2014

A Comparative Study of Reservoir Computing for Temporal Signal Processing

Alireza Goudarzi
University of New Mexico

Peter Banda
Portland State University

Matthew R. Lakin
University of New Mexico

Christof Teuscher
Portland State University, teuscher@pdx.edu

Darko Stefanovic
University of New Mexico

A Comparative Study of Reservoir Computing for Temporal Signal Processing

Alireza Goudarzi[1], Peter Banda[2], Matthew R. Lakin[1], Christof Teuscher[3], and Darko Stefanovic[1]

***Abstract*—Reservoir computing (RC) is a novel approach to time series prediction using recurrent neural networks. In RC, an input signal perturbs the intrinsic dynamics of a medium called a reservoir. A readout layer is then trained to reconstruct a target output from the reservoir's state. The multitude of RC architectures and evaluation metrics poses a challenge to both practitioners and theorists who study the task-solving performance and computational power of RC. In addition, in contrast to traditional computation models, the reservoir is a dynamical system in which computation and memory are inseparable, and therefore hard to analyze. Here, we compare echo state networks (ESN), a popular RC architecture, with tapped-delay lines (DL) and nonlinear autoregressive exogenous (NARX) networks, which we use to model systems with limited computation and limited memory respectively. We compare the performance of the three systems while computing three common benchmark time series: Hénon Map, NARMA10, and NARMA20. We find that the role of the reservoir in the reservoir computing paradigm goes beyond providing a memory of the past inputs. The DL and the NARX network have higher memorization capability, but fall short of the generalization power of the ESN.**

***Index Terms*—Reservoir computing, echo state networks, nonlinear autoregressive networks, time-delayed networks, time series computing**

I. Introduction

Reservoir computing is a recent development in recurrent neural network research with applications to temporal pattern recognition [1]. RC's performance in time series processing tasks and its flexible implementation has made it an intriguing concept in machine learning and unconventional computing communities [2]–[12]. In this paper, we functionally compare the performance of reservoir computing with linear and nonlinear autoregressive methods for temporal signal processing to develop a baseline for understanding memory and information processing in reservoir computing.

In reservoir computing, a high-dimensional dynamical core called a *reservoir* is perturbed with an external input. The reservoir states are then linearly combined to create the output. The readout parameters can be calculated by performing regression on the state of a teacher-driven reservoir and the expected teacher output. Figure 1 shows a sample RC architecture. Unlike other forms of neural computation, computation in RC takes place within the transient dynamics of the reservoir. The computational power of the reservoir is attributed to a short-term memory created by the reservoir [13] and the ability to preserve the temporal information from distinct signals over time [14], [15]. Several studies attributed this property to the dynamical regime of the reservoir and showed it to be optimal when the system operates in the critical dynamical regime—a regime in which perturbations to the system's trajectory in its phase space neither spread nor die out [15]–[19]. The reason for this observation remains unknown. Maass et al. [14] proved that given the two properties of *separation* and *approximation*, a reservoir system is capable of approximating any time series. The separation property ensures that the reservoir perturbations from distinct signals remain distinguishable whereas the approximation property ensures that the output layer can approximate any function of the reservoir states to an arbitrary degree of accuracy. Jaeger [20] proposed that an ideal reservoir needs to have the so-called echo state property (ESP), which means that the reservoir states asymptotically depend on the input and not the initial state of the reservoir. It has also been suggested that the reservoir dynamics acts like a spatiotemporal kernel, projecting the input signal onto a high-dimensional feature space [5], [21]. However, unlike in kernel methods, the reservoir explicitly computes the feature space.

RC's robustness to the underlying implementation as well as its efficient training algorithm makes it a suitable choice for time series analysis [22]. However, despite more than a decade of research in RC and many success stories, its wide-spread adoption is still forthcoming for three main reasons: first, the lack of theoretical understanding of RC's working and its computational power, and second, the absence of a unifying implementation framework and performance analysis results, and thirdly, missing comparison with conventional methods.

II. Objectives

Our main objective is to compare time series computing in the RC paradigm, in which memory and computation are integrated, with two basic time series computing methods: first, a device with perfect memory and no computational power, ordinary linear regression on tapped-delay line (DL); and second, a device with limited memory and arbitrary computational power, a nonlinear autoregressive exogenous (NARX) neural network. This is a first step toward a systematic investigation of topology, memory, computation, and dynamics in RC. In this article we restrict ourselves to ESNs with a fully connected reservoir and input

[1] Department of Computer Science, University of New Mexico, Albuquerque, NM 87131, USA e-mail: alirezag@cs.unm.edu.

[2] Department of Computer Science, Portland State University, Portland, OR 97207, USA.

[3] Department of Electrical and Computer Engineering, Portland State University, Portland, OR 97207, USA.

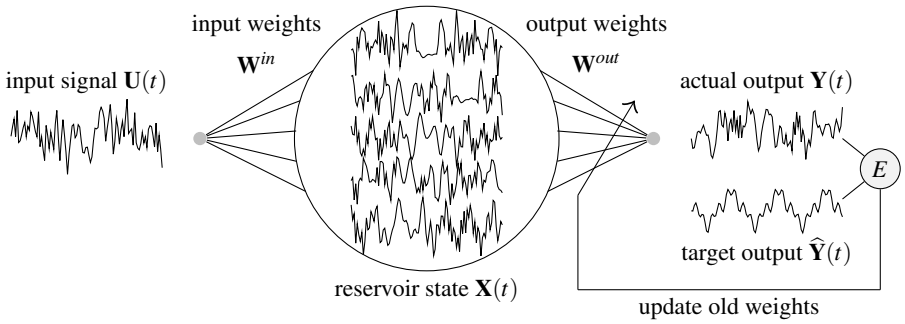


Figure 1. Computation in a reservoir computer. The reservoir is made up of a dynamical neural network with randomly assigned weights. The state of the nodes are represented by $\mathbf{X}(t)$. The input signal $\mathbf{U}(t)$ is fed into every reservoir node i with a corresponding weight w_i^{in} denoted with weight column vector $\mathbf{W}^{in} = [w_i^{in}]$. Reservoir nodes are themselves coupled with each other using the weight matrix $\mathbf{W}^{res} = [w_{ij}^{res}]$, where w_{ij}^{res} is the weight of the connection from node j to node i.

layer. We study the performance of ESN and autoregressive model on solving three time series problems: computing the 10th order NARMA time series [23], the 20th order NARMA time series [24], and the Hénon Map [25]. We also provide performance results using several variations of the mean squared error (MSE) and symmetric mean absolute percentage (SAMP) error to make our results accessible to the broader neural network and time series analysis research communities. Our systematic comparison between the ESN and autoregressive model provides solid evidence that the reservoir in the ESN performs non-trivial computation and is not just a memory device.

III. A Brief Survey of Previous Work

The first conception of the RC paradigm in the recurrent neural network (RNN) community was Jaeger's echo state network (ESN) [26]. In this early ESN, the reservoir consisted of N fully interconnected sigmoidal nodes. The reservoir connectivity was represented by a weight matrix with elements sampled from a uniform distribution in the interval $[-1,1]$. The weight matrix was then rescaled to have a spectral radius of $\lambda < 1$, a sufficient condition for ESP.

The input signal was connected to all the reservoir nodes and their weights were randomly assigned from the set $\{-1,1\}$. Later, Jaeger [20], [27] proposed that the sparsity of the connection weight matrix would improve performance and therefore only 20% of the connections were assigned weights from the set $\{-47,47\}$.Verstraeten et al. [28] used a 50% sparse reservoir and a normal distribution for the connection weights, and scaled the weight matrix posteriori to ensure the ESP; also, only 10% of the nodes were connected to the input. This study indicated that, contrary to the earlier report by Jaeger [26], the performance of the reservoir was sensitive to the spectral radius and showed optimality for $\lambda \approx 1.1$. Venayagamoorthy and Shishir [29] demonstrated experimentally that the spectral radius also affects training time, but, they did not study spectral radii larger than one. Gallicchio and Micheli [30] provided evidence that the sparsity of the reservoir has a negligible effect on ESN performance, but depending on the task, input weight heterogeneity can significantly improve performance. Büsing et al. [18] reported, from private communication with Jaeger, that different reservoir structures, such as the scale-free and the small-world topologies, do not have any significant effect on ESN performance. Song and Feng [31] demonstrated that in ESNs, with complex network reservoirs, high average path length and low clustering coefficient improved the performance. This finding is at odds with what has been observed in complex cortical circuits [32] and other studies of ESN [33].

Rodan and Tino [24] studied an ESN model with a very simple reservoir consisting of nodes that are interconnected in a cycle with homogeneous input weights and homogeneous reservoir weights, and showed that its performance can be made arbitrarily close to that of the classical ESN. This finding addressed for the first time concerns about the practical use of ESNs in embedded systems due to their complexity [2]. Massar and Massar [34] formulated a mean-field approximation to the ESN reservoir and demonstrated that the optimal standard deviation of a normally distributed weight matrix σ_w is an inverse power-law of the reservoir size N with exponent -0.5. However, this optimality is based on having critical dynamics and not task-solving performance.

IV. Models

To understand reservoir computation, we compare its behavior with a system with perfect memory and no computational power and a system with limited memory and arbitrary computational power.

We choose delay line systems, NARX neural networks, and echo state networks as described below. We use $U(t)$, $Y(t)$, and $\widehat{Y}(t)$ to denote the time-dependent input signals, the time dependent output signal, and the time-dependent target signal, respectively.

A. Delay line

A tapped-delay line (DL) is a simple system that allows us to access a delayed version of a signal. To compare the computation in a reservoir with the DL, we use a linear readout layer and connect it to read the states of

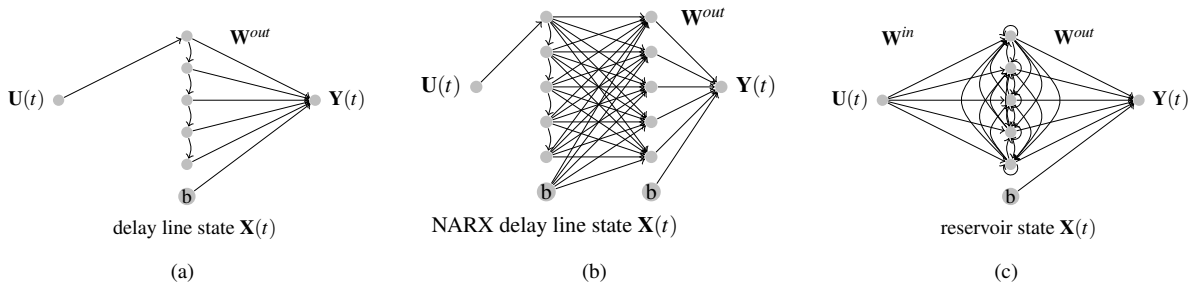


Figure 2. Architecture of delay line with a linear readout, the NARX neural network, and the ESN.

the DL. Figure 2(a) is a schematic for this architecture. Note that this architecture is different from the delay line used in [24] in that the input is only connected to a single unit in the delay line. The delay units do not perform any computation. The system is then fed with a teacher input and the weights are trained using an ordinary linear regression on the teacher output as follows:

$$\mathbf{W}^{out} = (\mathbf{X}^T \cdot \mathbf{X})^{-1} \cdot \mathbf{X}^T \cdot \widehat{\mathbf{Y}}, \tag{1}$$

where each row of $\mathbf{X}$ represents the state of the DL at a specific time $X(t_0)$ and the rows of $\widehat{\mathbf{Y}}$ are the teacher output at for the corresponding time $Y(t_0)$. Note that the DL states are augmented with a bias unit with a constant value $b = 1$. Initially, all the DL states are set to zero.

B. NARX Network

The NARX network is an autoregressive neural network architecture with a tapped delay input and one or more hidden layers. Both hidden layers and the output layer are provided with a bias input with constant value $b = 1$. We use tanh as the transfer function for the hidden layer and a linear transfer function for the output layer. The network is trained using the Marquardt algorithm [35]. This architecture performs a nonlinear regression on the teacher output using the previous values of the input accessible through the tapped delay line. A schematic of this architecture is given in Figure 2(b). Since we would like to study the effect of regression complexity on the performance, we fix the length of the tapped delay to 10 and vary the number of hidden nodes.

C. Echo State Network

In our ESN, the reservoir consists of a fully connected network of N nodes extended with a constant bias node $b = 1$. The input and the output nodes are connected to all the reservoir nodes. The input weight matrix is an $I \times N$ matrix $\mathbf{W}^{in} = [w^{in}_{i,j}]$, where I is the number of input nodes and $w^{in}_{j,i}$ is the weight of the connection from input node i to reservoir node j. The connection weights inside the reservoir are represented by an $N \times N$ matrix $\mathbf{W}^{res} = [w^{res}_{j,k}]$, where $w^{res}_{j,k}$ is the weight from node k to node j in the reservoir. The output weight matrix is an $(N+1(\times O$ matrix $\mathbf{W}^{out} = [w^{out}_{l,k}]$, where O is the number of output nodes and $w^{out}_{l,k}$ is the weight of the connection from reservoir node k to output node l. All the weights are samples of i.i.d. random variables from a normal distribution with mean $\mu_w = 0$ and standard deviation σ_w. We represent the time-varying input signal by an Ith order column vector $\mathbf{U}(t) = [u_i(t)]$, the reservoir state by an Nth order column vector $\mathbf{X}(t) = [x_j(t)]$, and the generated output by an Oth order column vector $\mathbf{Y}(t) = [y_l(t)]$. We compute the time evolution of each reservoir node in discrete time as:

$$x_j(t+1) = \tanh(\mathbf{W}^{res}_j \cdot \mathbf{X}(t) + \mathbf{W}^{in} \cdot \mathbf{U}(t)), \tag{2}$$

where tanh is the nonlinear transfer function of the reservoir nodes and $\mathbf{W}^{res}_j$ is the jth row of the reservoir weight matrix. The reservoir output is then given by:

$$\mathbf{Y}(t) = \mathbf{W}^{out} \cdot \mathbf{X}(t). \tag{3}$$

We train the output weights to minimize the squared output error $E = ||\mathbf{Y}(t) - \widehat{\mathbf{Y}}(t)||^2$ given the target output $\widehat{\mathbf{Y}}(t)$. As with DL, the output weights are calculated using the ordinary linear regression given in Equation 1.

V. Evaluation Methodology

The study of task-solving performance and analysis of computational power in RC is a major challenge because there are a variety of RC architectures, each with a unique set of parameters that potentially affect the performance. The optimal RC parameters are task-dependent and must be adjusted experimentally. Furthermore, not all studies use the same tasks and the same performance metrics to evaluate their results. In addition, in contrast to classical computation models in which a programmed automaton acts on a storage device, RC is a dynamical system in which memory and computing are inseparable parts of a single phenomenon. In other words, in RC the same dynamical process that performs computation also retains the memory of the previous results and inputs. Thus, it is not clear how much of the RC's performance can be attributed to its memory capacity and how much to its computational power. As a way of approaching this issue, we attempt to create a functional comparison between the ESN, DL, and NARX networks.

We choose three temporal tasks for our evaluation: the Hénon Map, the NARMA 10 time series, and the NARMA 20 time series. These tasks vary in increasing order in their

time lag dependencies and the number of terms involved and thus let us compare the performance of our systems based on the memory and computational requirements for task solving. We measure the performance using three variations of MSE and a SAMP error measure to allow easy comparison with related work.

A. Tasks

1) Hénon Map Time Series: This time series is generated by the following system:

$$y_t = 1 - 1.4y_{t-1}^2 + 0.3y_{t-2} + z_t, \tag{4}$$

where z_t is a white noise term with standard deviation 0.001. This is an example of a task that requires limited computation and memory, and can therefore be used as a baseline to evaluate ESN performance.

2) NARMA 10 Time Series: Nonlinear autoregressive moving average (NARMA) is a discrete-time temporal task with 10th-order time lag. To simplify the notation we use y_t to denote $y(t)$. The NARMA 10 time series is given by:

$$y_t = \alpha y_{t-1} + \beta y_{t-1} \sum_{i=1}^{n} y_{t-i} + \gamma u_{t-n} u_{t-1} + \delta, \tag{5}$$

where $n = 10$, $\alpha = 0.3, \beta = 0.05, \gamma = 1.5, \delta = 0.1$. The input u_t is drawn from a uniform distribution in the interval $[0, 0.5]$. This task presents a challenging problem to any computational system because of its nonlinearity and dependence on long time lags. Calculating the task is trivial if one has access to a device capable of algorithmic programming and perfect memory of both the input and the outputs of up to 10 previous time steps. This task is often used to evaluate the memory capacity and computational power of ESN and other recurrent neural networks.

3) NARMA 20 Time Series: NARMA 20 requires twice the memory and computation compared to NARMA 10 with an additional nonlinearity because of the saturation function *tanh*. This task is very unstable and the saturation function keeps its values bounded. NARMA 20 time series is given by:

$$y_t = \tanh(\alpha y_{t-1} + \beta y_{t-1} \sum_{i=1}^{n} y_{t-i} + \gamma u_{t-n} u_{t-1} + \delta), \tag{6}$$

where $n = 20$, and the rest of the constants are set as in NARMA 10.

B. Error Calculation

A challenge in comparing results across different studies is the way each study evaluates its results. In the case of time series analysis, each study may use a different error calculation to measure the performance of the presented methods. We present three different error calculations commonly used in the time series analysis literature. We use y to refer to the time-dependent output and $\widehat{y}$ to refer to the target output. The expectation operator $\langle \cdot \rangle$ refers to the time average of its operand.

1) Root normalized mean squared error: The most commonly used measure is a root normalized mean squared error (RNMSE) calculated as follows:

$$RNMSE = \sqrt{\frac{\langle (y - \widehat{y})^2 \rangle}{\sigma_{\widehat{y}} w^2}}. \tag{7}$$

Here $\sigma_{\widehat{y}}^2$ is the standard deviation of the target output over time. In some studies this calculation is used without taking the square root, in which case it is simply called a normalized mean squared error (NMSE).

2) Normalized root mean squared error: A variant of the normalized error is the normalized root mean square error (NRMSE), also known as normalized root mean squared deviation (NRMSD). It is calculated as follows:

$$NRMSE = \frac{\sqrt{\langle (y - \widehat{y})^2 \rangle}}{\max(\widehat{y}) - \min(\widehat{y})}. \tag{8}$$

In this variant, the error is normalized by the width of the range covered by the target signal. Both RNMSE and NRMSE attempt to normalize the error between 0 and 1. However, if the distance between the output and the target output is larger than the standard deviation of the target output or its range, they may produce an error value larger than 1.

3) Symmetric absolute mean percentage error: The symmetric absolute mean percentage (SAMP) error, on the other hand, is guaranteed to produce an error value between 0% and 100%. SAMP is given by:

$$SAMP = 100 \times \left\langle \frac{|y - \widehat{y}|}{y + \widehat{y}} \right\rangle. \tag{9}$$

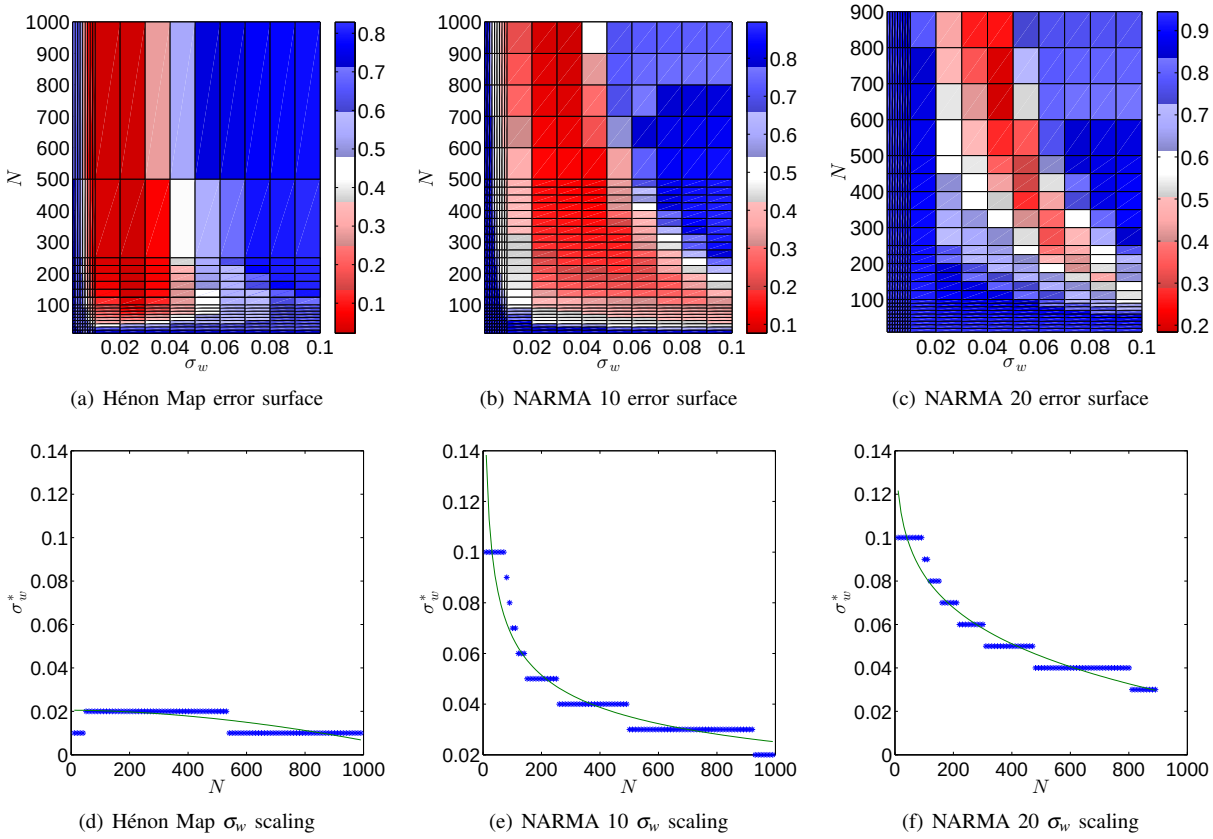
Throughout the rest of the paper, we use the RNMSE error to produce the plots and make our comparison between the three systems. We have tabulated the results using the other metrics in the Appendix B.

C. Reservoir optimization

Depending on the ESN architecture, its performance can be sensitive to some of the model's parameters. These parameters are optimized using offline cross-validation [24], [36] or online adaptation [19], [37]. This is a preliminary stage before the functional comparison. We are interested in the scaling of these parameters, which we study systematically. Figures 3(a)-3(c) shows the resulting error surface by averaging the result of the 10 runs of each σ_w-N combination. We observe that, as the nonlinearity of the task and its required memory increase, ESN performance becomes more sensitive to changes in σ_w and favors a more heterogeneous weight assignment (larger σ_w). We find the optimal standard deviation σ_w^* as a function of N for each task:

$$\sigma_w^*(N) = \arg\min_{\sigma_w} RNMSE(\sigma_w, N). \tag{10}$$

We found experimentally that $\sigma_w^*(N)$ is best fitted by a power-law curve. The bottom row of Figure 3 shows the result of this optimization and the power-law fit. The details of these fits are provided in Appendix A. For NARMA 10



(a) Hénon Map error surface (b) NARMA 10 error surface (c) NARMA 20 error surface

(d) Hénon Map σ_w scaling (e) NARMA 10 σ_w scaling (f) NARMA 20 σ_w scaling

Figure 3. Scaling and optimization in the ESN. Figures 3(a), 3(b), and 3(c) show the training error surface of the ESN on the Hénon Map, the NARMA 10, and the NARMA 20 tasks respectively. We create a scaling-law by finding the optimal standard deviation $\sigma_w^*(N)$ according to Equation 10 and fitting the power-law $ax^b + c$ to it. Figures 3(d), 3(e), and 3(f) show the data points (blue markers) and the fit (solid line) for the $\sigma_w^*(N)$.

and NARMA 20 the power law is well behaved, except for the Hénon Map which is not sensitive to σ_w. We use $\sigma_w^* = 0.02$ for Hénon Map experiments. It is noteworthy that this power-law behavior is qualitatively consistent with what we expect from the theoretical result in [34], although the exact power-law coefficient is task-dependent.

D. Functional Comparison

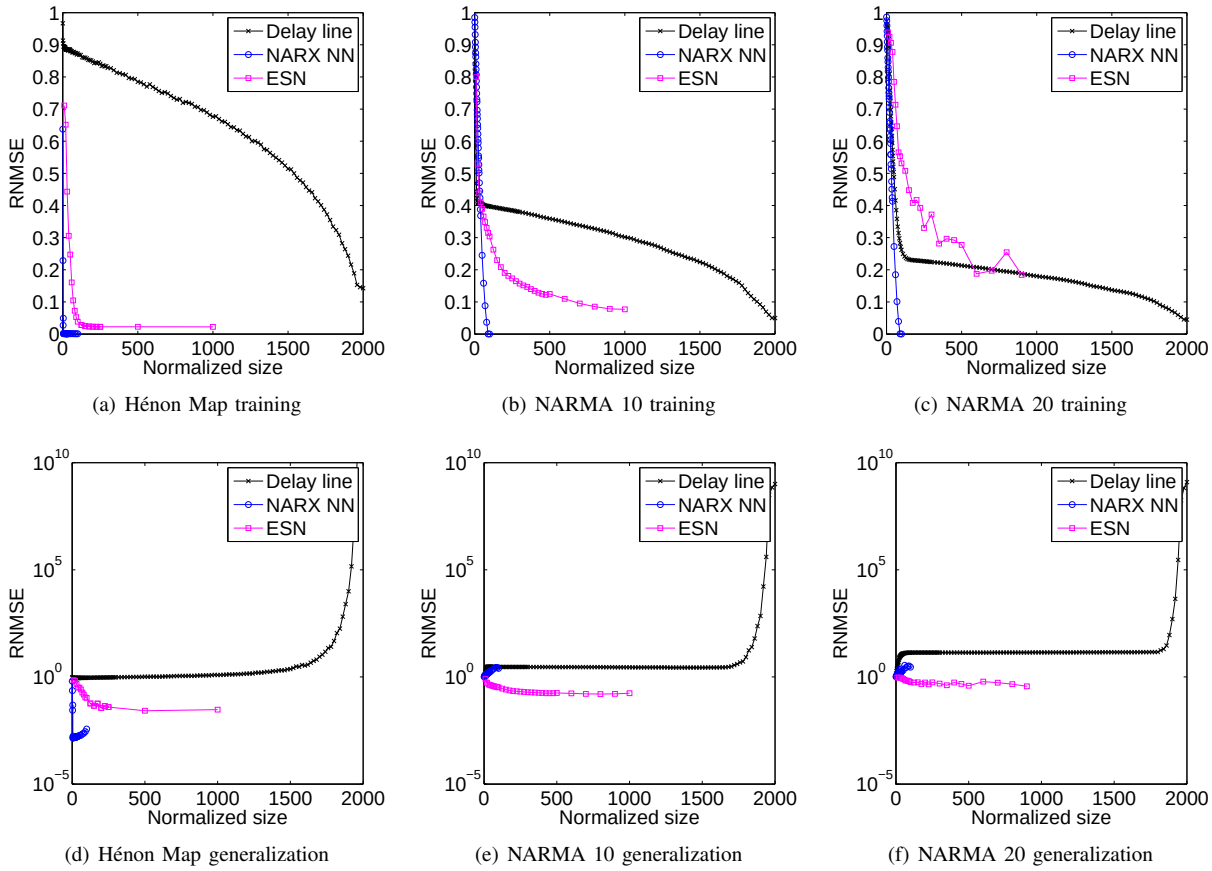
The division between memory and computation is not fully understood. Here, we attempt to compare the ESN size to the size of an equivalent device with only memory capacity and no computational power, and to a device with limited memory and theoretically arbitrary computational power. A DL of length n stores the perfect memory of the past n inputs and a NARX network with N hidden units can be a universal approximator of any time series. We would like to compare the performance of a linear readout with access to the reservoir state with a linear readout with access to the delay line states and also to the performance of the NARX network. It is clear that the ESN, the DL, and the NARX network are very different, i.e., the memory and computational power of ESN, DL, and NARX networks differ significantly even with identical N. Therefore, we perform a functional comparison in which we study the ESN, the DL, and the NARX network with equal RNMSE on the same task. For ESN, we have two parameters: the standard deviation of the normal distribution used to create the weight matrix σ_w and the number of nodes N in the reservoir. We chose the σ_w that optimizes the performance of ESN for each N as described in Section V-C.

E. Experimental Setup

In this section we describe the parameters and data sets used for our simulations. The training and testing for delay line and NARX networks are done by generating 10 time series of 4,000 time steps. We used 20 time seres of the same length for ESNs. We used the first 2,000 time steps of each time series to train the model and the second 2,000 steps to test the model. The training and testing performance metrics are then averaged over the 10 runs. The model specific setting are as follows:

1) Delay line: A delay line is a fixed system with only one parameter N that defines the number of taps. For this study we limit ourselves to $1 \leq N \leq 2,000$. We have also experimented with $2,000 \leq N \leq 3,000$, but the performance of the delay line does not change for $N > 2,000$. We take $N = 2,000$ as the largest delay line for this study.

2) NARX neural network: Since the training in the NARX network is sensitive to the random initial weights, we instantiate a new network for each time series. We fix



(a) Hénon Map training (b) NARMA 10 training (c) NARMA 20 training

(d) Hénon Map generalization (e) NARMA 10 generalization (f) NARMA 20 generalization

Figure 4. Training and generalization RNMSE of the DL, the NARX network, and the ESN for three different tasks. The DL can memorize the patterns, but not generalize for any of the tasks. The NARX network can generalize for the Hénon Map, but overfits for the NARMA 10 and the NARMA 20. The ESN can both memorize and generalize the temporal patterns for all the tasks.

the number of input taps to 10 and the number of hidden layers to 1. We use the number of nodes in the hidden layer N as the control parameter, with $1 \leq N \leq 100$.

3) Echo state network: A single instantiation of ESN contains randomly assigned weights and the reservoir initial states. To average the ESN properties over these variations we instantiate five ESN to train and test for every time series. The error is then averaged over the five instances and then over 20 time series. The variable parameters for ESN are the number of reservoir nodes N and the standard deviation of the normal distribution used to generate the reservoir and the input weights. However, for each N we only use the σ_w^* as described in Section V-C. We study the ESN with the reservoir size $10 \leq N \leq 1,000$.

VI. Results

Figures 4(a)-4(c) show the training performance measured in RNMSE as a function of the normalized system size. In all three tasks, the delay line shows a sharp decrease in error as soon as the system acquires enough capacity to hold all the required information for the task. This is $N = 2$ for Hénon Map, $N = 10$ for NARMA 10 task, and $N = 20$ for NARMA 20 task. After this point, the error decreases slowly until $N = 2,000$ where $RNMSE \approx 0.14$ after a sharp drop. The decrease in error is due to the "dimensionality curse": the fixed-length teacher time series are not representatives of the expanded state space of the delay line. This is expected to result in overfitting, which is reflected in the high testing errors on Figures 4(d)-4(f). Another expected behavior of overfitting to training data is that if the distribution of the data is wide, the error will be larger and for narrower distributions of data the error will be lower. This is why the delay line has the highest error for the simplest task, the Hénon Map, and the lowest error for the most difficult task, the NARMA 20 task. Note that to make the testing errors for all the system readable, we use logarithmic y-axes.

The NARX network behaves differently on the Hénon Map and the two NARMA tasks. For the Hénon Map, it shows the best training and testing performance around $N = 5$ and begins to overfits for $N > 5$. For both NARMA 10 and NARMA 20, the training error decreases gradually as the number of hidden nodes increases, but the system only memorizes the patterns and cannot generalize, which is characterized by increasing test RNMSE. For the NARMA 10 task, the observed training RNMSE for NARX network is comparable to the error of 0.17 reported by Atiya and Parlos [23] for the same network size $N = 40$. However, they used only 200 data points, which explains the slightly lower error. Atiya and Parlos focused on the convergence

times of different algorithms and did not publish their testing error.

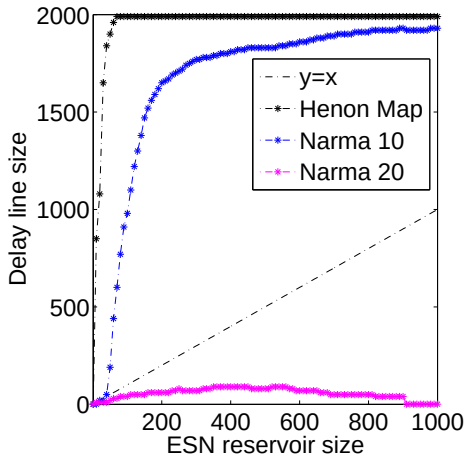
For the ESN, the training error on all three tasks decreases rapidly as the size of the reservoir increases and reaches a plateau for $N \approx 1,000$. However, unlike the DL and the NARX network, the testing error also decreases sharply as the reservoir size increases. As expected, this decrease is sharper for easier tasks. The main different of the ESN performance is that the training error increases across all N as the difficulty of the task increases, which is a sign that the readout layer is not merely memorizing the training patterns. We have tabulated the error values for all three systems on all the tasks for a few system sizes, which can be found in the Appendix B. Our ESN testing results are similar to those reported by Rodan and Tino [24] for the same system size.

Next, we compared the performance of the DL, the NARX network, and the ESN as described in Section V-D. Because the testing error of the systems do not overlap, we have to use the training error to perform a direct functional comparison between the three systems. This allows us to compare their memorization capabilities. Figure 5(a) shows the DL size as a function of the ESN size of identical training error for all three tasks. For the Hénon Map and the NARMA 10 tasks, the ESN achieves the same RNMSE as the delay line with significantly fewer nodes. For instance for the Hénon Map and NARMA 10, to achieve a training RNMSE of an ESN with 400 nodes, a delay line would need $1,990$ and $1,810$ nodes respectively. For NARMA 20, the delay line only needs 90 nodes to achieve the same RNMSE as an ESN with 400 nodes. The narrow distribution of the NARMA 20 time series allow the linear delay line to exploit the average case strategy to achieve a lower RNMSE. On the other hand, the ESN readout layer learns the task itself as in contrast to just memorizing patterns. The delay line use the same strategy for the easier tasks as well (NARMA 10 and Hénon Map), but ESN can fit to the the training data much better than the delay line and therefore requires much less resources to achieve the same error level.

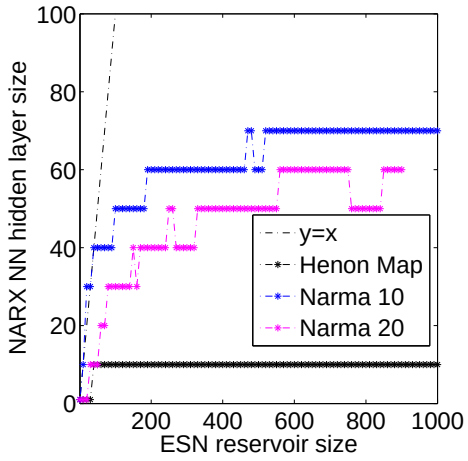
Figure 5(b) shows the functional comparison result between NARX networks and ESNs. A NARX network would need 10 hidden nodes to achieve the same RNMSE as a 400-node ESN on the Hénon Map task. The short time dependency and the simple form of this task make it very easy for the network to learn the system, which results in low training and testing RNMSE. For the NARMA 10 and the NARMA 20 tasks, the network requires 60 and 50 hidden nodes to be equivalent of the 400-node ESN. The strategy here is similar to the delay line where during learning the network tries to fit the training data on average, as best as it can. As expected this strategy will have two consequences: (1) the NARMA 20 task will be easier because of its distribution; (2) the network overfits the training data and cannot generalize to testing data.

VII. Discussion and Open Problems

The reservoir in an ESN is a dynamical system in which memory and computation are inseparable. To understand this type of information processing we compared the ESN performance with a memory-only device, the DL, and a limited-memory but computationally powerful device, the NARX network. Our results illustrate that the performance of ESN is not only due to its memory capacity; ESN readout does not create an autoregression of the input, such as in the DL or the NARX network. The information processing that takes place inside a reservoir is fundamentally different from other types of neural networks. The exact mechanism of this process remains to be investigated. Studying reservoir computing usually takes place by analyzing the systems performance for task solving with different com-



(a) DL vs. ESN



(b) NARX NN vs. ESN

Figure 5. Figure 5(a) shows how the complexity of the DL compares with the ESN of identical memorization performance. Except for the NARMA 20, which requires perfect memory of the 20 previous time steps, the ESN memorization capability far surpasses the DL. Figure 5(b) shows the same comparison between the NARX and the ESN. The NARX network out performs the ESN in all tasks. Due its complexity, the NARX network can memorize the patterns very well, but is not able to generalize to the new patterns (see Figure 4).

putational and memory requirements. To understand the details of information processing in a reservoir, we have to understand the effects of the reservoir's architecture on its fundamental memory and computational capacity. We also have to be able to define the classes of tasks that can be parametrically varied in memory requirement and nonlinearity. Our study reveals that although ESN cannot memorize patterns as well as a memory device or a neural network, it greatly outperforms them in generalizing to novel inputs. Also, increasing reservoir size in ESN improves the performance of generalization, whereas in the DL or the NARX network this will result in increased overfitting leading to poorer generalization. One solution would be to extend the receiver operation characteristic (ROC) and receiver error characteristic (REC) curve methods to decide on the quality of generalization in ESN [38]–[40]. In the neural network community, methods based on pruning, regularization, cross-validation, and information criterion have been used to alleviate the overfitting problem [41]–[53]. Among these methods, regularization has been successfully used in ESNs [3]. However, these methods focus on increasing the neural network's performance and are not suitable to quantify overfitting or to study task hardness. Another area that requires more research is the amount of training that the ESN requires to guarantee a certain performance, as is described in probably approximately correct methods [54]–[56]. To the best of our knowledge these problems have not been addressed in the case of high-dimensional dynamical systems. A well developed theory of computation in reservoir computing needs to address all of these aspects. In future work, we will study some of these issues experimentally and based on our observations, we will attempt to develop theoretical understanding of computation in the reservoir computing paradigm.

VIII. Conclusion

Reservoir computing is an approach to neural network training which has been successful in areas of machine learning, time series analysis, robot control, and sequence learning. There has been many studies aimed at understanding the working of RC and the factors that affect its performance. However, because of the complexity of the reservoir in RC, none of these studies have been completely satisfactory and have often resulted in contradictory conclusions. In this paper, we compared the performance of three approaches to time series analysis: the delay line, the NARX network, and the ESN. These methods vary in their memory capacity and computational power. The delay line retains a perfect memory of the past, but does not have any computational power. The NARX network only has limited memory of the past, but in principle can perform any computation. Finally, the ESN does not have an explicit access to past memory, but its reservoir carries out computation using the implicit memory of the past represented in its dynamics. Using a functional comparison we showed that for simple tasks with short time dependencies, the delay line requires more than four times as much resources that ESN requires to achieve the same error, while the NARX network requires 40 times less resources than ESN to achieve equivalent error. For tasks with long time dependencies and narrow distributions the delay line requires less than one fourth the resources of the ESN and the NARX network requires less then one fifth the same resources. However, neither a delay line nor a NARX network can achieve the generalization power of an ESN. Many theoretical aspects of reservoir computing, such as the memory-computation trade-off, and the relation between reservoir's structure, its dynamics, and its performance remain as open problems.

Acknowledgment

The work was support by NSF grants #1028238 and #1028120. M.R.L. gratefully acknowledges support from the New Mexico Cancer Nanoscience and Microsystems Training Center.

Appendix A
Fitting σ_w

Before we can fit σ_w^*, we have to interpolate the data points on the error surface with a linear fit. This allows us to use all values of N and σ_w and create a smooth fit. Table I shows the goodness of fit statistics for the linear fit to the error surface. Low SSE and high R^2 statistics on this fit shows the the surface accurately represent the data points. We then calculate the σ_w^* corresponding to the standard deviation of the weight matrix for each N that minimizes the error. We represent σ_w^* as a function of N and fit the power-law $ax^b + c$ to it. The result of the fit and the goodness of fit statistics are given in Table II.

task	SSE	R^2
Hénon Map	1.5486×10^{-31}	1
NARMA 10	2.989×10^{-31}	1
NARMA 20	7.3725×10^{-31}	1

Table I
Goodness of fit statistics for the interpolant fit to the error surface (Figure 3) as a function of reservoir σ_w and N for all three tasks.

Appendix B
The Performance Results

Tables III, IV, and V tabulate the average testing and training errors of optimal ESNs, delay lines, and NARX networks for the three different tasks using three different measures, i.e., RNMSE, NRMSE, and SAMP.

References

[1] B. Schrauwen, D. Verstraeten, and J. V. Campenhout, "An overview of reservoir computing: theory, applications and implementations," in *Proceedings of the 15th European Symposium on Artificial Neural Networks*, 2007, pp. 471–482.

[2] D. Prokhorov, "Echo state networks: appeal and challenges," in *Neural Networks, 2005. IJCNN '05. Proceedings. 2005 IEEE International Joint Conference on*, vol. 3, 2005, pp. 1463–1466 vol. 3.

task	measure		$N=50, \sigma_w^*=0.02$	$N=100, \sigma_w^*=0.02$	$N=150, \sigma_w^*=0.02$	$N=200, \sigma_w^*=0.02$	$N=500, \sigma_w^*=0.02$
Hénon Map	RNMSE	training	0.2474 ± 0.0215	0.0385 ± 0.0039	0.0247 ± 0.0008	0.0230 ± 0.0002	0.0226 ± 0.0001
		testing	0.3805 ± 0.0738	0.1053 ± 0.0225	0.0448 ± 0.0061	0.0359 ± 0.0048	0.0264 ± 0.0026
	NRMSE	training	0.0696 ± 0.0060	0.0109 ± 0.0011	0.0069 ± 0.0002	0.0065 ± 0.0000	0.0063 ± 0.0000
		testing	0.1071 ± 0.0208	0.0296 ± 0.0063	0.0126 ± 0.0017	0.0101 ± 0.0014	0.0074 ± 0.0007
	SAMP	training	16.2797 ± 1.1912	3.0074 ± 0.3899	1.2687 ± 0.1440	0.8498 ± 0.0584	0.6821 ± 0.0373
		testing	16.6023 ± 1.2046	3.2470 ± 0.4199	1.4182 ± 0.1577	0.9797 ± 0.0672	0.9441 ± 0.0527

task	measure		$N=50, \sigma_w^*=0.10$	$N=100, \sigma_w^*=0.07$	$N=150, \sigma_w^*=0.05$	$N=200, \sigma_w^*=0.05$	$N=500, \sigma_w^*=0.04$
NARMA 10	RNMSE	training	0.3896 ± 0.0073	0.3036 ± 0.0104	0.2300 ± 0.0058	0.1904 ± 0.0051	0.1248 ± 0.0056
		testing	0.4035 ± 0.0070	0.3270 ± 0.0113	0.2560 ± 0.0064	0.2199 ± 0.0054	0.1796 ± 0.0078
	NRMSE	training	0.0635 ± 0.0012	0.0495 ± 0.0017	0.0374 ± 0.0010	0.0309 ± 0.0008	0.0202 ± 0.0009
		testing	0.0653 ± 0.0011	0.0529 ± 0.0018	0.0414 ± 0.0011	0.0355 ± 0.0009	0.0291 ± 0.0013
	SAMP	training	4.6137 ± 0.0819	3.5742 ± 0.1343	2.6224 ± 0.0742	2.1247 ± 0.0642	1.4310 ± 0.0654
		testing	4.7961 ± 0.0862	3.8297 ± 0.1467	2.8827 ± 0.0881	2.4024 ± 0.0719	1.9710 ± 0.0900

task	measure		$N=50, \sigma_w^*=0.10$	$N=100, \sigma_w^*=0.09$	$N=150, \sigma_w^*=0.08$	$N=200, \sigma_w^*=0.07$	$N=500, \sigma_w^*=0.04$
NARMA 20	RNMSE	training	0.7846 ± 0.0246	0.5313 ± 0.0510	0.4478 ± 0.0650	0.4171 ± 0.0621	0.2776 ± 0.0183
		testing	0.8003 ± 0.0257	0.5746 ± 0.0499	8.6177 ± 24.2443	5.7683 ± 13.0557	0.3873 ± 0.0222
	NRMSE	training	0.0985 ± 0.0031	0.0667 ± 0.0065	0.0562 ± 0.0082	0.0524 ± 0.0078	0.0349 ± 0.0023
		testing	0.1012 ± 0.0032	0.0728 ± 0.0062	1.0662 ± 2.9647	0.7260 ± 1.6434	0.0491 ± 0.0028
	SAMP	training	2.9507 ± 0.0776	2.1860 ± 0.1922	1.9531 ± 0.2518	1.8459 ± 0.2461	1.3097 ± 0.0807
		testing	3.0570 ± 0.0830	2.3736 ± 0.1874	6.0811 ± 7.7539	5.3690 ± 7.2349	1.8134 ± 0.0965

Table III

TRAINING AND TESTING ERRORS FOR THE OPTIMAL ESN ON THREE DIFFERENT TASKS MEASURED USING THREE DIFFERENT ERROR METRICS RNMSE, NRMSE, AND SAMP. THE OPTIMAL σ_w^* IS MEASURED USING THE RNMSE ON THE TRAINING DATA. EACH DATA POINT IS AVERAGED OVER 100 EXPERIMENTS.

task	measure		$N=100$	$N=200$	$N=500$	$N=1000$	$N=2000$
Hénon Map	RNMSE	training	0.8726 ± 0.0044	0.8478 ± 0.0038	0.7848 ± 0.0049	0.6764 ± 0.0060	0.1426 ± 0.0066
		testing	0.9164 ± 0.0051	0.9378 ± 0.0037	1.0243 ± 0.0120	1.2294 ± 0.0183	$13305472391.5757 \pm 9380091076.0147$
	NRMSE	training	0.2452 ± 0.0011	0.2382 ± 0.0009	0.2208 ± 0.0018	0.1901 ± 0.0011	0.0402 ± 0.0018
		testing	0.2581 ± 0.0017	0.2636 ± 0.0017	0.2886 ± 0.0037	0.3463 ± 0.0046	$3748747654.4792 \pm 2646155454.1636$
	SAMP	training	56.0839 ± 0.3592	54.1909 ± 0.3698	50.1493 ± 0.3639	42.7874 ± 0.4785	2.5302 ± 0.1497
		testing	57.8635 ± 0.3953	58.3511 ± 0.2825	59.4723 ± 0.5885	60.9760 ± 0.5650	95.2081 ± 0.4781

task	measure		$N=100$	$N=200$	$N=500$	$N=1000$	$N=2000$
NARMA 10	RNMSE	training	0.3972 ± 0.0188	0.3884 ± 0.0193	0.3590 ± 0.0224	0.3017 ± 0.0230	0.0497 ± 0.0072
		testing	2.9150 ± 0.1827	2.9088 ± 0.1852	2.8797 ± 0.1919	2.8063 ± 0.2152	$1007103272.9273 \pm 482045875.2627$
	NRMSE	training	0.0650 ± 0.0021	0.0635 ± 0.0020	0.0586 ± 0.0021	0.0493 ± 0.0025	0.0082 ± 0.0013
		testing	0.4642 ± 0.0360	0.4633 ± 0.0363	0.4586 ± 0.0370	0.4468 ± 0.0401	$164091125.1086 \pm 83497086.3948$
	SAMP	training	4.6786 ± 0.1660	4.5824 ± 0.1747	4.2474 ± 0.1970	3.5989 ± 0.2575	0.1873 ± 0.0222
		testing	27.5133 ± 1.0130	27.4396 ± 1.0251	27.2029 ± 1.0782	26.5114 ± 1.2347	100.0000 ± 0.0000

task	measure		$N=100$	$N=200$	$N=500$	$N=1000$	$N=2000$
NARMA 20	RNMSE	training	0.2620 ± 0.0093	0.2289 ± 0.0068	0.2132 ± 0.0066	0.1804 ± 0.0091	0.0450 ± 0.0061
		testing	13.5807 ± 0.7275	13.6650 ± 0.7321	13.7338 ± 0.7514	13.9180 ± 0.7712	$1234560367.6547 \pm 811645432.8192$
	NRMSE	training	0.0333 ± 0.0020	0.0290 ± 0.0014	0.0271 ± 0.0014	0.0229 ± 0.0017	0.0057 ± 0.0007
		testing	1.7237 ± 0.0382	1.7346 ± 0.0402	1.7433 ± 0.0429	1.7667 ± 0.0457	$160391618.4866 \pm 109709916.7516$
	SAMP	training	1.3211 ± 0.0855	1.1533 ± 0.0600	1.0761 ± 0.0615	0.8980 ± 0.0743	0.1113 ± 0.0132
		testing	43.8473 ± 0.5246	44.0038 ± 0.5579	44.1198 ± 0.5911	44.4313 ± 0.6170	100.0000 ± 0.0000

Table IV

TRAINING AND TESTING ERRORS FOR THE DELAY LINE RESERVOIR ON THREE DIFFERENT TASKS MEASURED USING THREE DIFFERENT ERROR METRICS RNMSE, NRMSE, AND SAMP. EACH DATA POINT IS AVERAGED OVER 10 EXPERIMENTS.

task	measure		$N=5$	$N=10$	$N=30$	$N=50$	$N=100$
Hénon Map	RNMSE	training	0.0014 ± 0.0000	0.0013 ± 0.0000	0.0012 ± 0.0000	0.0011 ± 0.0000	0.0009 ± 0.0000
		testing	0.0014 ± 0.0000	0.0015 ± 0.0000	0.0016 ± 0.0000	0.0019 ± 0.0001	0.0037 ± 0.0008
	NRMSE	training	0.0004 ± 0.0000	0.0004 ± 0.0000	0.0003 ± 0.0000	0.0003 ± 0.0000	0.0003 ± 0.0000
		testing	0.0004 ± 0.0000	0.0004 ± 0.0000	0.0005 ± 0.0000	0.0005 ± 0.0000	0.0010 ± 0.0002
	SAMP	training	0.1900 ± 0.0304	0.1692 ± 0.0292	0.1674 ± 0.0293	0.1455 ± 0.0140	0.1098 ± 0.0142
		testing	0.2091 ± 0.0293	0.1914 ± 0.0218	0.2110 ± 0.0294	0.2323 ± 0.0354	0.3263 ± 0.0335
NARMA 10	RNMSE	training	0.9081 ± 0.0072	0.8235 ± 0.0091	0.5010 ± 0.0094	0.2450 ± 0.0100	0.0000 ± 0.0000
		testing	1.0894 ± 0.0074	1.1627 ± 0.0121	1.5363 ± 0.0233	1.9322 ± 0.0330	2.5015 ± 0.0755
	NRMSE	training	0.2617 ± 0.0018	0.2373 ± 0.0028	0.1444 ± 0.0026	0.0706 ± 0.0028	0.0000 ± 0.0000
		testing	0.3156 ± 0.0030	0.3368 ± 0.0048	0.4451 ± 0.0080	0.5598 ± 0.0110	0.7246 ± 0.0227
	SAMP	training	26.2977 ± 0.3168	24.2035 ± 0.3464	17.0853 ± 0.2634	10.3269 ± 0.4612	0.0000 ± 0.0000
		testing	30.4646 ± 0.4856	31.9513 ± 0.5791	40.4312 ± 0.8923	47.1503 ± 0.6935	52.7679 ± 1.0017
NARMA 20	RNMSE	training	0.9276 ± 0.0046	0.8468 ± 0.0063	0.5279 ± 0.0100	0.2725 ± 0.0084	0.0000 ± 0.0000
		testing	1.2201 ± 0.2734	1.4458 ± 0.5060	1.5594 ± 0.0238	2.0689 ± 0.0580	2.8666 ± 0.0921
	NRMSE	training	0.2673 ± 0.0015	0.2441 ± 0.0022	0.1521 ± 0.0030	0.0785 ± 0.0026	0.0000 ± 0.0000
		testing	0.3538 ± 0.0806	0.4189 ± 0.1466	0.4517 ± 0.0077	0.5994 ± 0.0186	0.8307 ± 0.0310
	SAMP	training	26.6541 ± 0.2840	24.7928 ± 0.2442	17.6939 ± 0.2953	11.0069 ± 0.4056	0.0000 ± 0.0000
		testing	30.4213 ± 0.6264	31.9725 ± 0.7206	40.2941 ± 0.7937	47.8871 ± 0.8487	54.5926 ± 1.1815

Table V

TRAINING AND TESTING ERRORS FOR THE NARX NETWORK ON THREE DIFFERENT TASKS MEASURED USING THREE DIFFERENT ERROR METRICS RNMSE, NRMSE, AND SAMP. EACH DATA POINT IS AVERAGED OVER 10 EXPERIMENTS.

[3] F. Wyffels and B. Schrauwen, "A comparative study of reservoir computing strategies for monthly time series prediction," *Neurocomputing*, vol. 73, no. 10–12, pp. 1958–1964, 2010.

[4] H. Jaeger, "Adaptive nonlinear system identification with echo state networks," in *NIPS*, 2002, pp. 593–600.

[5] M. Lukoševičius and H. Jaeger, "Reservoir computing approaches to

Hénon Map	
SSE	0.0011
R^2	0.5670
a	$-5.733 \times 10^{-8} \pm 2.9430 \times 10^{-7}$
b	1.795 ± 0.7450
c	0.02053 ± 0.0016

NARMA 10	
SSE	2.9686×10^{-4}
R^2	0.9926
a	0.306 ± 0.0095
b	-0.2609 ± 0.0249
c	-0.02537 ± 0.0079

NARMA 20	
SSE	5.8581×10^{-4}
R^2	0.9909
a	-0.03441 ± 0.0143
b	0.2156 ± 0.0408
c	0.1766 ± 0.0210

Table II
GOODNESS OF FIT STATISTICS FOR THE POWER-LAW FIT $ax^b + c$ TO σ_w^* (FIGURE 3) AS A FUNCTION OF RESERVOIR SIZE N FOR ALL THREE TASKS.

recurrent neural network training," *Computer Science Review*, vol. 3, no. 3, pp. 127–149, 2009.

[6] 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, 2013.

[7] M. Fiers, B. Maes, and P. Bienstman, "Dynamics of coupled cavities for optical reservoir computing," in *Proceedings of the 2009 Annual Symposium of the IEEE Photonics Benelux Chapter*, S. Beri, P. Tassin, G. Craggs, X. Leijtens, and J. Danckaert, Eds. VUB Press, 2009, pp. 129–132.

[8] G. Indiveri, B. Linares-Barranco, R. Legenstein, G. Deligeorgis, and T. Prodromakis, "Integration of nanoscale memristor synapses in neuromorphic computing architectures," *Nanotechnology*, vol. 24, no. 38, p. 384010, 2013.

[9] O. Obst, A. Trinchi, S. G. Hardin, M. Chadwick, I. Cole, T. H. Muster, N. Hoschke, D. Ostry, D. Price, K. N. Pham, and T. Wark, "Nano-scale reservoir computing," *Nano Communication Networks*, vol. 4, no. 4, pp. 189–196, 2013.

[10] Y. Paquot, F. Duport, A. Smerieri, J. Dambre, B. Schrauwen, M. Haelterman, and S. Massar, "Optoelectronic reservoir computing," *Scientific Reports*, vol. 2, 02 2012.

[11] C. Fernando and S. Sojakka, "Pattern recognition in a bucket," in *Advances in Artificial Life*, ser. Lecture Notes in Computer Science, W. Banzhaf, J. Ziegler, T. Christaller, P. Dittrich, and J. Kim, Eds. Springer Berlin Heidelberg, 2003, vol. 2801, pp. 588–597.

[12] A. Goudarzi, M. R. Lakin, and D. Stefanovic, "DNA reservoir computing: A novel molecular computing approach," in *DNA Computing and Molecular Programming*, ser. Lecture Notes in Computer Science, D. Soloveichik and B. Yurke, Eds. Springer International Publishing, 2013, vol. 8141, pp. 76–89.

[13] H. Jaeger and H. Haas, "Harnessing nonlinearity: Predicting chaotic systems and saving energy in wireless communication," *Science*, vol. 304, no. 5667, pp. 78–80, 2004.

[14] 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. 2531–60, 2002.

[15] T. Natschläger and W. Maass, "Information dynamics and emergent computation in recurrent circuits of spiking neurons," in *Proc. of NIPS 2003, Advances in Neural Information Processing Systems*, S. Thrun, L. Saul, and B. Schoelkpf, Eds., vol. 16. Cambridge: MIT Press, 2004, pp. 1255–1262.

[16] N. Bertschinger and T. Natschläger, "Real-time computation at the edge of chaos in recurrent neural networks," *Neural Computation*, vol. 16, no. 7, pp. 1413–1436, 2004.

[17] D. Snyder, A. Goudarzi, and C. Teuscher, "Computational capabilities of random automata networks for reservoir computing," *Phys. Rev. E*, vol. 87, p. 042808, Apr 2013.

[18] L. Büsing, B. Schrauwen, and R. Legenstein, "Connectivity, dynamics, and memory in reservoir computing with binary and analog neurons." *Neural Computation*, vol. 22, no. 5, pp. 1272–1311, 2010.

[19] J. Boedecker, O. Obst, N. M. Mayer, and M. Asada, "Initialization and self-organized optimization of recurrent neural network connectivity," *HFSP Journal*, vol. 3, no. 5, pp. 340–349, 2009.

[20] H. Jaeger, "Tutorial on training recurrent neural networks, covering BPPT, RTRL, EKF and the " echo state network" approach," German National Research Center for Information Technology, St. Augustin-Germany, Tech. Rep. GMD Report 159, 2002.

[21] M. Hermans and B. Schrauwen, "Recurrent kernel machines: Computing with infinite echo state networks," *Neural Computation*, vol. 24, no. 1, pp. 104–133, 2013/11/22 2011.

[22] M. Lukoševičius, H. Jaeger, and B. Schrauwen, "Reservoir computing trends," *KI - Künstliche Intelligenz*, vol. 26, no. 4, pp. 365–371, 2012.

[23] A. Atiya and A. Parlos, "New results on recurrent network training: unifying the algorithms and accelerating convergence," *Neural Networks, IEEE Transactions on*, vol. 11, no. 3, pp. 697–709, 2000.

[24] A. Rodan and P. Tino, "Minimum complexity echo state network," *Neural Networks, IEEE Transactions on*, vol. 22, no. 1, pp. 131–144, 2011.

[25] M. Hénon, "A two-dimensional mapping with a strange attractor," *Communications in Mathematical Physics*, vol. 50, no. 1, pp. 69–77, 1976.

[26] H. Jaeger, "The "echo state" approach to analysing and training recurrent neural networks," St. Augustin: German National Research Center for Information Technology, Tech. Rep. GMD Rep. 148, 2001.

[27] ——, "Short term memory in echo state networks," GMD-Forschungszentrum Informationstechnik, Tech. Rep. GMD Report 152, 2002.

[28] M. D. D. Verstraeten, B. Schrauwen and D. Stroobandt, "An experimental unification of reservoir computing methods," *Neural Networks*, vol. 20, no. 3, pp. 391–403, 2007.

[29] G. K. Venayagamoorthy and B. Shishir, "Effects of spectral radius and settling time in the performance of echo state networks," *Neural Networks*, vol. 22, no. 7, pp. 861 – 863, 2009.

[30] C. Gallicchio and A. Micheli, "Architectural and markovian factors of echo state networks," *Neural Networks*, vol. 24, no. 5, pp. 440 – 456, 2011.

[31] Q. Song and Z. Feng, "Effects of connectivity structure of complex echo state network on its prediction performance for nonlinear time series," *Neurocomputing*, vol. 73, no. 10–12, pp. 2177 – 2185, 2010.

[32] E. Bullmore and O. Sporns, "Complex brain networks: graph theoretical analysis of structural and functional systems," *Nat Rev Neurosci*, vol. 10, no. 4, pp. 312–312, 04 2009.

[33] S. Dasgupta, P. Manoonpong, and F. Woergoetter, "Small world topology of dynamic reservoir for effective solution of memory guided tasks," *Frontiers in Computational Neuroscience*, no. 177.

[34] M. Massar and S. Massar, "Mean-field theory of echo state networks," *Phys. Rev. E*, vol. 87, p. 042809, Apr 2013.

[35] M. Hagan and M.-B. Menhaj, "Training feedforward networks with the marquardt algorithm," *Neural Networks, IEEE Transactions on*, vol. 5, no. 6, pp. 989–993, 1994.

[36] H. Jaeger, M. Lukoševičius, D. Popovici, and U. Siewert, "Optimization and applications of echo state networks with leaky- integrator neurons," *Neural Networks*, vol. 20, no. 3, pp. 335–352, 2007.

[37] S. Dasgupta, F. Wrgtter, and P. Manoonpong, "Information theoretic self-organised adaptation in reservoirs for temporal memory tasks," in *Engineering Applications of Neural Networks*, ser. Communications in Computer and Information Science, C. Jayne, S. Yue, and L. Iliadis, Eds. Springer Berlin Heidelberg, 2012, vol. 311, pp. 31–40.

[38] J. Bi and K. P. Bennett, "Regression error characteristic curves." in *ICML*, T. Fawcett and N. Mishra, Eds. AAAI Press, 2003, pp. 43–50.

[39] W. Waegeman, B. De Baets, and L. Boullart, "A comparison of different ROC measures for ordinal regression," in *Proceedings of the 3rd International Workshop on ROC Analysis in Machine Learning.*, N. Lachiche, C. Ferri, and S. Macskassy, Eds., 2006, pp. 63–69.

[40] ——, "ROC analysis in ordinal regression learning," *Pattern Recogn. Lett.*, vol. 29, no. 1, pp. 1–9, Jan. 2008.

[41] J. Larsen, "A generalization error estimate for nonlinear systems,"

in *Neural Networks for Signal Processing, 1992. II., Proceedings of the 1992 IEEE Workshop on*, 1992, pp. 29–38.

[42] J. Larsen, C. Svarer, L. Andersen, and L. Hansen, "Adaptive regularization in neural network modeling," in *Neural Networks: Tricks of the Trade*, ser. Lecture Notes in Computer Science, G. Orr and K.-R. Mller, Eds. Springer Berlin Heidelberg, 1998, vol. 1524, pp. 113–132.

[43] J. Larsen, L. Hansen, C. Svarer, and M. Ohlsson, "Design and regularization of neural networks: the optimal use of a validation set," in *Neural Networks for Signal Processing, 1996. VI. Proceedings of the 1996 IEEE Workshop on*, 1996, pp. 62–71.

[44] J. Larsen and L. Hansen, "Empirical generalization assessment of neural network models," in *Neural Networks for Signal Processing, 1995. V. Proceedings of the 1995 IEEE Workshop on*, 1995, pp. 30–39.

[45] ——, "Generalization performance of regularized neural network models," in *Neural Networks for Signal Processing, 1994. IV. Proceedings of the 1994 IEEE Workshop on*, 1994, pp. 42–51.

[46] S. Lawrence, C. L. Giles, and A. C. Tsoi, "Lessons in neural network training: Overfitting may be harder than expected," in *In Proceedings of the Fourteenth National Conference on Artificial Intelligence, AAAI-97*. AAAI Press, 1997, pp. 540–545.

[47] N. Murata, S. Yoshizawa, and S.-I. Amari, "Network information criterion-determining the number of hidden units for an artificial neural network model," *Neural Networks, IEEE Transactions on*, vol. 5, no. 6, pp. 865–872, 1994.

[48] J. Moody, "Prediction risk and architecture selection for neural networks," in *From Statistics to Neural Networks*, ser. NATO ASI Series, V. Cherkassky, J. Friedman, and H. Wechsler, Eds. Springer Berlin Heidelberg, 1994, vol. 136, pp. 147–165.

[49] L. K. Hansen and C. E. Rasmussen, "Pruning from adaptive regularization," *Neural Computation*, vol. 6, no. 6, pp. 1223–1232, 2013/12/11 1994.

[50] B. R. Stallard and J. G. Taylor, "Quantifying multivariate classification performance: the problem of overfitting," pp. 426–436, 1999.

[51] D. M. Hawkins, "The problem of overfitting," *Journal of Chemical Information and Computer Sciences*, vol. 44, no. 1, pp. 1–12, 2004.

[52] E. Baum and D. Haussler, "What size net gives valid generalization?" *Neural Computation*, vol. 1, no. 1, pp. 151–160, 1989.

[53] B. Amirikian and H. Nishimura, "What size network is good for generalization of a specific task of interest?" *Neural Networks*, vol. 7, no. 2, pp. 321–329, 1994.

[54] M. Kearns, Y. Mansour, D. Ron, R. Rubinfeld, R. E. Schapire, and L. Sellie, "On the learnability of discrete distributions," in *Proceedings of the Twenty-sixth Annual ACM Symposium on Theory of Computing*, ser. STOC '94. New York, NY, USA: ACM, 1994, pp. 273–282.

[55] R. Lange and R. Männer, "Quantifying a critical training set size for generalization and overfitting using teacher neural networks," in *ICANN '94*, M. Marinaro and P. G. Morasso, Eds. Springer London, 1994, pp. 497–500.

[56] L. Valiant, "A theory of the learnable," *Communications of the ACM*, vol. 27, no. 11, pp. 1134–1142, 1984.