Simulating Neuromorphic Reservoir Computing: Abstract Feed-forward Hardware Models

Aaron Stockdill* and Kourosh Neshatian†
Department of Computer Science
and Software Engineering
University of Canterbury
Christchurch, New Zealand
Email: *aas75@uclive.ac.nz and †kourosh.neshatian@canterbury.ac.nz

***Abstract*—Recent developments of unconventional hardware using memristors and atomic switch networks has led to renewed interest in hardware neuromorphic solutions. Most hardware models rely upon a reservoir neural network as the basis of any learning, but the distinct differences between software implementations and hardware reality mean what we take for granted in traditional software reservoirs—such as cycles, loops, infinite energy, and discrete time—may be severely limited or unavailable in hardware, raising questions about how a hardware implementation would perform and how to potentially overcome these limitations. Proposed hardware additions, such as an echoer or an input delay mechanism, address some of these limitations.**

I. Introduction

A human brain is able to perform better than modern computers at deceptively "simple" tasks such as object recognition, while also using in the order of a millionth of the power: hence the allure of machines that can match it. Recent advances in neuromorphic computing have meant renewed interest in the hopes of building such a machine [11]. A significant cause of this sudden explosion of popularity is the discovery of the memristor [17], a fourth fundamental circuit component that links flux and charge, and behaves as a variable resistor that depends on the history of inputs first theorised by Chua in 1971 [5]. Thus the promise of low-power, intelligent computing has reappeared.

Because the memristor has a state associated with it, they lend themselves to learning temporal sequences. Current research has looked into existing machine learning techniques which were themselves designed to operate with time-series datasets, and adapt them for use with memristive hardware. Using hardware, and thus an inherently *fixed* system, as the basis of computing means a recurrent neural network called the *reservoir neural network* is commonly considered appropriate. In this paper we present a flexible simulation of neuromorphic hardware as a reservoir in reservoir neural networks.

A prominent model of reservoir neural network is the *echo state network* (ESN) described by Jaeger in 2001 [7]. Jaeger defines the ESN by a reservoir of neurons connected by synapses with weights that remain unaltered, in contrast to trained artificial neural networks, and instead restricts all the learning to a readout layer. Because an ESN is considered to be the more "approachable" definition of a reservoir neural network, that is the model we use in this paper. Section II provides a thorough description of ESNs, while Section III has a discussion on using memristors in reservoirs.

978-1-5386-4276-4/17/$31.00

Another model of reservoir neural networks is the *liquid state machine* (LSM) devised by Maass in 2002 [10]. Maass uses the analogy of a liquid with *ripples*, rather than an echo property. Ripples are analogous to memory, because although the input has finished, the ripples continue, uniquely encoding the past inputs. There are two important features that an LSM must have: the separation property, and the approximation property. The separation property ensures that a unique history of input maps to a unique liquid state, while the approximation property ensures that the readout layer is capable of approximating the desired continuous function.

Reservoir neural networks place no restrictions on how the neurons connect. This gives the network an implicit memory, where past inputs entirely define the current state. This is the *echo property* in ESNs. The structure of the reservoir is known to have significant effects on the learning capabilities of the system [16], and in this paper we will frame these structural effects in terms of hardware limitations, with a particular focus on atomic switch networks.

II. Echo state networks

An echo state network (ESN) is a recurrent neural network first described by Jaeger in 2001 [7] that uses a reservoir of neurons that do not need training. Instead, the training happens in a readout layer. A more complete summary of ESNs is available from Lukoševičius [9], but Equation set (1) defines them.

$$\begin{aligned} \mathbf{y}(t) &= \mathbf{W}^{\text{out}}[1; \mathbf{u}(t); \mathbf{x}(t)] \\ \mathbf{x}(t) &= (1-\alpha) \times \mathbf{x}(t-1) + \alpha \times \tilde{\mathbf{x}}(t) \\ \tilde{\mathbf{x}}(t) &= \tanh\left(\mathbf{W}^{\text{in}}[1; \mathbf{u}(t)] + \mathbf{W}\mathbf{x}(t-1)\right) \end{aligned} \tag{1}$$

The readout layer $\mathbf{W}^{\text{out}}$ is the least-squares linear map from the input vectors $\mathbf{u}(t)$ to the output vectors $\mathbf{y}(t)$, based on the internal state of the network $\mathbf{x}(t)$. The operator $[\cdot;\cdot]$ denotes vertical vector concatenation. The constant α is the leaking rate, referring to the mixing between current and previous output.

The ESN's readout layer is typically trained using a linear model, leading to a simpler recurrent neural network than otherwise possible with techniques such as back-propagation through time, or extended Kalman filters, while retaining expressiveness. The linear model is for the ESN author to choose, but a common and recommended choice is ridge regression [9]. This is the technique we will use in this paper, using regularisation hyperparameter λ:

$$\mathbf{W}^{\text{out}} = \mathbf{Y}^{\text{target}}\mathbf{X}^{\mathrm{T}}\left(\mathbf{X}\mathbf{X}^{\mathrm{T}} + \lambda\mathbf{I}\right)^{-1} \tag{2}$$

The matrix $\mathbf{Y}$ is the sequence of input vectors arranged horizontally, while the matrix $\mathbf{X}$ is the sequence of $[1; \mathbf{u}(t); \mathbf{x}(t)]$ vectors arranged horizontally.

The stateful nature of ESNs, and their simplified training, means they have frequent applications in computer vision, time-series prediction, and signal processing. As their use grows, it is important to understand the inherent limitations of the paradigm, and understand the cause of any apparent weakening.

The reservoir used by an ESN is the source of, and restriction on, its memory capacity. Normally the only restriction placed on an ESN is the spectral radius, a measure of scaling performed by the reservoir. If the input could potentially contain a zero vector, then the spectral radius must be less than one [9]. Sometimes, the sparsity of the reservoir is also restricted, but this is for performance reasons. Beyond this, we usually have no reason to change the setup: random generation works well, being likely to construct a recurrent, connected graph of neurons.

But by placing further restrictions on the reservoir, we can reduce the "power" of the ESN. Čerňanský and Makula demonstrated this in their work exploring the *feed-forward ESN* [4]. By removing both cycles and loops in the reservoir, they show an ESN becomes equivalent to a feed-forward neural network with inputs representing up to n steps back in the input history, where n is the number of neurons in the reservoir.

The name "feed-forward ESN" does not imply a similar training or propagation process to a traditional neural network. Instead, this refers to the inherent direction of information in the reservoir, from input to output. That is, the graph defined by neuron connections is directed and acyclic, thus having a topological sort. We adopt the name feed-forward in this paper for a similar purpose.

III. Memristors and atomic switches

Memristor networks and atomic switch networks are a promising area of research for neuromorphic hardware. But because they are circuit components, they must obey Kirchhoff's Laws, so there are restrictions on how they behave.

A memristor is a fundamental circuit element linking flux Φ and charge Q, and so completing the relationships between current, voltage, flux, and charge. First theorised by Chua in 1971 [5], not until 2008 was the memristor found to exist by Strukov et al. [17], occurring naturally at nanometre scales.

A *flux-controlled* memristor is defined by two equations,

$$I = V \cdot G(x, V) \qquad \frac{dx}{dt} = f(x, V). \tag{3}$$

The family of devices that these equations permit are sensitive to the present and past inputs. This effectively grants the device a memory, hence the name is derived from "memory resistor." This equation also gives rise to the distinctive *pinched hysteresis* when plotting voltage against current, visible in Figure 1a.

As well as the abstract definition, we require a concrete realisation of a memristor to perform simulations. For this, we use the standard memristor as defined by Konkoli et al. [8],

$$\frac{dx}{dt} = \beta V + \frac{1}{2}(\alpha - \beta)(|V + V_T| - |V - V_T|) \quad G(x, V) = \frac{1}{x}.$$

This memristor will switch between low and high conductance over the threshold voltage V_T. The constants α (distinct from that in (1)) and β are physical properties unique to a device.

Atomic switches are another recent avenue for neuromorphic research. Because of their manufacturing process, they are much cheaper and easier to produce in large networks than standard memristors, and hence they are commonly used to study neuromorphic hardware. First becoming prominent in 2012, research first focused on silver nanowire approaches [1]. Although not the same as memristors, they share the key characteristic of naturally encoding their history into their current state. This is shown in Figure 1b, based on the percolating network technology from Fostner and Brown [6].

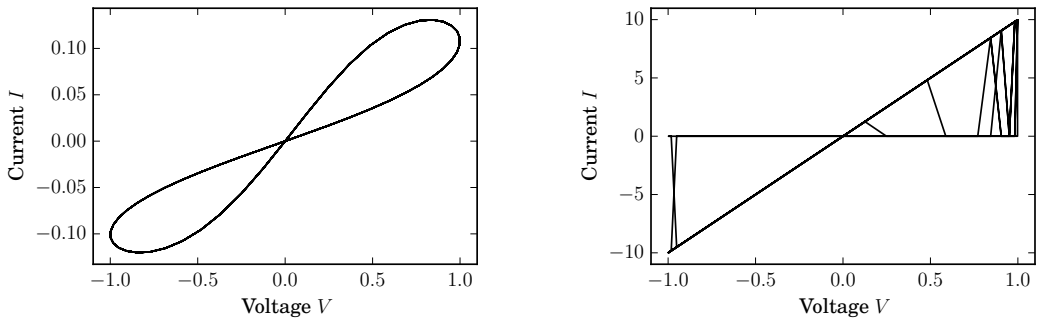
Work on atomic switches continues following successful initial work by Sillin et al. [14] showing potential neuromorphic abilities using a reservoir-style approach to machine learning. Burger and Teuscher also demonstrated how individual device variation would not impact the performance of the reservoir [2]. Competing atomic switch network architectures have emerged, such as the percolation networks from Sattar et al. [13].

IV. Neuromorphic circuits

An individual circuit component is not much good by itself. Despite the memory capabilities, memristors can only give an indication of whether the current input is near some average of the past inputs. However, just as the perceptron is limited when acting alone but more powerful when joined together, the memristor may too gain power when arranged into a network.

To explore these networks, we need to be able to efficiently create these component networks, and calculate the effect of passing a current and voltage over every component at once. To lay out the network of memristors and atomic switches, we build on the techniques developed by Fostner and Brown [6]. To calculate how the network behaves when applying current and voltage, we use a similar approach to Smith [15], which we outline shortly.

Both Fostner and Brown [6], and Smith [15], take the approach of simulating individual clusters, determining how they organise themselves in to groups, and calculating the distance between these groups. Because the actual location of groups is less important than their general distribution,



(a) Memristor

(b) Switch

Fig. 1: A pinched hysteresis generated by standard memristors, and atomic switches.

we take a statistical approach to particle deposition. We then randomly deposit the particle group centroids, and use a Delaunay triangulation to connect the centroids in a planar manner, and take the connection distances from a connection width distribution. Figure 2b shows the generated structure, where the edges are the memristive component: memristors or switches.

Once the network is set, we must be able to simulate electricity applied across it. Current and voltage are calculated through Kirchhoff's Circuit Laws. Kirchhoff's Voltage Law states that the directed potential differences around a circuit must sum to zero. We use this law to ensure that the potential difference between the input groups and the output groups is equal to the voltage applied to the input groups (that is, the network consumes all the voltage).

Kirchhoff's Current Law states that the sum of directed current at junction i is 0. Using the notation that $\mathcal{N}(\cdot)$ is a function that returns the neighbours of a given group, and that I_{ij} is the directed current flowing between groups i and j, we can write

$$\sum_{j \in \mathcal{N}(i)} I_{ij} = 0. \tag{4}$$

Because determining I_{ij} directly is difficult, we can use Ohm's Law and the potential difference between two groups, and set the input and output currents to be 1 A, appropriately signed. Thus we can rewrite Equation (4) as

$$I_i^{\text{in}} + I_i^{\text{out}} + \sum_{j \in \mathcal{N}(i)} G_{ij}(V_j - V_i) = 0, \tag{5}$$

which forms the basis of what Smith called the $\mathbf{G}$ matrix [15]. Figure 2a shows the structure of this $\mathbf{G}$ matrix.

V. Memory in hardware

An ESN has four sources of memory: leaking, cycles, loops, and the discrete time steps [16]. Previous work by Stockdill and Neshatian has shown that these properties provide different types of memory, and that not all are necessary. In this section we link these results to neuromorphic hardware, and how the limitations can be overcome. As in [16], we will not consider leaking any further as it is strictly outside the reservoir. Two distinctions remain between the fully-weakened ESN and a memristor network. First, the memristor network is updating the connection weights while the network is running. Second, the software reservoir is able to amplify and suppress energy arbitrarily, whereas hardware must obey Kirchhoff's Laws.

A. Loops

As shown by Čerňanský and Makula, removing both cycles and loops reduces an ESN to a feed-forward network with delayed-time inputs [4]. The memory of the network is limited by the longest chain. In neuromorphic hardware this is less of a concern, as the systems are often composed of a large number of units—in this case, groups.

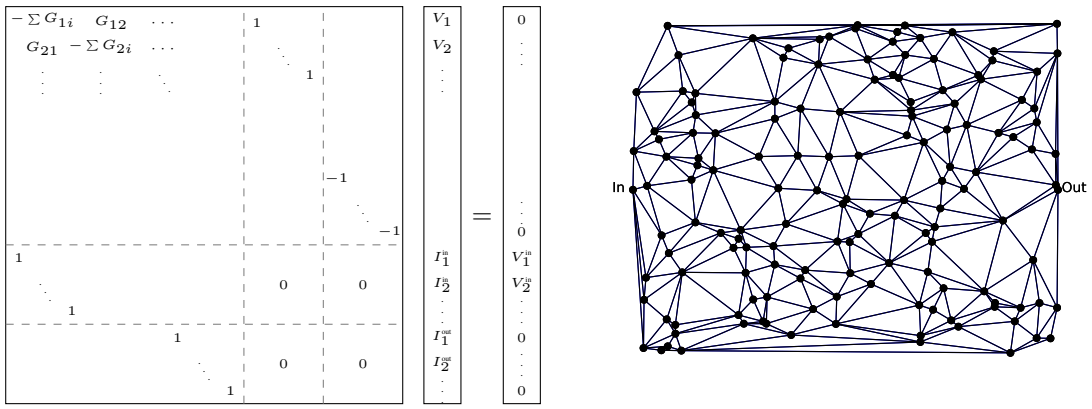
A loop is an unlikely physical phenomenon to occur naturally. Somehow, a single group in the network would have to feed back into itself. This is unrealistic, however hardware capable of repeating back voltage signals is possible. This hardware "echoing" would mimic the effect of a loop, letting the network potentially retain some of the reservoir power required.

B. Cycles

Kirchhoff's Current Law limits the amount of energy in a circuit, and forces conservation. That is, a junction is unable to amplify a signal. In a hardware circuit this means that the larger the reservoir, the smaller the share of the potential difference between each pair of groups. However, it also reveals there cannot be *cycles* in the network, because this would imply a cycle of groups where the potential difference drops forever, Equation (6), leading to an impossible infinitely-descending structure.

$$V_1 > V_2 > \cdots > V_k > V_1 > \cdots \implies V_1 > V_1 \lightning \tag{6}$$

Cycles are not something that can be added back to a hardware reservoir easily. There is ongoing discussion about whether switching events that happen on short enough timescales to occur during electron flow could generate the



(a) (b)

Fig. 2: (a) The structure of the $\mathbf{G}$ matrix used to solve Kirchhoff's Laws, from Smith [15]. (b) An example generated network, where nodes are the group centroids, and edges are the memristive component.

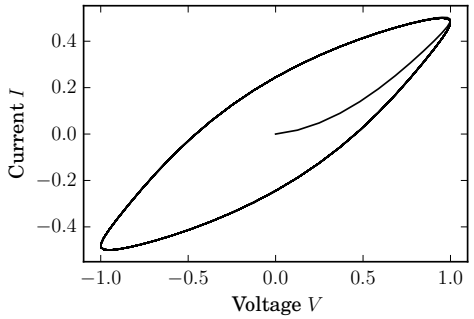


Fig. 3: The "IV" curve of an Echo State neuron. Note the lack of $(0, 0)$ intercept.

correct conditions for temporary cycles to form, however this is at best beyond current technology. For this reason, we are left building on potential "echoer" hardware to retain some memory.

C. Conservation of energy

Conserving energy in a reservoir neural network is not damaging to the system. It is equivalent to restricting the spectral radius of the reservoir to less than one, as required by the reservoir to work reliably [16].

An interesting consequence of using voltage as input is that the power source and the input signal are now linked. This is entirely different to how software reservoir neural networks behave, which are capable of producing output with no input. Figure 3 reveals this effect with an ESN with a single neuron, modelling the "current-voltage" with input and output signals. The "pinched hysteresis" so distinctive of the memristor disappears, and the $(0, 0)$ intercept is replaced by new intercepts which can be interpreted as no input producing output. This is because the signal and the power are completely unrelated. A possible solution is to use a different encoding to hardware reservoirs, and instead all inputs are mapped to a higher voltage to get around this. This would ensure that even for zero input, the network is still receiving power.

D. Discrete time steps

Software ESNs are by nature discrete-time, but this is physically unrealistic. Because a network of memristors will have the current pass through the network at significant fractions of the speed of light, no matter how rapidly we switch the input voltage, we are essentially saturating the network with the same signal millions of times before switching. Because of this speed disparity, a direct model of the hardware network will not contain discrete time steps, instead it will function more like a traditional feed-forward neural network, which we will call the one-hop reservoir, where the input $\mathbf{u}(t)$ is influencing the entire network at time t, but inputs $\mathbf{u}(s)$ from times $s < t$ are not in the network. There is no new information written to the network before the propagation is complete. Because of this distinct termination, the network is not allowed to have cycles or loops. The propagation algorithm is given in Algorithm 1.

The network is now unable to learn any function requiring knowledge of previous time steps. The state now depends solely on the random initial weights, not the history of previous inputs as required by an ESN—this network is now an untrained feed-forward neural network.

This comparison is not fair, because a network of memristors *does* maintain a state, because the weights *do* get updated. Unfortunately, a memristor's state cannot act act as a suitable substitute for the ESN's discrete time steps. The state will work *against* the readout layer, causing the connection strength between neurons to "wobble" over time.

By making the ESN have a "wobbling" weights matrix to simulate the updating conductances of the memristors and switches, we handicap the readout layer by removing the

Algorithm 1 Feed-forward Reservoir Propagation

▷ Propagate input $\mathbf{u}(t)$ over the reservoir defined by $\mathbf{W}$
procedure PROPAGATE($\mathbf{W}$, $\mathbf{W}^{\text{in}}, \mathbf{u}(t)$)
  $\mathbf{v} \leftarrow \mathbf{W}^{\text{in}}[1; \mathbf{u}(t)]$
  $\mathbf{o} \leftarrow (0, 0, \ldots, 0)^{\text{T}}$ ▷ $\mathbf{o}$ holds final value for output
  for all $n \in \text{toposort}(\mathbf{W})$ **do**
    $t \coloneqq \mathbf{v}_n$
    for all $m \in \text{predecessors}(n)$ **do**
      $t \coloneqq t + \mathbf{o}_m \times \mathbf{W}_{n,m}$
    end for
    $\mathbf{o}_n \leftarrow f(t)$ ▷ f is tanh or sigmoid
  end for
  return $\mathbf{o}$
end procedure

underlying assumption of regression—for a given input x, there is a function $f(x)$ that we attempt to find. Because f is a function, each x maps to a unique y. By changing the weights matrix, we change the x value for a given y,[1] and introduce the chance of a *collision*. That is, two different histories of input may result in the same internal state, and the readout layer is now unable to learn a function to map the state to the desired output. Another way of looking at this is that the "wobbling" ESN violates the separation property defined by Maass [10] in reference to the liquid state machine. The separation property states that, for two inputs a and b, there should be a filter F such that if a and b differ anywhere at time $t < T$, then $F(a) \neq F(b)$. The reservoir (or liquid) serves as this filter, but the wobbling reservoir is no longer appropriate.

VI. Empirical comparison of memristive state with discrete time conservative networks

By generating a large variety of memristor networks and feed-forward conservative ESNs that are both discrete-time and one-hop with a range of reservoir sizes, we can explore whether they all exhibit similar learning tendencies, and if not which networks behave most similarly. We generated ten reservoirs of each learner, with reservoir sizes ranging from 50 to 2000 neurons, trained the reservoir to predict the Mackey-Glass $\tau = 17$ problem set using the output at time $t-1$ as input at time t, and calculated the correlation distance between the output curve and the expected curve for the next 200 steps. The correlation distance between two curves represented by vectors $\mathbf{a}$ and $\mathbf{b}$ is

$$d_{\text{corr}}(\mathbf{a}, \mathbf{b}) = 1 - \frac{(\mathbf{a} - \bar{\mathbf{a}}) \cdot (\mathbf{b} - \bar{\mathbf{b}})}{\|\mathbf{a} - \bar{\mathbf{a}}\|_2 \|\mathbf{b} - \bar{\mathbf{b}}\|_2}. \tag{7}$$

The symbol $\bar{\mathbf{a}}$ is the mean value of $\mathbf{a}$, $\cdot$ is the dot product of vectors, and $\|\cdot\|_2$ is the euclidean length of a vector. This particular distance metric was chosen as it better captures the intent to follow a curve rather than how far apart two curves happen to be. Before running statistical tests, we throw out the "failed" learnings. We determine these by calculating the area between the output and expected curves. We consider any area that exceeds 10^{10} as a failed learning.

A one-way ANOVA test, where the grouping is the pair (learner, size), reveals there is a significant difference between the groups ($F_{24} = 6.62$, $p < 0.0001$). To see where these differences actually occur, we perform a Student's t-test between two learners at each size, and the result of this is in Table I. Because there are a large number of t-tests conducted, and a Bonferroni correction is too conservative, the chance of making a Type-I error rises. Hence we consider more the "broad strokes" rather than the precise p-values. The first thing that is clear is that distinguishing between the one-hop ESN and the memristor is difficult, with only one significant result out of all the reservoir sizes. This is in contrast to what occurs between the discrete-step ESN and both other learners, in particular memristors. We can distinguish the discrete-step ESN from either of the other learners with good consistency.

These differences become clear when we plot the predicted curve alongside the actual curves they were intended to match. The examples in Figure 4 show how each reservoir fares as a predictor, and makes clear why the memristors and one-hop ESNs are so difficult to tell apart. The correlation distances for each prediction are: 0.13638 for the discrete ESN; 1.30299 for the one-hop ESN; and 0.62263 for the memristor reservoir. The closer the two curves follow one another, the smaller the correlation distance between them.

VII. Moving forward

To overcome these hardware limitations, the reservoir designers can make some important alterations, and move the missing features to external hardware or software. An example of this would be the "echoer" mentioned earlier. Other options include an n-step delayed input, where the input size of the network is made n times larger, and input from times $t-1, t-2, \ldots, t-n$ are also input to the network. This removes the responsibility of storing past inputs from the network state to the external input-delay mechanism..

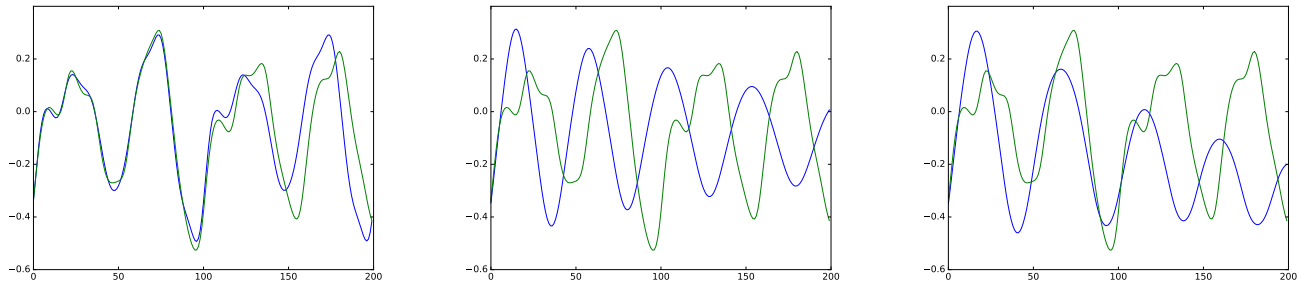
Another potential path forward is that taken by Bürger et al. [3], in composing a collection of memristive reservoirs into a "reservoir of reservoirs." This architecture moves the concerns of loops and cycles out to a digital reservoir, meaning the limitations of an electrical circuit no longer apply. It also means that a potential limitation of large networks is mitigated. When the same potential difference is spread over many memristors, there is a smaller potential difference across each memristor, so the switching is less effective.

Alternatively, instead of general-purpose machine learning, hardware designers can pursue an approach more like that of Pershin and Di Ventra [12]. The arrangement of memristors is no longer random, but carefully chosen to solve a particular problem, or class of problems. The example they explore is maze-solving, where by arranging the memristors in a grid pattern and enabling and disabling certain paths the shortest path is naturally found by the circuit. This is still a useful application of memristors, but can no longer solve problems that cannot be modelled with that particular circuit layout.

[1] That is, at time t_1 we have $x_1 \mapsto y$, but at time t_2 we find that $x_2 \mapsto y$ and $x_1 \not\mapsto y$. The function f has changed.

TABLE I: Significance levels between distributions of correlation distances for different learners. Stars signify 95%, 99% and 99.9% confidence intervals.

	Discrete vs One-hop		Discrete vs Memristor		One-hop vs Memristor	
Size	p	Sig.	p	Sig.	p	Sig.
50	0.0160	*	0.0001	***	0.3381	
100	0.1668		0.0054	**	0.2224	
200	0.0190	*	< 0.0001	***	0.0593	
500	0.0440	*	< 0.0001	***	0.0185	*
750	0.0291	*	0.0001	***	0.1068	
1000	0.0022	**	< 0.0001	***	0.4198	
1500	0.0036	**	0.0003	***	0.7328	
2000	0.0237	*	0.0006	***	0.3445	



(a) Discrete ESN (b) Onehop ESN (c) Memristor Reservoir

Fig. 4: The predicted Mackey-Glass curves, blue (dark grey), plotted against the true curves, green (light grey), for each type of reservoir.

Finally, we must consider alternatives to the reservoir paradigm. Although the limitations of updating a hardware reservoir make similarities clear, it obscures many fundamental differences in the assumptions of their development. Loops, cycles, and the discrete time steps are all vital features to the representational power of ESNs and LSMs and need to be built into the next generation model of neuromorphic networks. While still under development, in the latest hardware implementations of reservoir neural networks cycles, loops, and discrete time steps are present due to intrinsic capacitive time delays or implemented through external circuitry.

VIII. Conclusion

Reservoir neural networks such as echo state networks or liquid state machines were designed to run in software, which allows enormous flexibility in, for example, the type and number of connections between nodes. Hardware variants have intrinsic limitations because they obey the laws of Physics. This means a direct translation of reservoir neural networks onto neuromorphic hardware may suffer from reduced representational power compared to their software counterparts.

Potential hardware additions such as the input delayer hint that the hardware implementation of neural networks is still viable, but a new, more creative approach may be needed. Further research is needed to explore how to overcome the lack of cycles, loops, and discrete time steps in circuits, and whether there are possible replacements. Other ways to learn from hardware reservoirs may present themselves with a different set of restrictions to those of echo state networks or liquid state machines.

References

[1] A. V. Avizienis, H. O. Sillin, C. Martin-Olmos, H. H. Shieh, M. Aono, A. Z. Stieg, and J. K. Gimzewski, "Neuromorphic atomic switch networks," *PLoS ONE*, vol. 7, no. 8, pp. 1–8, August 2012.

[2] J. Bürger and C. Teuscher, "Variation-tolerant computing with memristive reservoirs," in *IEEE/ACM International Symposium on Nanoscale Architectures*, July 2013, pp. 1–6.

[3] J. Bürger, A. Goudarzi, D. Stefanovic, and C. Teuscher, "Hierarchical composition of memristive networks for real-time computing," in *Proceedings of the 2015 IEEE/ACM International Symposium on Nanoscale Architectures (NANOARCH15)*. IEEE, July 2015, pp. 33–38.

[4] M. Čerňanský and M. Makula, "Feed-forward echo state networks," in *Proceedings. 2005 IEEE International Joint Conference on Neural Networks, 2005.*, vol. 3, July 2005, pp. 1479–1482.

[5] L. O. Chua, "Memristor – the missing circuit element," *IEEE Transactions on Circuit Theory*, vol. 18, no. 5, pp. 507–519, September 1971.

[6] S. Fostner and S. A. Brown, "Neuromorphic behavior in percolating nanoparticle films," *Phys. Rev. E*, vol. 92, no. 5, p. 052134, November 2015.

[7] H. Jaeger, "The "echo state" approach to analysing and training recurrent neural networks," German National Research Institute for Computer Science, GMD Report 148, January 2001.

[8] Z. Konkoli and G. Wendin, "On information processing with networks of nano-scale switching elements," *International Journal of Unconventional Computing*, vol. 10, no. 5/6, pp. 405–428, November 2014.

[9] M. Lukoševičius, *Neural Networks: Tricks of the Trade: Second Edition*. Springer, 2012, vol. 7700, ch. A Practical Guide to Applying Echo State Networks, pp. 659–686.

[10] W. Maass, T. Natschläger, and H. Markram, "Real-time computing without stable states: A new framework for neural computation based on perturbations." *Neural Computation*, vol. 14, no. 11, pp. 2531 – 2560, November 2002.

[11] D. Monroe, "Neuromorphic computing gets ready for the (really) big time," *Commun. ACM*, vol. 57, no. 6, pp. 13–15, June 2014.

[12] Y. V. Pershin and M. Di Ventra, "Solving mazes with memristors: A massively parallel approach," *Phys. Rev. E*, vol. 4, no. 84, p. 046704, March 2011.

[13] A. Sattar, S. Fostner, and S. A. Brown, "Quantized conductance and switching in percolating nanoparticle films," *Physical Review Letters*, vol. 111, no. 13, p. 136808, June 2013.

[14] H. O. Sillin, R. Aguilera, H.-H. Shieh, A. V. Avizienis, M. Aono, A. Z. Stieg, and J. K. Gimzewski, "A theoretical and experimental study of neuromorphic atomic switch networks for reservoir computing," *Nanotechnology*, vol. 24, no. 38, p. 384004, September 2013.

[15] A. Smith, "Simulating percolating superconductors," Master's thesis, University of Canterbury, 2014.

[16] A. Stockdill and K. Neshatian, *Restricted Echo State Networks*, ser. LNAI. Cham: Springer International Publishing, December 2016, vol. 9992, pp. 555–560.

[17] D. B. Strukov, G. S. Snider, D. R. Stewart, and R. S. Williams, "The missing memristor found," *Nature*, vol. 453, no. 7191, pp. 80–83, May 2008.