a PRC is more difficult than training an RNN: there is noise, and there may be no differentiable model of the nodes for calculating backpropagation gradients. Additionally, training requiring multiple evaluations can be slow in hardware. Recent work has been tackling these issues in a variety of ways; see, for example, Manneschi et al. (2024), Momeni et al. (2024), Nakajima et al. (2022), Wright et al. (2022).

5 Input

The first link between the abstract computational model and the physical computing device is established by the instantiation of inputs (the down arrow of Fig. 1).

There are distinct kinds of input to a PRC being used to compute:

- Substrate configuration values that make the system a good reservoir;
- Input interface values including $\mathbf{W}_u$;
- Trained output weight matrix $\mathbf{W}_v$ for performance of a particular task (Sect. 6);
- Input stream $\mathbf{u}$ for the specific task instance.

5.1 Substrate configuration values

Substrate configuration values are what makes a material a specific reservoir substrate, what makes it a specific computer; they can be thought of as the analogue of the ESN parameters such as $\mathbf{W}$ and ρ (Eqn.1).

These may be fixed hardware values, such as material dimensions and component placement, applied when engineering the substrate. They may be field programmable values, such as applied voltages, used when the system is running. Field programmable values may be varied for different tasks.

Suitable configuration values may be found through search (Sect. 7.4) in physical or simulated systems (Sect. 8).

5.2 Input interface values

In the ESN model, the input weight matrix $\mathbf{W}_u$ takes the external input and distributes it randomly to each node. For vector valued input, it distributes a randomly weighted sum of each vector component to each node.

In a PRC substrate, there may be no direct analogue of an ESN's nodes. A means of distributing the input signal with different weights to stimulate different degrees of freedom of the substrate needs to be devised and implemented. The temporal input masking for DFRCs requires different design considerations: a given input signal needs to be modulated over time.

If negative-valued weights are required, this may constrain implementation options: for example, voltages can be negative or positive, but resistances are positive only. This constraint may be overcome by offseting the mapping between abstract and physical values, for example, by mapping the abstract interval $[-1, 1]$ to the physical interval $[250\Omega, 750\Omega]$.

Other design considerations include modality changes: is the external input the same modality as that applied to the substrate, or is some transduction layer needed, for example, from acoustic input to electrical stimulus. In many cases, this transduction is achieved using conventional computing, with associated analogue to digital conversion of the physical input, digital multiplication by the input weight matrix, then digital to analogue conversion to stimulate the substrate; fully analogue implementations are rare.

ESN input weights are randomly chosen, but improved values may be found by search. For PRC, input weights can also be chosen randomly, or, like substrate configuration values, may be optimised through search. Similarly, if the input weights are field programmable, they may be optimised for different tasks.

5.3 Input stream

There are different uses of the input stream:

1. Washout, to remove the influence of any previous computing activity;
2. Training, to provide outputs for training the output weight matrix $\mathbf{W}_v$ (Sect. 3.2.4);
3. Testing, for evaluating how well the trained reservoir performs (Sect. 7.1);
4. Task, for using the trained reservoir to compute.

All of these require the same consideration of physical constraints.

Encoding

The physical signals comprising the input stream may be directly sensed inputs, such as acoustic or magnetic signals, in which case the data is trivially encoded as analogue values.

In other uses (particularly in benchmarking and characterisation, Sect. 7), some separate source of data may need to be encoded in the physical signal, or, more usually, directly sent to the input interface processing layer (Sect. 5.2).

If significant transduction is performed, consideration of how the raw input is encoded for stimulating the substrate is needed. For example, audio inputs might be fourier transformed, with different frequency components extracted. If the input is encoded as the modulation of some carrier signal, then it could be amplitude or frequency modulated. Experimentation might be needed to discover a good encoding. Ideally, such preprocessing should be minimised, and its cost needs to be considered as part of the overall computational cost (Blakey 2017).

Normalisation and modality

Physical inputs should be normalised to the dynamic range acceptable to the substrate, to avoid damaging or saturating it. Normalisation might also require amplification, so that the input signal sufficiently stimulates the PRC.

For vector input, on multiple channels potentially from different sources, the separate channels may need to be normalised individually. If the individual inputs are of different modalities, extra care is needed to ensure they all receive equal attention from the reservoir.

Timescale: discrete or continuous

The input timescale and the reservoir timescale need to match (Sect. 4.3). If the input is from a real-time source, then the substrate needs to be matched to it. If it is not real-time data (for example, benchmark data), then the input rate can be matched to the substrate.

The ESN model is a discrete time model. If the input is discrete (again, for example, because it is ESN benchmark data), then it can be fed into the reservoir at discrete time-steps. If the input is continuous data, then there are two approaches. It can be discretised through a standard 'sample and hold' approach. Alternatively, if the input interfacing (Sect. 5.2) has been implemented in an analogue manner, the input can be fed as a continuous signal to the reservoir.

6 Output

The other link between the abstract computational model and the physical computing device is established by the representation of outputs (the up arrow of Fig. 1).

Many of the issues in designing the output are complementary to designing the input:

- Exciting the reservoir state with inputs, versus measuring reservoir state for output
- Processing external inputs through the input weight matrix, versus processing measured state values through the output weight matrix
- Encoding data in the input stream, versus decoding output data from the output stream

6.1 Measuring reservoir state

A means of measuring a high-dimensional reservoir state that can then be multiplied by the output weight matrix needs to be implemented. A given substrate, despite potentially having a high number of internal degrees of freedom, may have a restricted number that can be readily observed. Dimension-increasing techniques (Sect. 3.4) may be needed, which may add complexity to the implementation.

6.2 Output weight matrix

Output weight matrix values are in essence the program that makes the RC perform a particular task. They are found through training (Sect. 3.2.4).

In the ESN model, the measured state of the nodes, $\mathbf{x}$, is multiplied by the weight matrix $\mathbf{W}_v$ to give the trained output $\mathbf{v}$. Implementing this involves many issues similar to those of implementing the input weight matrix (Sect. 5.2), some in reverse, for example: accessing and gathering the state values, implementing negative-valued weights, and modality transduction.

The output is often assumed to be digital and electronic, so often the observed state is passed through analogue to digital conversion, and the output weight matrix multiplication is performed digitally.

6.3 Decoding the output stream

Even if a fully analogue approach to output weight matrix multiplication is designed, some form of final output discretisation is needed to implement the standard training algorithm (Sect. 3.2.4). Having the input and output timesteps the same makes this training much simpler (Cucchi et al. 2022, Sect.3.3).

A fully analogue means of generating the output stream potentially allows embedded devices with multiple output modalities, allowing outputs to be directed to a range of other embedded processors and actuators.

Once the raw physical output has been extracted from the reservoir, as a voltage value, or a spin, or some other quantity, further processing and filtering can be applied as needed. For example, an averaging across multiple sources might be needed to reduce noise, thresholding might be applied to determine a dominant value. The final value, if to be used in some other computation, may needs to be represented at a higher level of abstraction in the usual manner in computing (decoding as bits, digits, or other symbols, for example).

In the case of a feedback reservoir (Sect. 3.3.2) or DFRC (Sect. 3.4.2), the output will need to be fed back in as another source of input. This may require a change of modality and rescaling.

7 Characterising the computational capacity of a PRC

Once we have the links between the abstract computational model and the physical computing device from the instantiation of inputs and the representation of outputs, the final part of the physical computing design and analysis is to ensure that the resulting diagram 'sufficiently commutes' (the $=_\varepsilon$ in Fig. 1). Once this has been established, the device can be used to compute thereafter.

7.1 Defining the error

Several different error measures, for how much the observed output deviates from the target output, are used in the literature. Here we document the most common ones.

Notation

Given a time series[3] $\mathbf{v} = v(t), t \in 1..T$, we write the mean of these values in the usual way as

$$\langle \mathbf{v} \rangle := \frac{1}{T} \sum_{t=1}^{T} v(t) \tag{12}$$

In the following definitions, the target time series is $\hat{\mathbf{v}}$ and the observed is $\mathbf{v}$.

Mean square error, MSE

Mean squared deviation of observed and target output values; analogue of the variance. If the values have dimension D, the MSE has dimension D^2

$$\text{MSE } (\hat{\mathbf{v}}, \mathbf{v}) := \frac{1}{T} \sum_{t=1}^{T} (\hat{v}(t) - v(t))^2 = \langle (\hat{\mathbf{v}} - \mathbf{v})^2 \rangle \tag{13}$$

[3] Here, for notational simplicity, values at each time point are assumed to be scalar, $v(t)$, and we use $\mathbf{v}$ to denote the list of length T formed from the time series of these scalar values. The definitions can be extended to vector-values data points in the usual way.

Root mean square error, RMSE

Root mean squared deviation of observed and target output values; analogue of the standard deviation. If the values have dimension D, the RMSE also has dimension D.

$$\text{RMSE}\ (\hat{\mathbf{v}}, \mathbf{v}) := \sqrt{\text{MSE}\ (\hat{\mathbf{v}}, \mathbf{v})} = \sqrt{\langle (\hat{\mathbf{v}} - \mathbf{v})^2 \rangle} \tag{14}$$

Normalised mean square error, NMSE

The normalised measures are normalised with respect to the deviation of the target from its mean. The NMSE is dimensionless: it is independent of the units or scaling of the target values. NMSE = 1 if each v_t is set to the mean target value $\langle \hat{\mathbf{v}} \rangle$ (Lukoševičius 2012, p.661).

$$\text{NMSE}\ (\hat{\mathbf{v}}, \mathbf{v}) := \frac{\text{MSE}\ (\hat{\mathbf{v}}, \mathbf{v})}{\text{MSE}\ (\hat{\mathbf{v}}, \langle \hat{\mathbf{v}} \rangle)} = \frac{\langle (\hat{\mathbf{v}} - \mathbf{v})^2 \rangle}{\langle (\hat{\mathbf{v}} - \langle \hat{\mathbf{v}} \rangle)^2 \rangle} \tag{15}$$

Normalised root mean square error, NRMSE

This is the measure most commonly used in the literature.

$$\text{NRMSE}\ (\hat{\mathbf{v}}, \mathbf{v}) := \sqrt{\text{NMSE}\ (\hat{\mathbf{v}}, \mathbf{v})} = \sqrt{\frac{\langle (\hat{\mathbf{v}} - \mathbf{v})^2 \rangle}{\langle (\hat{\mathbf{v}} - \langle \hat{\mathbf{v}} \rangle)^2 \rangle}} \tag{16}$$

7.2 Benchmarking

Benchmarking is a popular way to evaluate the computational capacity of a device.

7.2.1 Common RC benchmarks

There are several common benchmarks used in the RC literature. These can be sub-divided into (Wringe et al. 2024):

- Classification tasks, of classifying input sequences;
- Imitation tasks, of imitating the behaviour of an open dynamical system, given the same inputs;
- Prediction tasks, of predicting the next item(s) in a time series of a closed dynamical system (or a dynamical system where the inputs are unknown or not provided).

Examples of classification tasks include the TI-46 Isolated Spoken Words Dataset classification (Doddington and Schalk 1981), often restricted to just the spoken digits subset. Examples of imitation tasks include NARMA (Sect. 7.2.2), and channel equalisation (Mathews and Lee 1994; Jaeger and Haas 2004). Examples of prediction tasks include the Santa Fe Data Set A laser task (Hübner et al. 1994), the sunspot prediction task (NCEI, nd), and the Mackey–Glass equation (Eqn.10)[4].

A great variety of datasets that can be used for benchmark tasks, although not yet common in the RC literature, can be found in Gilpin (2021)'s database of chaotic dynamical systems, all specified according to guidelines proposed by Gebru et al. (2021).

7.2.2 NARMA

The NARMA (nonlinear autoregressive moving-average) benchmark is possibly the most common benchmark used in the RC literature, so we spend some time here discussing it and its limitations.

The NARMA-N time series is an open delay difference dynamical system. It is an imitation task: the reservoir is trained to imitate, or reproduce, the observed NARMA dynamics given the same inputs.

The general form of the NARMA-N family (Atiya and Parlos 2000; Rodan and Tiňo 2011) is:

$$\begin{aligned} x(t+1) = {} & \alpha x(t) \\ & + \beta x(t) \left(\sum_{i=0}^{N-1} x(t-i) \right) \\ & + \gamma u(t-N+1) u(t) + \delta \end{aligned} \tag{17}$$

Its (quadratic) non-linearity comes from the various $x(t)x(t-i)$ terms, and the $u(t-N+1)u(t)$ term; it also requires memory of the previous N states.

Atiya and Parlos (2000, Eqn.86) introduce this equational form to define NARMA-10. They use inputs $u \in U[0, 0.5]$, and parameter values $(\alpha, \beta, \gamma, \delta) = (0.3, 0.05, 1.5, 0.1)$.

Adapting this for different values of N reveals that the equation can rapidly diverge if not tuned correctly. Indeed, even the standard NARMA-10 setup does itself occasionally diverge (Kubota et al. 2021). Other authors have defined the form for further values of N, and taken a variety of approaches to controlling divergence. For example, Schrauwen et al. (2008, p.1164) define NARMA-30 with different parameter values of (0.2, 0.04, 1.5, 0.001), nevertheless, it too very occasionally diverges; Rodan and Tiňo (2011, Eqn.6) defines NARMA-20 and add a tanh(.) wrapper to stop divergence; Fujii and Nakajim (2017, Eqn.18) define NARMA-5, 10, 15, and 20 with the standard parameter values but a reduced input range $u \in U[0, 0.2]$.

Furthermore, various NARMA timeseries each have their own range of output x values. For example, Atiya and Parlos (2000)'s NARMA-10 x values range between (ignoring divergences) 0.15 and 1, Fujii and Nakajima (2017)'s NARMA-10 with reduced u range has x values ranging between 0.15 and 0.25, and Schrauwen et al. (2008)'s NARMA-30 x values range between (again

[4] As noted earlier (Sect. 3.4.2), this is a different use of the Mackey–Glass system from its use as the non-linear component in DFRC.

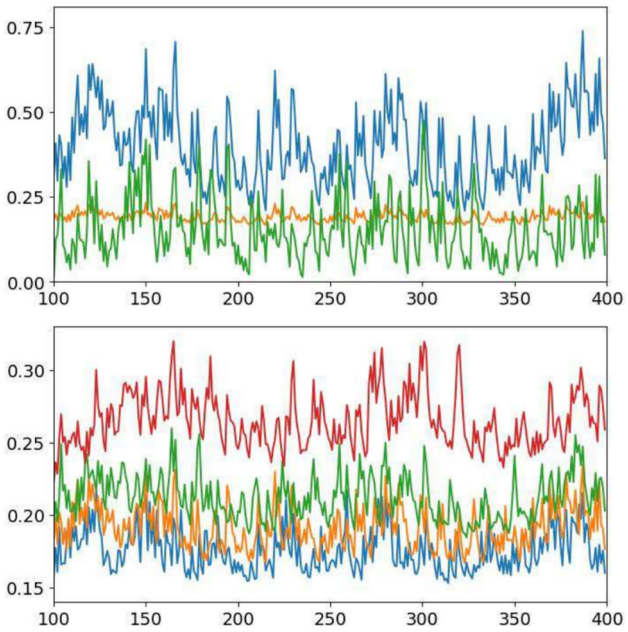

Fig. 2 The NARMA family, for various parameter values $N, (\alpha, \beta, \gamma, \delta), u_{max}$, same random inputs u_i, plotted for timesteps $t = 100 - 400$. (top) blue, Atiya and Parlos (2000)'s original NARMA-10 values, 10, (0.3, 0.05, 1.5, 0.1), 0.5; orange, the original values, but with $u_{max} = 0.2$; green, Schrauwen et al. (2008)'s values 30, (0.2, 0.04, 1.5, 0.001), 0.5. (bottom) Fujii and Nakajima (2017)'s approach, using Atiya and Parlos (2000)'s original values but with $u_{max} = 0.2$: blue, $N = 5$; orange, $N = 10$; green, $N = 15$; red, $N = 20$

ignoring divergences) 0 and 0.6 (see Fig. 2, top). Even using consistent parameter values and input range yields systematically different output ranges for different values of N (see Fig. 2, bottom). This makes any performance comparison as a function of N problematic.

There are several issues with the NARMA benchmark: divergence, a multitude of parameter values in the literature, and difficulty of making any conclusions as a function of N. As such, its use as a benchmark should be approached with caution. NARMA has features of a non-linear memory task, so using the generic, and more readily interpretable, information processing capacity measure (Sect. 7.3.2) overcomes these limitations.

7.2.3 Issues with benchmarks

A given reservoir might perform very well on one class of benchmark tasks, but less well on others. A range of benchmark classes should be used. Even so, using benchmarks as a means to assess general capabilities of a reservoir, as opposed to capabilities on specific tasks, have their issues.

Different benchmark tasks may require very different computational capabilities; for example, one task may require memory of many past inputs, but little processing, whereas another may require the opposite. How does performing well on one of these tasks transfer to the suitability of the reservoir for another?

Also, it is often interesting to assess how reservoir capabilities scale with reservoir size or other parameter values. This requires a parameterised range of task difficulties. Even benchmarks that appear to have such a parameter (for example, NARMA, Sect. 7.2.2) are not easy to interpret in this manner.

7.3 Task-independent measures

Because of the issues with benchmarks, *task-independent measures*, more closely related to underlying theoretical computational concepts, can be used to assess generic properties of reservoirs. There are several theoretical measures that can be used for simulated ESNs, where one has access to the internal parameters of the system. However, for PRC, measures that rely only on the input–output mapping, rather than on, say, building two reservoirs with only a small difference in internal state, are needed, since one rarely has access to internal values in the manner needed.

There are several measures that fulfil these requirements. Dale (2018), Love et al. (2021) discuss a range. Here, we describe some rank-based measures, and some memory capacity measures, that are often used in the literature. These measure different capabilities of the reservoir.

7.3.1 Rank-based measures

We can calculate a general rank measure of an N-node ESN as follows.

1. Provide input to the reservoir.
2. Measure the reservoir state $\mathbf{x}$ at $S \geq N$ timepoints t_i, giving a sequence of N-d (column) vectors $\mathbf{x}_i$.
3. Construct the $N \times S$ matrix $\mathbf{X}$, where column i is $\mathbf{x}_i$. So $\mathbf{X} = [\mathbf{x}_1\, \mathbf{x}_2 \ldots \mathbf{x}_S]$.
4. Calculate the rank of $\mathbf{X}$.

What distinguishes the various rank measures is the form of the input; what distinguishes the variant implementations in the literature are the state measurement points.

The maximum possible value of the rank is $\min(N, S)$. Here we have $N \leq S$, so the maximum possible value is N.

Kernel rank and generalisation rank

The particular rank property being calculated depends on the nature of the input. Legenstein and Maass (2007), Büsing et al. (2010) introduce two such measures: kernel quality (kernel rank, KR), and generalisation rank (GR), in the context of evaluating the classification capability of neuronal circuits.

Kernel rank (KR) measures how well the inputs are projected into a high dimensional state space, such that they can be separated by a linear output weight matrix.

High KR indicates good separability. Generalisation rank (GR) measures how robust the reservoir is to noise and avoiding overfitting. Low GR indicates good robustness to noise.

The original definitions are in the context of classifications tasks, and continuous spike train inputs (Legenstein and Maass 2007), or discrete time binary inputs (Büsing et al. 2010). These definitions need to be adapted and interpreted for the discrete time, real-valued inputs of RC, and times series tasks. There are two main approaches in the literature.

Direct translation The original definitions use S distinct input streams, and measure the state at time T, corresponding to the end of the input. For KR, these streams are maximally different, for GR, they are set to be similar.

Vidamour et al. (2022) use a direct translation of this approach, with each stream's values drawn from $U[-1, 1]$. In the case of GR, the last few values of each stream are set to be the same values, making the streams similar. Büsing et al. (2010), Vidamour et al. (2022) take the length of this common tail to be 3. The resulting measures are given in Algorithms 2 and 3.

Algorithm 2 KR, Vidamour et al. (2022)

$S :=$ number of input streams $\geq N$
$T :=$ length of each input stream (timepoints)
run reservoir with washout stream
for $i \in 1..S$ **do**
    $u_i(t), t \in 1..T := U[-1, 1]$
    run reservoir with input stream u_i
    $\mathbf{x}_i(T) :=$ reservoir state at final time T
end for
$\mathbf{X} := [\mathbf{x}_1(T)\, \mathbf{x}_2(T)\, \ldots\, \mathbf{x}_S(T)]$
return rank($\mathbf{X}$)

Algorithm 3 GR, Vidamour et al. (2022)

$S :=$ number of input streams $\geq N$
$T :=$ length of each input stream (timepoints)
$\tau :=$ length of common input stream
$tail(t), t \in 1..\tau := U[-1, 1]$
run reservoir with washout stream
for $i \in 1..S$ **do**
    $u_i(t), t \in 1..(T - \tau) := U[-1, 1]$
    $u_i := u_i + tail$ ▷ append the tail
    run reservoir with input stream u_i
    $\mathbf{x}_i(T) :=$ reservoir state at final time T
end for
$\mathbf{X} := [\mathbf{x}_1(T)\, \mathbf{x}_2(T)\, \ldots\, \mathbf{x}_S(T)]$
return rank($\mathbf{X}$)

Adapted translation

Dale et al. (2019) use a different interpretation, which is computationally less intensive (requiring S inputs, rather than $S \times T$), and adapted to time series tasks (where the reservoir state at each timepoint is used), but is further from the original definitions. In this approach, there is a single input stream, of length S, and the reservoir state is measured at each timepoint. For KR, the stream's values are drawn from $U[-1, 1]$, thereby making them distinct; for GR they are drawn from $U[-\delta, \delta]$, a small amount representing the noise, thereby making them similar. The choice of δ is somewhat arbitrary, but should represent the expected level of noise in the inputs. The resulting measures are given in Algorithm 4. In Dale et al. (2019), KR has range $r = 1$; GR has range $r = 0.1$.

Algorithm 4 KR and GR, Dale et al. (2019)

$S :=$ length of input stream $\geq N$
$r :=$ range of input, 1 for KR, 0.1 for GR
run reservoir with washout stream
for $t \in 1..S$ **do**
    step reservoir with input $u(t) \in U[-r, r]$
    $\mathbf{x}(t) :=$ reservoir state at time t
end for
$\mathbf{X} := [\mathbf{x}(1)\, \mathbf{x}(2)\, \ldots\, \mathbf{x}(S)]$
return rank($\mathbf{X}$)

Choosing parameter S

Büsing et al. (2010) use the minimal value $S = N$, giving a square matrix $\mathbf{M}$. Dale et al. (2019, §D.a) note that the measured rank increases with S, until it eventually converges (Fig. 3). This is due to increasing the chance that the sampled inputs drive the reservoir into the full space that it can reach. Preliminary investigation should be performed to establish a suitable value for S.

Calculating the rank

The standard way to calculate the rank of the matrix $\mathbf{M}$ is to use singular value decomposition (SVD). This gives $\mathbf{M} = \mathbf{U}\Sigma\mathbf{V}^{\mathsf{T}}$, where $\mathbf{U}$ and $\mathbf{V}$ are rotation matrices, and Σ is an $N \times S$ diagonal matrix. The diagonal values σ_i of Σ are the *singular values*, which are real and non-negative. The *rank* of $\mathbf{M}$ is the number of these σ_i that are non-zero. Rank is an integer, so, for small N, this can result in a very granular measure.

Due to numerical effects and noise, typically all the singular values derived by this process are non-zero, but some may be very small. To make the measure meaningful (rather than always N) in practice a threshold is chosen, below which the singular values are taken to be effectively

zero. This threshold is typically expressed as some percentage of the maximum singular value. (It should not be chosen as an absolute value, as rescaling $\mathbf{M}$, for example, due to a change in physical units, similarly rescales the σ_i.) The threshold value is essentially arbitrary, and affects the measured rank (Fig. 3), so should be stated in any results.

An alternative to this integer-valued rank with threshold is the real-valued *effective rank* (Roy and Vetterli 2007; Love et al. 2021). Normalise the singular values to sum to one: $p_i = \sigma_i / \sum_i \sigma_i$. The effective rank is defined as $\exp(-\sum_i p_i \ln p_i)$. In the case that R of the singular values are the same, and the rest are zero, then each $p_i = 1/R$ or 0, giving an effective rank of R, which is the same as the standard rank value. For other cases, effective rank gives continuous values, has no arbitrary threshold, and weights the singular values according to their size, potentially giving a more meaningful result.

Discussion of rank measures

For KR, alg.2 reduces to alg.4 if $T = 1$ (and $r = 1$). These variants can be considered essentially the same measure, as they typically give similar results: they are merely altering which states are used in the calculation.

For GR, algorithm 3 and algorithm 4 are more different, both in how they define similarity, and in the range of inputs used to drive the reservoir. They can give quite different results, so should be considered distinct measures. GR has further issues in its definition, in particular, the definition of 'similarity' is somewhat arbitrary. Other measures might be preferable, for example, entropy based measures (Griffin and Stepney 2024), but are not yet as well-established in the literature.

If the physical inputs are provided as a modulation to some driving input, for example, as amplitude modulation to a sinusoidal magnetic field, then all the input streams used for measuring KR and GR should be similarly added to that driving input.

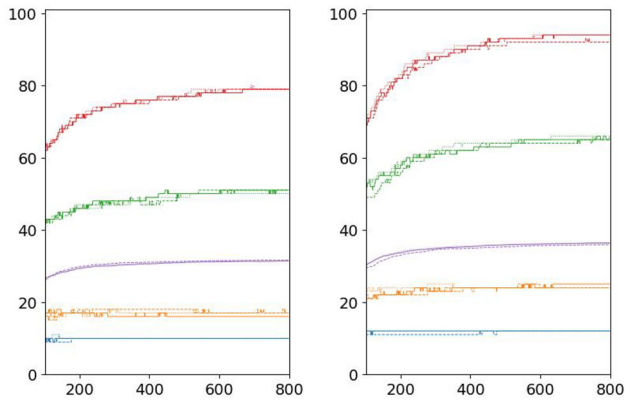

Fig. 3 Illustration of the Dale et al. (2019) algorithm for KR (y-axis), as a function of length of stream S (x-axis) and threshold (blue line 10% of maximum singular value, orange 5%, green 2%, red 1%, purple effective rank). The two plots show two different $N = 100$ node reservoirs, each with three different input streams; the variation between reservoirs is much larger than the variation between inputs

7.3.2 Memory capacity

Linear memory capacity

Jaeger (2002) introduces (linear) memory capacity for a reservoir with scalar input. Given input $u(t) \in U[-1, 1]$, the reservoir is trained to reproduce this input, delayed by a number of timesteps k. To do this, the target output vector is defined as $\hat{\mathbf{v}}(t) = (u(t-1), u(t-2), \ldots, u(t-k), \ldots)^{\mathsf{T}}$. Then linear memory capacity for a delay of k is defined as the covariance squared of the delayed input and the respective observed output, normalised by the variances of the input and the observed output:

$$MC_k = \frac{cov^2(u(t-k), v_k(t))}{var(u(t))var(v_k(t))} \tag{18}$$

$$MC = \sum_{k=1}^{\infty} MC_k \tag{19}$$

where v_k is the kth component of the trained output vector $\mathbf{v}(t) = \mathbf{W}_v \mathbf{x}(t)$, the reservoir's 'memory' of the k-delayed input value.

Jaeger (2002) shows that for i.i.d. (independent and identically distributed) random input, $MC \leq N$. White et al. (2004), Rodan and Tiňo (2011) also investigate this MC.

As the delay k grows, each MC_k tends to decrease (and the corresponding NRMSE tends to increase), indicating that inputs after longer delays are remembered less well, and are dominated by noise. In the formal definition of MC (Eqn. 19) the sum over delays goes to infinity; since the small values at high k are essentially noise, however, they should be neglected, so in practice a cutoff is used. Dambre et al. (2012, SupMat3.2) define a threshold based on the size of the reported capacity. Dale (2018) uses a cutoff of $k_{max} = 2N$ (number of nodes).

Non-linear memory capacities

Dambre et al. (2012) generalise the definition of linear memory capacity in a way that allows them to define various non-linear capacities, too: they call this Information Processing Capacity (IPC). In these cases, in addition to remembering past inputs, the output is required to be a non-linear function of those inputs, for example, a cubic function such as $u^3(t-1)$ or $u(t-1)u^2(t-2)$. The cases where the polynomials involve different time delays are called cross memory capacities (Duport et al. 2012).

Dambre et al. (2012) use sets of orthogonal polynomials, to cover all possible non-linear capacities and cross memory capacities with no double counting. They use

normalised Legendre polynomials for reservoir inputs $u(t) \in U[-1, 1]$. They note that different sets of orthogonal polynomials should be used for different input distributions, for example, Hermite polynomials for Gaussian-distributed inputs. Orthogonal sets of trigonometric functions can also be used as the basis.

As with linear memory capacity, the reservoir is trained to output the relevant (here, polynomial) function of its delayed input. For a given degree d of polynomial (linear, quadratic, cubic, etc), the contributions for all delays k, including all combinations of delays in the cross memory capacities, are summed to give an IPC for that degree. Hence linear memory capacity is the IPC for degree $d = 1$. Dambre et al. (2012) prove that the total IPC (from the contributions of all the linear and non-linear polynomials) is $MC = N$ for a system with fading memory. Tsunegi et al. (2023) measure the IPC of a physical spintronic oscillator.

This procedure is computationally intensive, as there are many combinations of polynomials and delays as degree d increases. Contributions at high d decrease, and a cutoff at large d is used. The same observed state data can be used in all the training.

7.3.3 Discussion of task-independent measures

On the one hand, task-independent measures are to be preferred for characterising general reservoir properties, as they minimise bias towards particular inputs or behaviours. Additionally, their results are more readily interpretable in computational terms, and have theoretical bounds. For example, although NARMA (Sect. 7.2.2) looks similar to a non-linear (quadratic) memory capacity, it is unclear exactly what it is measuring, how to generalise it, or what any theoretical bounds on its performance might be. Kubota et al. (2021) use NARMA-10 as a case study, and critique it, in their in-depth analysis of IPC.

On the other hand, since there is no specific well-defined task involved, definitions of the measures may differ in the literature (Sect. 7.3.1). They can also have multiple arbitrary parameters (such as thresholds and input stream lengths). It can be hard to compare results if these details are not reported.

These measures may have particular mathematical properties (such as rank bounded by N, or IPC $MC = N$). However, these are theoretical properties, and both numerical noise (finite input streams and other influences of randomness) and, in physical systems, experimental noise and systematic effects, can make these theoretical properties either trivially achievable (such as ranks, naïvely calculated, always being N), or require arbitrary cutoffs and thresholds to get meaningful results.

In summary: task-independent measures reduce bias and can be computationally meaningful; however, the functions calculated and parameter values used in their measurement need to be explicitly reported, and may vary from system to system.

7.4 Searching for good reservoir configurations

ESNs and PRCs are not in fact black boxes: they have many configuration parameters that can be adjusted to affect performance. As can be seen from Fig. 4, the values of the task-independent measures varied widely amongst randomly chosen ESNs. To find a specific ESN good for a particular task requires a search through ESN parameter space. The same is true for PRCs, where the configuration parameter space is substrate and architecture specific.

Optimising PRCs

The approach of searching for good configuration parameter values follows from the original *in materio* computing approach (Miller and Downing 2002), where material configurations (for example, voltages applied at certain positions on the substrate) are evolved such that the system can perform a specific computational task directly. The evaluation of the fitness function is performed directly on the device, leading to 'substrate in the loop' evolution (Harding and Miller

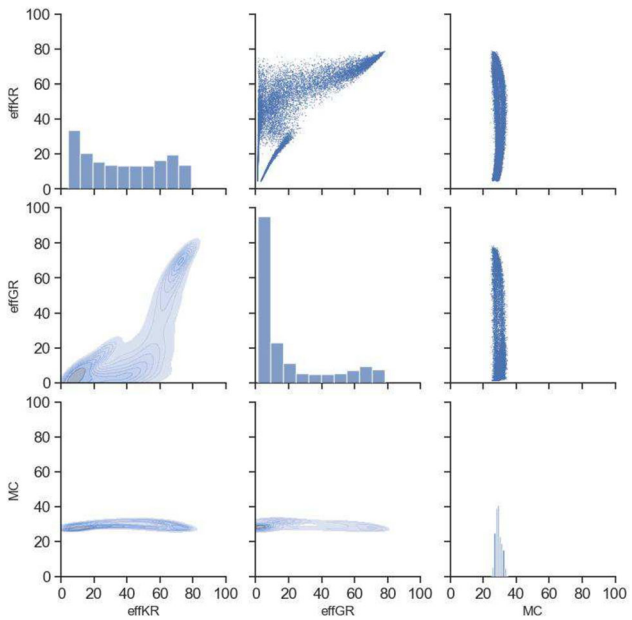

Fig. 4 The effective KR, effective GR (Alg.4) and linear MC (Eqn.19 with cutoff $k_{\max} = 2N$) of 20,000 randomly selected $N = 100$ node ESN reservoirs, displayed as a Scatter Plot Matrix (SPLOM: diagonal shows single measure histograms; upper triangle, pairwise scatter plots; lower triangle, pairwise kernel density estimate plots). Random parameters: random input weight matrix, $\mathbf{W}_{in}$ elements $\in U[-1, 1]$; random state weight matrix, $\mathbf{W}$ elements $\in U[-1, 1]$ then normalised such that the maximum absolute eigenvalue $\max|\lambda_i| = 1$; spectral radius $\rho \in U[0.5, 2]$; $\mathbf{W}$ density (proportion of non-zero weights) $\in U[0.2, 1]$; bias input $\in U[T, F]$

2004, 2005; Clegg et al. 2014; Mohid et al. 2014a, b, c). Each new task requires its own separate evolutionary search.

Dale et al. (2016a, b) use similar substrates, and evolve configurations to perform the higher-level task of 'being a PRC', which can then be trained and retrained for a specific tasks. This substantially reduces the time-intensive evolutionary search stage, which needs be performed only once, to evolve the PRC configuration values, not for each individual task.

The search process needs to restrict the various configuration parameter values to the physically realistic region, to ensure devices are not damaged by being driven outside their design domain during the evolutionary process.

Some configuration parameters require more effort to search over if they are not readily adjustable in a design loop, if they are not 'field programmable'. Such parameters include component values, material composition, and device size and shape, where a change in value requires fabrication of a new instance of the device. In such cases the search might be performed in simulation, if there is a sufficiently good model available (Sect. 8), with only a few devices being fabricated to validate the search process. Care needs to be taken if the search process moves proposed solutions outside the domain of validity of the simulation.

Optimising for tasks

Optimisation requires an objective function, or fitness function. What this should be for a good generic reservoir configuration is not clear. Using specific benchmarks biasses towards those tasks. Using task-independent measures raises the question: what are suitable values of these measures? The answer is that the most appropriate values are task-dependent: some tasks might require high memory capacity, whereas others might do better with a higher kernel rank and lower memory capacity.

How much to tune a PRC through re-evolving configuration parameters, versus reusing a generic configuration, depends on the desired performance, and the use context. If a given reservoir is to be switched between multiple tasks, it might be worth training several configurations, and switching between those as well as switching output weights. In other cases, the reservoir might be required to perform two tasks simultaneously. For example, Cucchi et al. (2022, Sect.2.1.1) describe a hypothetical case of a reservoir being trained to simultaneously classify a time series as a whole, and to detect anomalies within the time series. In this case, the output weight layer would need to be trained on a single 'compromise' reservoir, that can perform each task adequately, but possibly not as well as if it were optimised for either specific task.

7.5 Exploring reservoir configurations: CHARC

It is possible to search for a good reservoir configuration for a particular task, where there is a well-defined objective function (Sect. 7.4). However, a rather different class of task might require a new search for a differently configured reservoir. These individual searches give little insight into the full range of capabilities of a potential substrate.

The CHARC (CHAracterisation of Reservoir Computers) framework (Dale et al. 2019) instead seeks to characterise the full potential of a substrate to be configured for a range of tasks. Instead of using an optimising search with a fitness function, CHARC uses Novelty Search (Lehman and Stanley 2008, 2011). Rather than trying to find a configuration that maps to an optimal place in some fitness landscape, novelty search attempts to find a collection of configurations that map to an approximately uniformly distribution in some 'behaviour space'. In CHARC, the configuration space is whatever the configuration parameters are for the given substrate reservoir, and the behaviour space is taken to be the (KR, GR, MC) values of the relevant reservoirs. The method is not dependent on this particular choice of measures; further or different measures can be used to define different behaviour spaces as appropriate. The search is needed, because a uniform distribution of parameter values in configuration space does not map to a uniform distribution in behaviour space (Fig. 4).

The characterisation process is as follows:

1. Use CHARC to characterise ESNs of a particular size (number of nodes N). This provides the baseline, and need be done only once for a given N, not for every substrate. N is typically a little larger than the expected equivalent N of the PRCs, given by the number of observed nodes, possibly augmented by a method such as that described in Sect. 3.4.
2. For each desired task, find the position in this behaviour space, $(\mathrm{KR}, \mathrm{GR}, \mathrm{MC})_{task}$, where the optimal ESN lies. This location gives the optimal ESN for this task.
3. For each given substrate instance, use CHARC to characterise the PRC in the same manner.
4. Examine the behaviour space volume occupied by the substrate's CHARC results: this gives a measure for the ESN-equivalent size of the PRC, and hence for the range of behaviours the substrate instance can realise.
5. For each desired task, use the values of $(\mathrm{KR}, \mathrm{GR}, \mathrm{MC})_{task}$ found in the ESN step, and back-map to the substrate configuration values that give these same values. Use these configuration values as a starting point to further optimise the configuration, eg, by hill climbing, if needed.

See Dale et al. (2021b) for examples of the technique applied to a carbon nanotube PRC, and a simulated feedback delay RC.

8 Simulating a substrate

Although it is not necessary to have a detailed understanding of substrate dynamics to use a PRC, it can help when designing one. It may be possible to build a PRC purely empirically, but when trying to optimise a design or characterise a substrate (Sect. 7), a physical model can be a great help, both for suggesting the design space, and for exploring it in simulation.

Allwood et al. (2023, Sect.IV) discuss a range of models for nanoscale magnetic systems, from general purpose physics simulators to higher level, special-purpose models. Many authors develop bespoke simulators for their own substrates.

In some cases there may be no straightforward way to model a substrate of interest, for example, due to irregularities in the material. Then it may be useful to *learn* a proxy model of the device, for example, through the approach of neural-ODEs and neural-SDEs (Chen et al. 2022; Manneschi et al. 2024). These models have the advantage of being both learned (so little or no prior model knowledge is needed) and differentiable (for use in other models).

It is necessary to simulate the input and output processes, too. For example, if using an existing simulator built for a different purpose, it may be necessary to add the input term in a manner analogous to modifying the Mackey–Glass system (Eqn.10) to include inputs (Eqn.11). Any pre-processing of inputs should also be simulated, to enable the effects of noise and limited precision to be included. The internal state of a simulated system can often be examined directly: the mapping between the microscopic modelled state values and the experimentally measurable macroscopic values needs to be determined, as does any limitation to the number of values measurable.

If using a simulation in a design search loop, it is important to have reality checks. Evolutionary search is notorious for exploiting bugs in simulations in order to optimise a fitness function (Lehman et al. 2020), and novelty search as used in CHARC (Sect. 7.5) specifically pushes the design space envelope, potentially outside the domain of applicability of the simulation.

9 The future of PRC

The goal of PRC is twofold: to exploit the natural physical dynamics of substrates to perform efficient computation, and to interface naturally to the physical world. Combined, this promises a route to smart sensors and edge computing, where sensing, processing, and signal transduction are all performed seamlessly and efficiently in the same system. For example, one potentially exciting application is recognising that a robot body is a dynamical system, and exploiting this for computation (Hauser et al. 2011; Nakajima 2020; Hauser 2021).

PRC research is moving towards its goal, but has some way to go still. Currently, the main focus is on material substrates, and how they can be made into reservoirs. In the move to real world applications, interfacing, scaling, and training all need to be addressed, moving from single embedded devices with digital interfacing, to architectures of fully embodied devices, integrated into the overall system.

9.1 Interfacing

Some applications are integrating sensing and processing in a single device, such as the MEMS accelerometer PRC (Barazani et al. 2020; Chiasson-Poirier et al. 2022) being developed for gait analysis. However, fully analogue implementations, including the interfacing to input weights (Sect. 5.2) and output weights (Sect. 6.2), are rare.

Embedded reservoirs intended to solve a variety of tasks would potentially need dynamic reconfiguration, of both their configuration parameters and their output weight matrix, which will require more complex interfacing.

9.2 Theory and training algorithms

The ESN training algorithm (Sect. 3.2.4) is also used to train PRCs, even thought they do not necessarily have the same dynamics as an ESN. All the theoretical results relate to the ESN model. The training algorithm is robust, but there is potentially room for improvement by adding some extra facilities or different approaches (Manneschi et al. 2023).

Theoretical modelling and training architectures of multiple connected PRCs, particularly heterogeneous ones, is pushing the state of the art.

9.3 Multi-reservoir architectures

Rather than scaling individual reservoirs (Sect. 4.5), one might wish to connect multiple reservoirs in a network of reservoirs, ‘deep’ reservoirs (Gallicchio et al. 2017), or a reservoir of reservoirs. Homogeneous sub-reservoirs give an easier model, and a potentially easier training approach. Heterogeneous reservoirs would allow for multiple substrates, multiple timescales, multiple input and output modalities, and individual configurations tuned for different parts of the task.

System architectures will need to combine such devices into larger solutions, including other sensors and actuators, and potentially digital processors to perform the compute-intensive tasks beyond the range of reservoirs' capabilities.

PRC is not the last word in physical computing. It does have a computational model particularly suited for smart sensors and edge computing, but other forms of computing exist. An overarching physical computational model, unifying neuromorphic, probabilistic, dynamical systems, classic symbolic computing, and more (Jaeger 2021b; Jaeger et al. 2023), could be the way forward for more complicated systems incorporating computational devices implementing a range of different paradigms.

10 Conclusions

PRC demonstrates that physical materials can naturally compute, although care needs to be taken in distinguishing true computation from mere physical time evolution. Small reservoirs of single materials are routinely designed, built and evaluated. Reservoirs with larger computational capacity, fully analogue, and parts of larger systems, are needed for the technology to reach its full potential. This requires both computational theory and materials engineering advances.

Acknowledgements Thanks to David Griffin, Matt Ellis, and two anonymous referees, for their helpful comments on earlier versions of this piece.

Funding This work was part-funded by the SpInspired project, EPSRC grant EP/R032823/1, and the MARCH project, EPSRC grant EP/V006029/1.

Declarations

Conflict of interest The author has no conflict of interest of a financial or personal nature, nor other interests that might be perceived to influence the results and/or discussion reported in this paper.

Ethical approval Not applicable: no human or animal research involved.

References

Allwood DA, Ellis MOA, Griffin D, Hayward TJ, Manneschi L, Musameh MFKH, O'Keefe S, Stepney S, Swindells C, Trefzer MA, Vasilaki E, Venkat G, Vidamour I, Wringe C (2023) A perspective on physical reservoir computing with nanomagnetic devices. *Appl Phys Lett*, 122(4)

Appeltant L, Soriano MC, Van der Sande G, Danckaert J, Massar S, Dambre J, Schrauwen B, Mirasso CR, Fischer I (2011) Information processing using a single dynamical node as complex system. Nat Commun 2:468

Atiya AF, Parlos AG (2000) New results on recurrent network training: unifying the algorithms and accelerating convergence. IEEE Trans Neural Netw 11(3):697–709

Barazani B, Dion G, Morissette J-F, Beaudoin L, Sylvestre J (2020) Microfabricated neuroaccelerometer: integrating sensing and reservoir computing in MEMS. J Microelectromech Syst 29(3):338–347

Blakey E (2017) Unconventional computers and unconventional complexity measures. In: Adamatzky A (ed) Advances in unconventional computing volume I: theory. Springer, New York, pp 165–182

Büsing L, Schrauwen B, Legenstein R (2010) Connectivity, dynamics, and memory in reservoir computing with binary and analog neurons. Neural Comput 22(5):1272–1311

Caluwaerts K, D'Haene M, Verstraeten D, Schrauwen B (2013) Locomotion without a brain: Physical reservoir computing in tensegrity structures. Artif Life 19(1):35–66

Chalmers DJ (1996) Does a rock implement every finite-state automaton? Synthese 108(3):309–333

Chen X, Araujo FA, Riou M, Torrejon J, Ravelosona D, Kang W, Zhao W, Grollier J, Querlioz D (2022) Forecasting the outcome of spintronic experiments with neural ordinary differential equations. Nat Commun 13(1):1016

Chiasson-Poirier L, Younesian H, Turcot K, Sylvestre J (2022) Detecting gait events from accelerations using reservoir computing. Sensors 22(19):7180

Clegg KD, Miller JF, Massey K, Petty M (2014) Travelling Salesman Problem Solved 'in materio' by Evolved Carbon Nanotube Device. In *PPSN XIII*, pages 692–701. Springer

Cucchi M, Abreu S, Ciccone G, Brunner D, Kleemann H (2022) Hands-on reservoir computing: a tutorial for practical implementation. Neuromorph Comput Eng 2(3):032002

Dale M (2018) *Reservoir Computing in Materio*. PhD thesis, University of York. http://etheses.whiterose.ac.uk/22306/

Dale M, Jenkins S, Evans RFL, O'Keefe S, Sebald A, Stepney S, Trefzer M (2024) Reservoir computing with magnetic thin films. Int J Unconv Comput 19(1):63–92

Dale M, Miller JF, Stepney S, Trefzer MA (2016) Evolving carbon nanotube reservoir computers. In: Amos M, Condon A (eds) Unconventional computation and natural computation, LNCS. Springer, New York, pp 49–61

Dale M, Miller JF, Stepney S, Trefzer MA (2019) A substrate-independent framework to characterize reservoir computers. Proc Royal Soc A 475(2226):20180723

Dale M, Miller JF, Stepney S, Trefzer MA (2021a) Reservoir Computing in Material Substrates. In Nakajima and Fischer (2021), pages 141–166

Dale M, O'Keefe S, Sebald A, Stepney S, Trefzer MA (2021) Reservoir computing quality: connectivity and topology. Nat Comput 20(2):127–143

Dale M, O'Keefe S, Sebald A, Stepney S, Trefzer MA (2021) Computing with magnetic thin films: using film geometry to improve dynamics. In Kostitsyna and Orponen 2021:19–34

Dale M, Stepney S, Miller JF, Trefzer M (2016b) Reservoir computing in materio: An evaluation of configuration through evolution. In *2016 IEEE Symposium Series on Computational Intelligence (SSCI)*, pages 1–8

Dambre J, Verstraeten D, Schrauwen B, Massar S (2012) Information processing capacity of dynamical systems. Sci Rep 2:514

Dion G, Mejaouri S, Sylvestre J (2018) Reservoir computing with a single delay-coupled non-linear mechanical oscillator. J Appl Phys 124(15):152132

Doddington GR, Schalk TB (1981) Computers: Speech recognition: Turning theory to practice: New ICs have brought the requisite computer power to speech technology; an evaluation of equipment shows where it stands today. IEEE Spectr 18(9):26–32

Duport F, Schneider B, Smerieri A, Haelterman M, Massar S (2012) All-optical reservoir computing. Opt Express 20(20):22783–22795

Fujii K, Nakajima K (2017) Harnessing disordered-ensemble quantum dynamics for machine learning. Phys Rev Appl 8(2):024030

Gallicchio C, Micheli A, Pedrelli L (2017) Deep reservoir computing: a critical experimental analysis. Neurocomputing 268:87–99

Gebru T, Morgenstern J, Vecchione B, Vaughan JW, Wallach H, Daumé III H, Crawford K (2021) Datasheets for datasets. arXiv:1803.09010 [cs.DB]

Gilpin W (2021) Chaos as an interpretable benchmark for forecasting and data-driven modelling. arXiv:2110.05266 [cs.LG]

Glass L, Mackey MC (2010) Mackey-Glass equation. Scholarpedia 5(3):6908

Griffin D, Stepney S (2024) Entropy transformation measures for computational capacity. In *UCNC 2024, Pohang, South Korea*, volume 14776 of *LNCS*, pages 119–133. Springer

Grollier J, Querlioz D, Camsari KY, Everschor-Sitte K, Fukami S, Stiles MD (2020) Neuromorphic Spintronics. *Nature Electronics*, 3(7)

Harding S, Miller JF (2004) Evolution in materio: a tone discriminator in liquid crystal. In *CEC 2004*, volume 2, pages 1800–1807. IEEE Press

Harding S, Miller JF (2005) Evolution in materio: a real-time robot controller in liquid crystal. In *2005 NASA/DoD Conference on Evolvable Hardware*, pages 229–238

Hauser H (2021) Physical reservoir computing in robotics. In Nakajima and Fischer (2021), pages 169–190

Hauser H, Ijspeert AJ, Füchslin RM, Pfeifer R, Maass W (2011) Towards a theoretical foundation for morphological computation with compliant bodies. Biol Cybern 105(5–6):355–370

Horsman D, Kendon V, Stepney S (2018) Abstraction/representation theory and the natural science of computation. In: Cuffaro ME, Fletcher SC (eds) Physical perspectives on computation. Cambridge University Press, Computational Perspectives on Physics

Horsman D, Stepney S, Kendon V (2017) The natural science of computation. Comms ACM 60(8):31–34

Horsman D, Stepney S, Wagner RC, Kendon V (2014) When does a physical system compute? Proc Royal Soc A 470(2169):20140182

Huang G-B, Zhu Q-Y, Siew C-K (2006) Extreme learning machine: theory and applications. Neurocomputing 70(1–3):489–501

Hübner U, Weiss C-O, Abraham NB, Tang D (1994) Lorenz-like chaos in NH_3-FIR lasers (data set A). In: Weigend AS, Gershenfeld NA (eds) Time series prediction: forecasting the future and understanding the past. Westview Press, pp 73–104

Jaeger H (2001) The "echo state'' approach to analysing and training recurrent neural networks - with an erratum note. Bonn Germany German Nat Res Center Inf Technol GMD Tech Rep 148(34):13

Jaeger H (2002) Short Term Memory in Echo State Networks. Technical Report GMD report 152

Jaeger H (2007) Echo state network. Scholarpedia 2(9):2330

Jaeger H (2021a) Foreword. In Nakajima and Fischer 2021:v–xi

Jaeger H (2021b) Toward a generalized theory comprising digital, neuromorphic, and unconventional computing. Neuromorph Comput Eng 1(1):012002

Jaeger H, Haas H (2004) Harnessing nonlinearity: predicting chaotic systems and saving energy in wireless communication. Science 304(5667):78–80

Jaeger H, Lukosevicius M, Popovici D, Siewert U (2007) Optimization and applications of echo state networks with leaky-integrator neurons. Neural Netw 20(3):335–352

Jaeger H, Noheda B, van der Wiel WG (2023) Toward a formal theory for computing machines made out of whatever physics offers. Nat Commun 14(1):4911

Jensen JH, Tufte G (2017) Reservoir computing with a chaotic circuit. In *ECAL 2017, Lyon, France*, pages 222–229. MIT Press

Kostitsyna I, Orponen P, editors (2021) *UCNC 2021 Espoo, Finland*, volume 12984 of *LNCS*. Springer

Kubota T, Takahashi H, Nakajima K (2021) Unifying framework for information processing in stochastically driven dynamical systems. Phys Rev Res 3(4):043135

Landauer R (1991) Information is Physical. Phys Today 44(5):23–29

Legenstein R, Maass W (2007) Edge of chaos and prediction of computational performance for neural circuit models. Neural Netw 20(3):323–334

Lehman J, Clune J, Misevic D, Adami C, Beaulieu J, Bentley PJ, Bernard S, Belson G, Bryson DM, Cheney N, Cully A, Donciuex S, Dyer FC, Ellefsen KO, Feldt R, Fischer S, Forrest S, Frénoy A, Gagneé C, Le Goff L, Grabowski LM, Hodjat B, Keller L, Knibbe C, Krcah P, Lenski RE, Lipson H, MacCurdy R, Maestre C, Miikkulainen R, Mitri S, Moriarty DE, Mouret J-B, Nguyen A, Ofria C, Parizeau M, Parsons D, Pennock RT, Punch WF, Ray TS, Schoenauer M, Shulte E, Sims K, Stanley KO, Taddei F, Tarapore D, Thibault S, Weimer W, Watson R, Yosinksi J (2020) The surprising creativity of digital evolution: a collection of anecdotes from the evolutionary computation and artificial life research communities. Artif Life 26(2):274–306

Lehman J, Stanley KO (2008) Exploiting open-endedness to solve problems through the search for novelty. In *ALife XI, Boston, MA, USA*, pages 329–336. MIT Press

Lehman J, Stanley KO (2011) Abandoning objectives: evolution through the search for novelty alone. Evol Comput 19(2):189–223

Liang X, Tang J, Zhong Y, Gao B, Qian H, Wu H (2024) Physical reservoir computing with emerging electronics. *Nature Electronics*, pages 1–14

Liang X, Zhong Y, Tang J, Liu Z, Yao P, Sun K, Zhang Q, Gao B, Heidari H, Qian H, Wu H (2022) Rotating neurons for all-analog implementation of cyclic reservoir computing. Nat Commun 13(1):1549

Lloyd S (2005) Programming the Universe. Vintage

Love J, Mulkers J, Bourianoff G, Leliaert J, Everschor-Sitte K (2021) Task agnostic metrics for reservoir computing. arXiv: 2108.01512v1 [cs.LG]

Lukoševičius M (2012) A practical guide to applying echo state networks. In Montavon, G., Orr, G. B., and Müller, K.-R., editors, *Neural Networks: Tricks of the Trade*, number 7700 in LNCS, chapter 27, pages 659–686. Springer, 2nd edition

Mackey MC, Glass L (1977) Oscillations and chaos in physiological control systems. Science 197(4300):287–289

Manneschi L, Lin AC, Vasilaki E (2023) SpaRCe: Improved learning of reservoir computing systems through sparse representations. IEEE Trans Neural Netw Learn Syst 34(2):824–838

Manneschi L, Vidamour IT, Stenning KD, Gartside JC, Swindells C, Venkat G, Griffin D, Stepney S, Branford WR, Hayward T, Ellis MO, Vasilaki E (2024) Optimising network interactions through device agnostic models. arXiv:2401.07387 [cs.LG]

Mathews VJ, Lee J (1994) Adaptive algorithms for bilinear filtering. In *SPIE 2296, Advanced Signal Processing: Algorithms, Architectures, and Implementations V*, volume 2296

Miller JF, Downing K (2002) Evolution in materio: looking beyond the silicon box. In NASA/DoD Conference on Evolvable Hardware 2002:167–176

Mohid M, Miller JF, Harding SL, Tufte G, Lykkebø OR, Massey MK, Petty MC (2014a) Evolution-in-materio: A frequency classifier using materials. In *2014 IEEE International Conference on Evolvable Systems*, pages 46–53

Mohid M, Miller JF, Harding SL, Tufte G, Lykkebø OR, Massey MK, Petty MC (2014b) Evolution-in-materio: Solving bin packing problems using materials. In *2014 IEEE International Conference on Evolvable Systems*, pages 38–45

Mohid M, Miller JF, Harding SL, Tufte G, Lykkebø OR, Massey MK, Petty MC (2014c) Evolution-In-Materio: Solving Machine Learning Classification Problems Using Materials. In *PPSN XIII*, pages 721–730. Springer

Momeni A, Rahmani B, Scellier B, Wright LG, McMahon PL, Wanjura CC, Li Y, Skalli A, Berloff NG, Onodera T, Oguz I, Morichetti F, del Hougne P, Gallo ML, Sebastian A, Mirhoseini A, Zhang C, Marković D, Brunner D, Moser C, Gigan S, Marquardt F, Ozcan A, Grollier J, Liu AJ, Psaltis D, Alù A, Fleury R (2024) Training of physical neural networks. arXiv: 2406.03372 [physics.app-ph]

Nakajima K (2020) Physical reservoir computing–an introductory perspective. Jpn J Appl Phys 59(6):060501

Nakajima K, Fischer I editors (2021) *Reservoir Computing: Theory, Physical Implementations, and Applications*. Springer

Nakajima K, Hauser H, Kang R, Guglielmino E, Caldwell DG, Pfeifer R (2013) A soft body as a reservoir: case studies in a dynamic model of octopus-inspired soft robotic arm. Front Comput Neurosci 7:91

Nakajima M, Inoue K, Tanaka K, Kuniyoshi Y, Hashimoto T, Nakajima K (2022) Physical deep learning with biologically inspired training method: gradient-free approach for physical hardware. Nat Commun 13(1):7847

NCEI (n.d.). National Centers for Environmental Information. Solar Indices. https://www.ngdc.noaa.gov/stp/solar/solar-indices.html. Accessed: 2024-04-28

Nowshin F, Zhang Y, Liu L, Yi Y (2020) Recent Advances in Reservoir Computing With A Focus on Electronic Reservoirs. In *2020 11th International Green and Sustainable Computing Workshops (IGSC)*, pages 1–8. IEEE

Pathak J, Hunt B, Girvan M, Lu Z, Ott E (2018) Model-free prediction of large spatiotemporally chaotic systems from data: a reservoir computing approach. Phys Rev Lett 120(2):024102

Rodan A, Tiňo P (2011) Minimum complexity echo state network. IEEE Trans Neural Netw 22(1):131–144

Roy O, Vetterli M (2007) The effective rank: A measure of effective dimensionality. In *2007 15th European Signal Processing Conference*, pages 606–610

Schrauwen B, Wardermann M, Verstraeten D, Steil JJ, Stroobandt D (2008) Improving reservoirs using intrinsic plasticity. Neurocomputing 71(7):1159–1171

Shannon CE (1941) Mathematical theory of the differential analyzer. J Math Phys 20(1–4):337–354

Stepney S (2019) Co-designing the computational model and the computing substrate. In *UCNC 2019, Tokyo, Japan*, volume 11493 of *LNCS*, pages 5–14. Springer

Stepney S (2021) Non-instantaneous information transfer in physical reservoir computing. In Kostitsyna and Orponen 2021:164–176

Stepney S, Kendon V (2021) The representational entity in physical computing. Nat Comput 20(2):233–242

Stepney S, Rasmussen S, Amos M (eds) (2018) Computational matter. Springer, New York

Suárez LE, Mihalik A, Milisav F, Marshall K, Li M, Vértes PE, Lajoie G, Misic B (2024) Connectome-based reservoir computing with the conn2res toolbox. Nat Commun 15(1):656

Tanaka G, Yamane T, Héroux JB, Nakane R, Kanazawa N, Takeda S, Numata H, Nakano D, Hirose A (2019) Recent advances in physical reservoir computing: a review. Neural Netw 115:100–123

Tsunegi S, Kubota T, Kamimaki A, Grollier J, Cros V, Yakushiji K, Fukushima A, Yuasa S, Kubota H, Nakajima K, Taniguchi T (2023) Information processing capacity of spintronic oscillator. *Advanced Intelligent Systems*, 2300175

Turing AM (1937) On Computable Numbers, with an Application to the Entscheidungsproblem. *Proceedings of the London Mathematical Society*, s2-42(1):230–265

Vidamour IT, Ellis MOA, Griffin D, Venkat G, Swindells C, Dawidek RWS, Broomhall TJ, Steinke NJ, Cooper JFK, Maccherozzi F, Dhesi SS, Stepney S, Vasilaki E, Allwood DA, Hayward TJ (2022) Quantifying the computational capability of a nanomagnetic reservoir computing platform with emergent magnetisation dynamics. *Nanotechnology*, 33(48)

White OL, Lee DD, Sompolinsky H (2004) Short-term memory in orthogonal neural networks. Phys Rev Lett 92(14):148102

Wright LG, Onodera T, Stein MM, Wang T, Schachter DT, Hu Z, McMahon PL (2022) Deep physical neural networks trained with backpropagation. Nature 601(7894):549–555

Wringe C, Stepney S, Trefzer M (2024) Reservoir computing benchmarks: a review. (submitted)

Zolfagharinejad M, Alegre-Ibarra U, Chen T, Kinge S, van der Wiel WG (2024) Brain-inspired computing systems: a systematic literature review. Eur Phys J B 97(6):1–23