Leveraging device variation to strengthen memristor based reservoir computing

Samip Karki[1], Sundar Kunwar[1], Francesco Caravelli[2], & Aiping Chen[1]

The conventional Von Neumann computer architecture is showing to be inadequate to handle the increasing demand for artificial neural networks in our society. Inspired by the human brain, the field of neuromorphic computing seeks to create hardware-based neural networks to bypass the bottleneck of software-based computation. Memristors, for their electrical memory properties, are one popular candidate for hardware implementation of neuromorphic computing. One type of neuromorphic computing scheme is reservoir computing, a recurrent neural network architecture which specializes in temporal tasks and is easily trained using linear regression. In this paper, I successfully implement reservoir computing using a delay-feedback system and numerically simulated memristor dynamics to perform a chaotic time series prediction task and show that the reservoir network can be expanded by adding unique devices in parallel.

[1]Center for Integrated Nanotechnologies (CINT), Los Alamos National Laboratory, Los Alamos, NM 87545, USA
[2] Physics of Condensed Matter & Complex Systems, Los Alamos National Laboratory, Los Alamos, NM 87545, USA

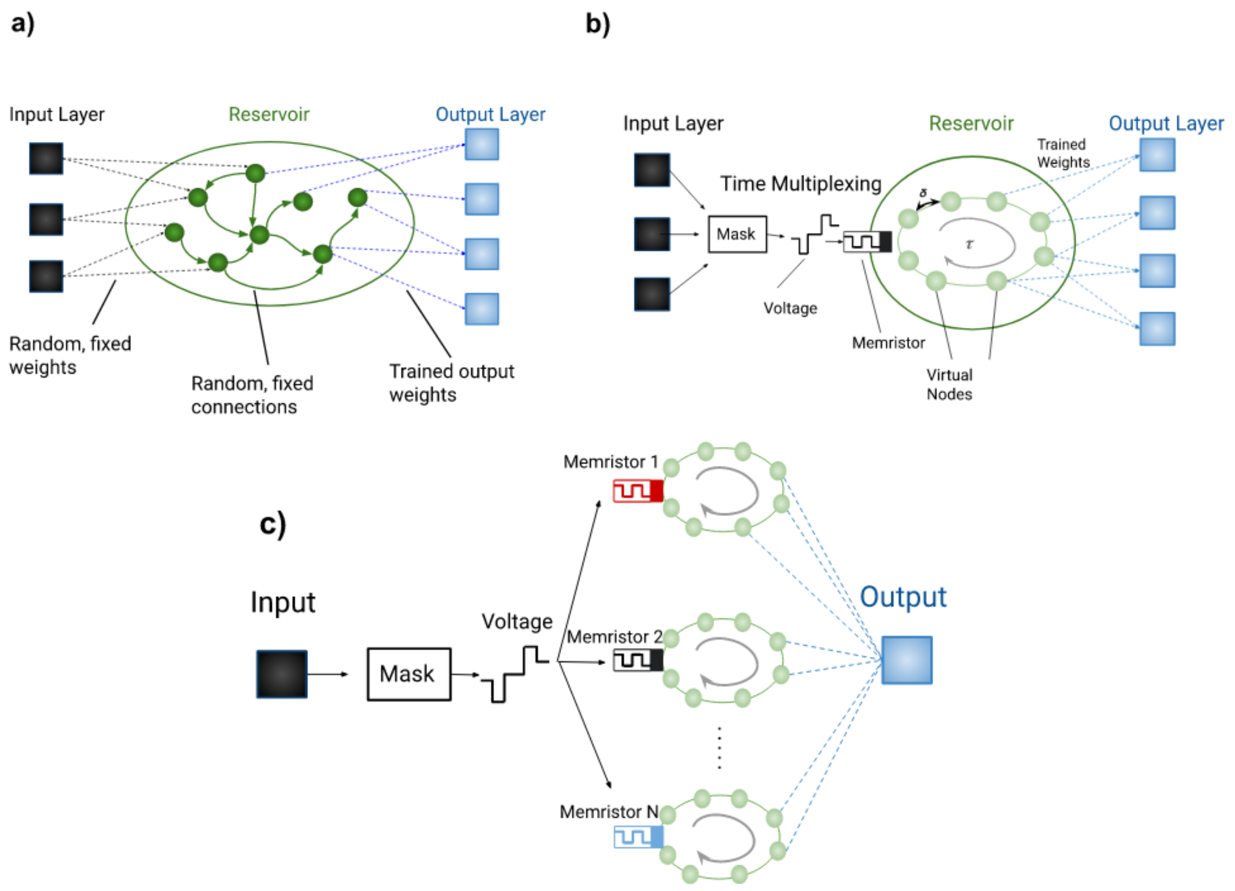

Figure 1| Reservoir computing architecture. a) Schematic of the traditional reservoir computing system. Input layer is connected to the reservoir, where nonlinear reservoir nodes have fixed but random weights and connections. The output is a linear weighted sum of the reservoir nodes. **b)** Schematic of memristor-based time delay reservoir computing system. With time multiplexing, the input is transformed into a temporal voltage input stream of length τ. This voltage is applied to the memristor. The reservoir is made up of M virtual nodes which correspond to the memristor's response at every interval δ, where $\tau = \delta \cdot M$. Connections between the virtual nodes of the reservoir and the output layer are the same as in the traditional RC system. **c)** Schematic of extended time-delay RC system. The voltage generated from the time multiplexing process is applied to different memristors in parallel. The output is a weighted linear combination of all virtual nodes.

INTRODUCTION

Artificial neural networks (ANNs) are powerful algorithms inspired from biological neural networks. Just as children learn to walk, speak language, and identify objects in their surroundings by repetitive experiences, the same principle allows artificial neural networks to solve a variety of different tasks. ANNs can be thought up as mathematical functions which do a series of operations on some input. These operations have adjustable parameters called weights that control the magnitude of the operations. During training, ANNs can learn by incrementally adjusting their weights so that the network output approaches the desired output of training data. In this way, ANNs can be taught to learn to solve a variety of problems.

Today in our data driven world, ANNs are being used to identify objects in images, translate spoken words into text, and much more. The use of ANNs will only increase into the future, however, limitations of computational efficiency hinder their scalability, flexibility, and sustainability. Currently, the power required to train large neural networks such as the GPT-4 language model is on the order of magnitude of several kilowatts[1]. This much power consumption is not feasible for the research and development of ANNs in offline applications such as in the use of phones, mobile robots, and self-driving cars. In addition, the energy usage of ANNs has a significant global environmental cost[2]. A primary reason for the large amount of power being used by ANNs is that digital computers use the von Neumann architecture which separates computation and memory storage. This so called von Neumann bottleneck fundamentally limits the speed of computation and is a major inefficiency when training large ANNs, as the networks must use large amounts of computational resources to move millions or even billions of weights between memory and CPU. To resolve this bottleneck issue, researchers have looked at the human brain for answers.

In contrast to the several kilowatts of power required to train large ANNs, the human brain only uses on average $20W$ of power, yet it is still able to easily perform tasks which would be difficult for ANNs such as image recognition and speech recognition[1]. For this reason, the field of neuromorphic computing wishes to implement aspects of the human brain into ANNs for more efficient computing. A characteristic of neuromorphic computing is in-memory computing, which typically requires doing the computation in a physical hardware or analog system.

Reservoir Computing (RC) is an example of neuromorphic computing which takes influence from the recurrent dynamics of the brain. The recurrent connections of RC make it particularly well suited for temporal tasks, such as time series prediction and voice recognition, tasks which can be difficult for traditional feedforward networks. A schematic of the traditional RC system is given in figure 1a). The RC network is made up of three parts, the input layer, the reservoir which is made up of random but fixed connections, and the output layer. RC works by nonlinearly mapping the input to a higher dimensional reservoir space which can be used to discern features of the input. While other recurrent neural network frameworks can be difficult to train with traditional methods like gradient descent, training in RC is simple because only the weights connecting the reservoir to the output are trained, reducing training to a linear task. In addition, reservoirs have a fading memory, meaning the state of the reservoir is dependent on the state in the recent past but not the far past. This fading memory is why RC is successful at time series prediction. The RC network has memory of multiple past iterations of the time series even when the input may only be a single iteration. Reservoir computers have been implemented in a handful of physical systems such as photonics, spintronic oscillators, and memristors. Because of their resistive switching memory properties, memristors are a popular candidate for implementation of reservoir computers and neuromorphic computing in general.

Memristors are nanoelectronic circuit devices which exhibit memory properties. The electronic resistance of a memristor is dependent on the voltage applied to it in the past. For example, applying two $3V$ pulses to a memristor may result in a $30\mu A$ response during the first pulse and a $35\mu A$ response during the second pulse, because the resistance of the memristor changed as a result of the first pulse. Because of these memory properties, memristors are a popular topic of research in the neuromorphic computing field.

Previous works[4,5] have explored using memristor based RC with a time-delay system, shown in figure 1 b). These are a popular way of implementing RC systems because the hardware implementation is very simple, often only needing a single memristor and a voltage source along with the appropriate software for data processing. To extend the reservoir further, additional voltage sources and memristors are added in parallel[4,5]. However, every additional voltage source added to the system further complicates the hardware and increases energy consumption of the whole system. Instead, we

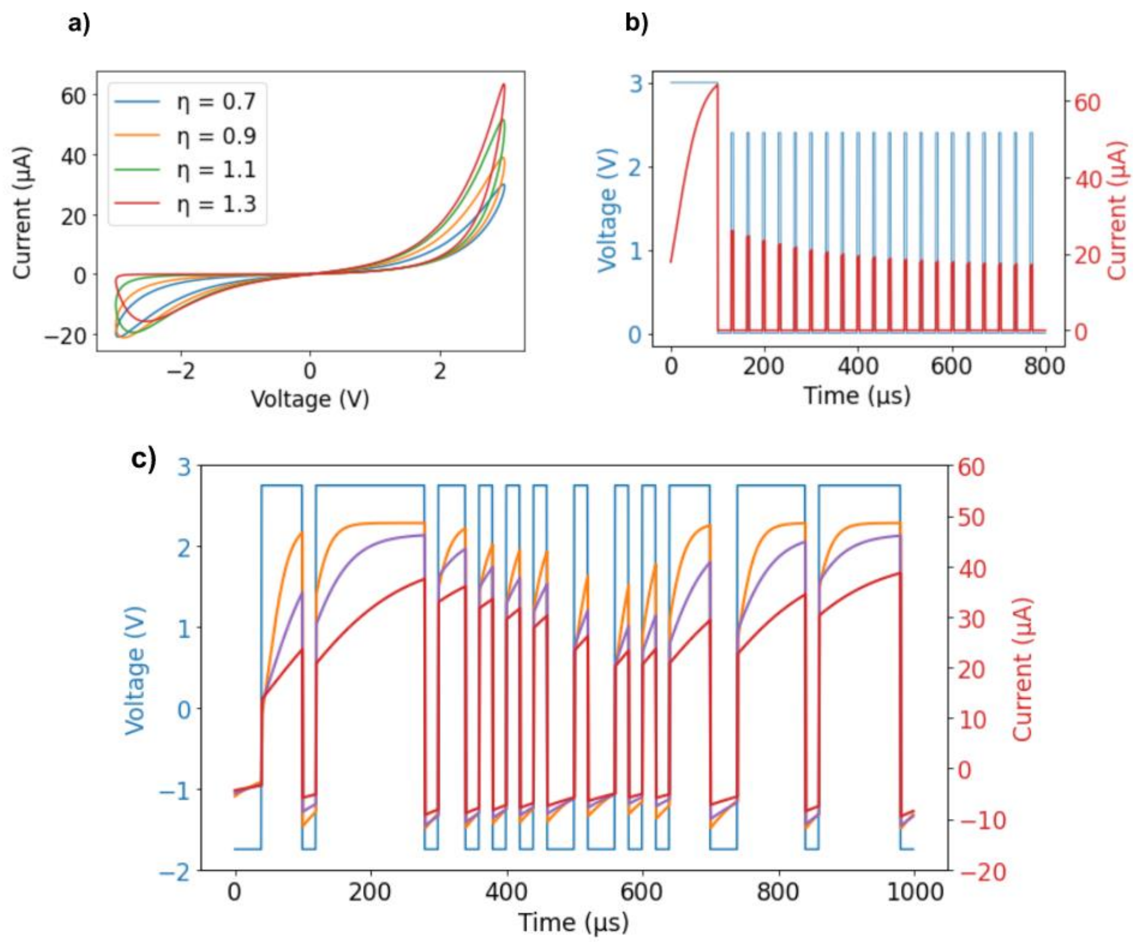

Figure 2| Simulated Memristor Dynamics and Device Variation. a) Simulated I-V hysteresis curves of memristor model with different values of parameter η. **b)** Demonstration of short-term memory of simulated memristor. Input voltage sequence and current response are in blue and red, respectively. Initial write voltage (3V, 100μs) is applied to perturb the system away from the equilibrium state. Several read pulses of (2.4V, 5μs) outline the gradual relaxation of the memristor model back to equilibrium. **c)** Device response variation during reservoir computing. Input voltage sequence (blue) is generated from time multiplexing during Hénon map prediction task. Three devices with different values of parameter η (orange η = 1.3, purple η = 1.0, red η = 0.7) each have a distinct current response.

propose extending the time delay RC system by simply applying a single voltage source to several memristors in parallel, with each memristor having unique dynamics. In this way, we balance increasing the dimensionality of the reservoir and improve performance while keeping the overall system low energy and simple to implement. We test the performance of our extended time-delay RC system with two chaotic time series tasks.

PROCEDURE

A. Delay-Feedback RC

Time-delay RC system with a single delay line, as shown in figure 1 b), uses the dynamic response of a single nonlinear node, in our case a memristor, to extract features from the input. For the case of time series prediction, the input to the network is a single scalar value $u(n)$, the n^{th} value of the time series, and the output is the prediction of $u(n+1)$. With a process called time multiplexing (methods), the input $u(n)$ is used to generate a sequence of $\boldsymbol{M}$ voltage pulses with binary values which is applied to the memristor. The values of the memristor response after each voltage pulse are the M virtual nodes of

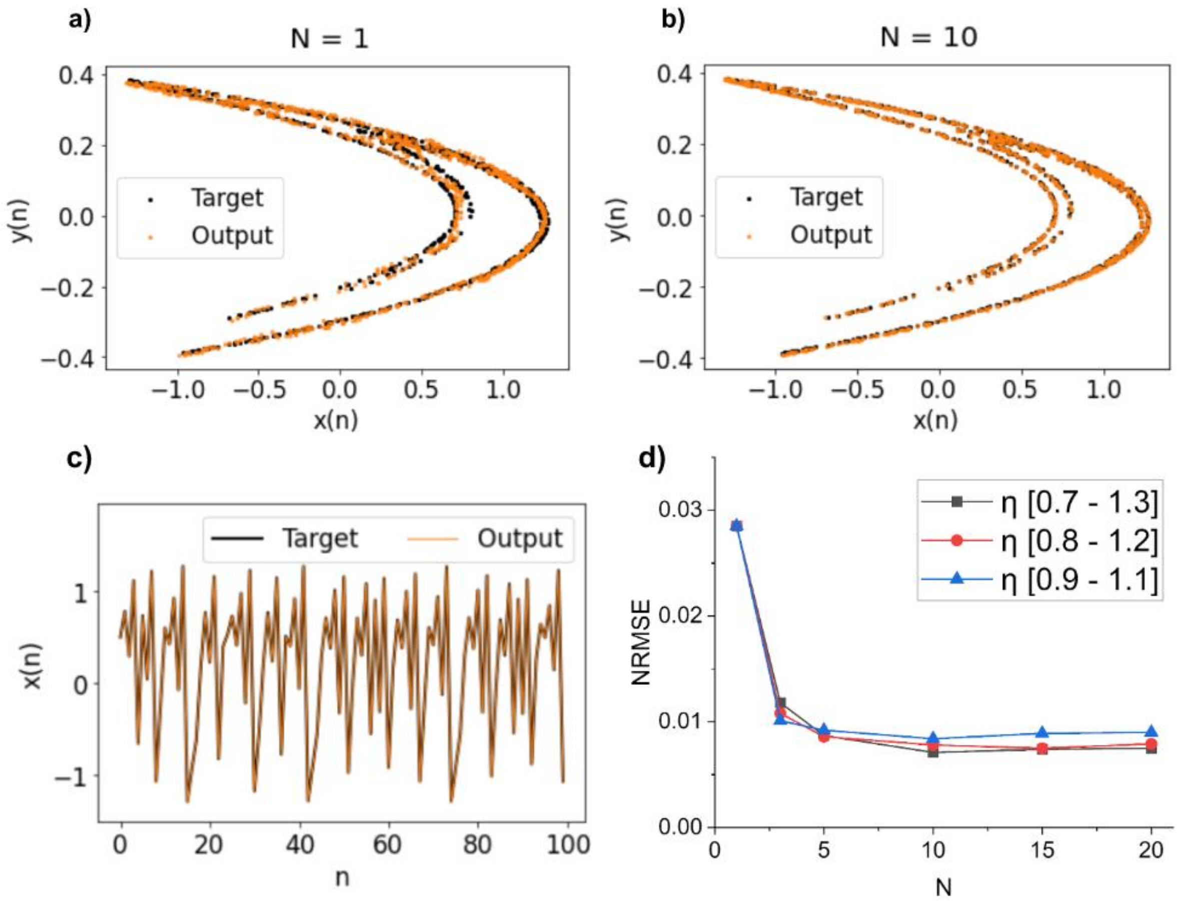

Figure 3| Performance of Hénon Map Prediction with Impact of Device Variation. a) and **b)** are 2-D representations of the predicted results of Hénon map task where the target is in black and the RC output is in orange. **a)** shows the results of a single memristor RC system, which had a NRMSE of 0.0279. **b)** shows the result as an RC network with 10 memristors in parallel, where parameter η was linearly spaced between [0.7 - 1.3], achieving a NRMSE of 0.0082. Other parameters are set to be $M = 30$, $V_{max} = 3.0V$, $V_{min} = 2.0V$, and $\delta = 15\mu s$. **c)** Performance of the 10 memristor RC network from **b)** with the predicted sequence in orange and the target sequence in black. **d)** Performance improves as more memristors are added in parallel. To consider networks with optimized masks, each point represents the best performance from 30 simulations each with randomized masks. Memristors in parallel each had different parameters η which were linearly spaced between [0.7 - 1.3] (black), [0.8 - 1.2] (red), or [0.9 - 1.1] (blue). The same training sequence was used in every simulation.

the reservoir, which completes the nonlinear transformation of the input. These virtual nodes are coupled to each other because of the memory dynamics of the memristor. The network is trained by finding the correct weight matrix $\boldsymbol{W}$ which maps the virtual nodes to a prediction of $u(n+1)$ (Methods).

The reservoir is extended by having multiple devices in parallel as shown in figure 1c). Each memristor will receive the same voltage sequence but generate different responses. Using virtual nodes from all the memristors, we extend the dimensionality of the reservoir further than with a single device.

B. Chaotic Time Series Prediction

Two different chaotic time series are considered in this work. One is the Hénon map:

$$x(n+1) = y(n) - 1.4x(n)^2 \qquad (1)$$

$$y(n+1) = 0.3x(n) + w(n) \qquad (2)$$

where $w(n)$ is gaussian noise with mean value of 0 and standard deviation of 0.0025. The RC network predicts will predict $x(n+1)$ with $x(n)$ as input. The other time series is the Mackey-Glass oscillator:

$$\frac{dx}{dt} = \beta \frac{x(t-\tau)}{1+\left(x(t-\tau)\right)^n} - \gamma x(t) \qquad (3)$$

where parameters are set to $\gamma = 0.1$, $\tau = 18$, $n = 10$, and $\beta = 0.2$. To make this sequence discrete, the time series is made by sampling every 3 time-units of the $x(t)$.

2000 iterations of each time series are generated. The first half of these datasets are used for training the weights of the network, and the second half of these datasets are used to evaluate performance (Methods).

Performance of each prediction was measured with Normalized-Root-Mean-Square-Error (NRMSE):

$$NRMSE = \sqrt{\frac{\left\langle (\mathbf{WS} - \mathbf{Y})^2 \right\rangle}{\left\langle (\mathbf{Y} - \overline{\mathbf{Y}})^2 \right\rangle}} \qquad (4)$$

where $(\cdot)^2$ is elementwise, $\langle \cdot \rangle$, is the mean of the vector, **S** is a matrix where the column are the virtual nodes generated for each $u(n)$, **Y** is a vector of the targets, and $\overline{\mathbf{Y}}$ is the mean of the targets.

C. Memristor Model

The following numerical model was used to simulate the dynamics of a memristor device from *Zhong et al.* (2021):

$$\frac{dw}{dt} = \lambda R(w) \sinh\left(\eta V(t)\right) - \frac{w(t) - w_0}{\kappa} \qquad (5)$$

$$I(t) = \gamma w(t)^2 \sinh\left(d\, V(t)\right) \qquad (6)$$

where w is the state variable, $V(t)$ is the input voltage, $I(t)$ is the current response, $R(w)$ is a window function, and γ, d, λ, η, κ are all adjustable parameters. A detailed description of the model is given in "methods".

In this work, we are interested in exploring how variation of devices in parallel affects the network performance. To model different devices, the parameter η was varied between 0.7 and 1.3. Figure 2a) illustrates how when performing a voltage sweep, the shape of the hysteresis loop differs when the parameter η is changed. When η is increased, the device conductance becomes more sensitive to the applied voltage. During the time multiplexing algorithm, each device will receive the same input voltage train and generate a unique response, figure 2c). Each memristor response will have a unique set of virtual nodes. By including all the virtual nodes in our reservoir, the dimensionality of the reservoir is increased far beyond what it would be with a single device. Our model also incorporates short-term memory dynamics of volatile memristors which is crucial for RC. Figure 2b) illustrates the short-term memory relaxation of the memristor model after a large initial voltage pulse.

RESULTS

Figure 3 shows the performance of the reservoir computing network during the Hénon Map prediction task. The side-by-side comparison of figure 3a) and figure 3b) show visually that the network with 10 unique devices in parallel was able to perform better than the single device. The 10-device network was able to achieve a NRMSE of 0.0082 compared to the single device which only achieved 0.0279. Figure 3d) shows that NRMSE decreases as the number of devices in parallel is increased. In addition, performance was slightly better when the interval of parameter η was increased,

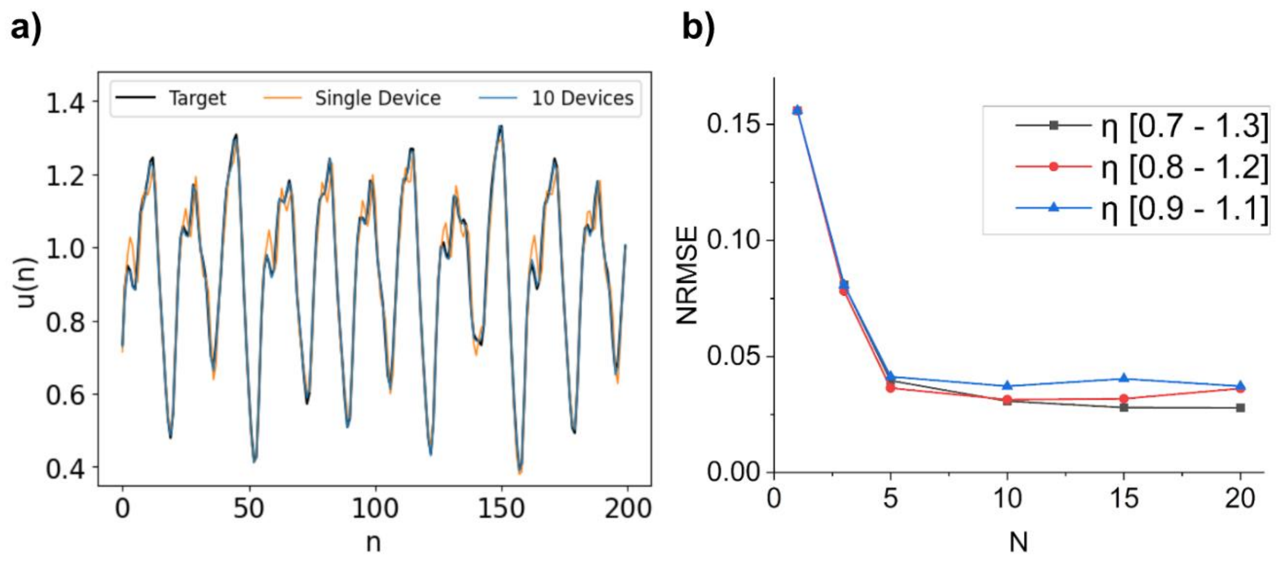

Figure 4| Performance of Mackey-Glass Prediction with Impact of Device Variation. a) Comparison of reservoir computing output of single device network (orange), 10 device network (blue), and target sequence (black) during the Mackey-Glass prediction task. Parameter η was linearly spaced between [0.7 - 1.3] in the network with 10 devices. Mask was shuffled for both the single device and 10 device networks until optimal masks were found to eliminate the change in performance due to random masks. The single device network achieved a NRMSE of 0.1586. The 10 device network achieved a NRMSE of 0.0387. **b)** Performance improves as more memristors are added in parallel. Each point represents the best performing network out of 30 simulations where the mask is randomized each simulation. Memristors in parallel each had different parameters η which were linearly spaced between [0.7 - 1.3] (black), [0.8 - 1.2] (red), or [0.9 - 1.1] (blue). The same training sequence was used in every simulation.

suggesting that more variation between the devices will further improve performance.

Results of the Mackey-Glass prediction task are in figure 4. Figure 4a) shows a side-by-side comparison of the performance of the 10-device network and the single device network. Just like in the Hénon map prediction task, the 10-device network was able to perform much better, achieving a NRMSE of 0.0387, compared to the single device network which achieved NRMSE of 0.1586. Figure 4b) shows how NRMSE decreases with the number of devices in parallel. Just like in the Hénon-Map prediction, at around 10 devices, performance will saturate and no longer lower NRMSE. This is because as more devices are added in parallel, the devices are becoming more similar to their neighbors because the interval of parameter η has not changed. However, in the simulations where the interval of parameter η was greater, NRMSE continued to decrease even further with an increased number of devices before saturating. This suggests that if we want to continue to increase the performance of the network, even greater device variation is required to avoid having device responses which are too similar.

CONCLUSION

The spring SULI semester was an extension of my work during the fall SULI project. Because I continued the same project, I was able to explore reservoir computing in a far more sophisticated way than my initial exposure of it in the fall. There is still much more I would like to do before finishing this project. The idea of applying the same voltage sequence to different devices in parallel is a novel idea which I would like to publish a paper about. To make my results stronger, I am integrating experimental data into computational simulations. Currently, I am working to model current-voltage behavior of devices fabricated by scientists in my group. By incorporating these device characteristics, the reservoir computing results I have found will be grounded in the physics of the memristor device. I plan to continue to work on this project during the summer in order to refine and publish my results.

My SULI internships at Los Alamos National Laboratory have been incredibly formative. I have developed a handful of skills which will be useful for graduate school and beyond. I develop the ability to find answers by diving deep into established literature. I learned how to ask questions and develop protocols to test ideas I had. Most importantly, I gained confidence that I could lead my own research project. There were countless instances where I did not understand what I was reading or did not know how to execute an idea I had in my head. But, after enough time of close reading and brainstorming, I was always able to arrive at a moment where it made sense. With this in mind, I am sure I will be able to tackle all sorts of novel problem I encounter in graduate school and elsewhere.

METHODS

1. Time Multiplexing and Training

The time multiplexing step is as follows. The input $u(n)$ is multiplied by a random and fixed vector $\boldsymbol{M}$, called the mask, with dimensions $(M \times 1)$. In this work, we consider a binary mask where the elements in $\boldsymbol{M}$ take on the values of 1 or -1 by random. The product $\boldsymbol{M} \cdot u(n) = \boldsymbol{J}$ is then scaled to an appropriate maximum and minimum voltage, V_{min} and V_{max}. Optimal values for V_{min} and V_{max} were found by trial and error. A sequence of voltage pulses of total duration τ is generated by holding each component in $\boldsymbol{J}$ constant for a duration δ, where $\tau = M \cdot \delta$. This voltage train is applied to each memristor in parallel to generate a current response of length τ. M virtual nodes are sampled from each memristor response at every time interval δ, the total virtual nodes of the network is $M \times N$, where N is the number of memristors in parallel. The collection of virtual nodes makes up the elements of $\boldsymbol{s}(n)$, the reservoir state vector. For every $u(n)$ in the training dataset, $\boldsymbol{s}(n)$ will make up the columns of $\boldsymbol{S}$. Training is done by finding the weight matrix $\boldsymbol{W}$ which minimizes the distance between the output of the RC network $\boldsymbol{WS}$ and the targets $\boldsymbol{Y}$, that is minimizes $||\boldsymbol{WS} - \boldsymbol{Y}||$. This is done by choosing

$$\boldsymbol{W} = \boldsymbol{Y}\boldsymbol{S}^{\boldsymbol{T}}(\boldsymbol{S}\boldsymbol{S}^{\boldsymbol{T}})^{\dagger} \qquad (6)$$

Once the network is trained, a separate dataset is used to evaluate the performance of the RC network.

2. Generating Time-Series Datasets

The time series datasets are generated using equations (1) & (2) or (3). For each time series task, 2001 iterations are generated far away from the initial conditions of the sequences. The iterations 2 to 1001 are used for training and the iterations 1002 to 2001 are used for testing. To avoid the initial transient phase of the reservoir, the first 5 points of the reservoir output and the target sequence are thrown out in both training and testing datasets for the Hénon map prediction. Likewise, for the Mackey-Glass oscillator prediction, the first 20 are thrown out of the reservoir output and target for training and testing datasets.

3. Memristor Model

The parameters for the numerical model described by equations (5) and (6) have the following values: $\gamma = 2.14 \times 10^{-6}$, $d = 1.4$, $\lambda = 1300$, $\kappa = 400 \times 10^{-6}$, and $w_0 = 0.5$. This model describes the dynamics of the state variable $w(t)$, a dimensionless variable related to conductance which affects how much current will pass through the memristor for a given voltage $V(t)$; when $w(t)$ is large, the magnitude of the current response will increase. The window function:

$$R(w) = \begin{cases} 1 - e^{3(w-1)} & V > 0 \\ 1 - e^{-3w} & V < 0 \end{cases} \qquad (7)$$

fixes w between 0 and 1. w will increase when there is a positive voltage and decrease when there is a negative voltage. In equation (5), short term memory is achieved with the second term in the first equation. Parameter κ is the decay time constant which describes how fast the model returns to its equilibrium state. For our model, the parameter κ is set to $400\ \mu s$. Equation (5) was solved using first order Runge-Kutta algorithm with a time step of $1\mu s$.

REFERENCES

1. Matteo Cucchi et al 2022 *Neuromorph. Comput*. Eng. 2 032002
2. Strubell, E., Ganesh, A., & McCallum, A. (2020). Energy and Policy Considerations for Modern Deep Learning Research. *Proceedings of the AAAI Conference on Artificial Intelligence*, *34*(09), 13693-13696.
3. Appeltant, L. et al. Information processing using a single dynamical node as complex system. *Nat. Commun.* 2:468 doi: 10.1038 / ncomms1476 (2011)
4. Zhong, Y., Tang, J., Li, X. et al. Dynamic memristor-based reservoir computing for high-efficiency temporal signal processing. *Nat Commun* 12, 408 (2021).
5. Du, C., Cai, F., Zidan, M.A. *et al.* Reservoir computing using dynamic memristors for temporal information processing. *Nat Commun* 8, 2204 (2017).

Acknowledgements

I'd like to thank the Department of Energy for funding my research internship as well as Dr. Aiping Chen, Dr. Sundar Kunwar, and Dr. Francesco Caravelli for their mentorship.