scientific reports

Check for updates

OPEN

Model-size reduction for reservoir computing by concatenating internal states through time

Yusuke Sakemi[1,2]✉, Kai Morino[1,3], Timothée Leleu[1,4] & Kazuyuki Aihara[1,4]

Reservoir computing (RC) is a machine learning algorithm that can learn complex time series from data very rapidly based on the use of high-dimensional dynamical systems, such as random networks of neurons, called "reservoirs." To implement RC in edge computing, it is highly important to reduce the amount of computational resources that RC requires. In this study, we propose methods that reduce the size of the reservoir by inputting the past or drifting states of the reservoir to the output layer at the current time step. To elucidate the mechanism of model-size reduction, the proposed methods are analyzed based on information processing capacity proposed by Dambre et al. (Sci Rep 2:514, 2012). In addition, we evaluate the effectiveness of the proposed methods on time-series prediction tasks: the generalized Hénon-map and NARMA. On these tasks, we found that the proposed methods were able to reduce the size of the reservoir up to one tenth without a substantial increase in regression error.

Efficiently processing time-series data is important for various tasks, such as time-series forecasting, anomaly detection, natural language processing, and system control. Recently, machine-learning approaches for these tasks have attracted much attention of researchers and engineers because they not only require little domain knowledge but also often perform better than traditional approaches. In particular, machine-learning models that employ recurrent neural networks, such as long short-term memory, have achieved great success in natural language processing and speech recognition[1], and their fields of applications continue to expand. However, the standard learning algorithms for recurrent neural networks, which include backpropagation through time[2] and its variants[3], require large computational resources. These computational burdens often hinder real-world applications, especially when computing is performed near end users or data sources instead of data centers. Such computing has been attracting considerable interest because the amount of data often exceeds the network bandwidth capacity, which leads to network congestion and makes it difficult to efficiently send data to data centers. In addition, transferring personal data across networks is often avoided due to privacy issues. This new computing paradigm is called "edge computing," which is characterized by limited computational power and limited battery capacity[4,5].

Reservoir computing (RC) is a machine-learning algorithm that aims to reduce the computational resources required for predicting time series without reducing accuracy. As shown in Fig. 1, a typical RC consists of three parts: an input layer, a "*reservoir*" layer where neurons are randomly connected, and an output layer[6,7]. Because only the weights between the reservoir layer and the output layer are trained while the other weights remain fixed, the learning process of RC is much faster than that of backpropagation through time[8–10]. Therefore, RC is expected to be a lightweight machine-learning algorithm that enables machine learning in edge computing[11].

The RC training process is fast and accurate. In addition, RC has shown high performance on various time-series forecasting tasks, including chaotic time-series[12–14], weather[15], wind-power generation[16], and finance[17]. Moreover, the range of applications of RC has extended into control engineering[18,19] and video processing[20–22].

To develop the applications for RC in edge computing, its hardware implementation must be improved to enhance its computational speed and energy efficiency. For realizing such efficient hardware implementation, variants of RC models, some of which employ delay-feedback systems[23], simple network topologies such as ring-topology and delay lines[24–26], and billiard systems[27], have been proposed. Efficient hardware based on these variants have been implemented using field programmable integrated gate arrays (FPGAs)[28–31]. Moreover,

[1]Institute of Industrial Science, The University of Tokyo, 4-6-1 Komaba Meguro-ku, Tokyo 153-8505, Japan. [2]NEC Corporation, 1753 Shimonumabe Nakahara-ku, Kanagawa 211-8666, Japan. [3]Interdisciplinary Graduate School of Engineering Sciences, Kyushu University, 6-1 Kasuga-Koen, Kasuga-shi, Fukuoka 816-8580, Japan. [4]International Research Center for Neurointelligence (WPI-IRCN), The University of Tokyo Institutes for Advanced Study, The University of Tokyo, Tokyo 113-0033, Japan. ✉email: sakemi@iis.u-tokyo.ac.jp

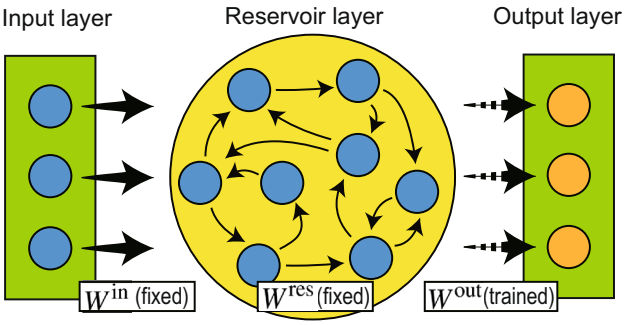

Figure 1. Typical RC architecture. The reservoir layer consists of randomly connected neurons. The connections between the input and reservoir layer W^{in} and connections within the reservoir layer W^{res} are fixed (solid arrows), whereas the output weights W^{out} are trained (dashed arrows).

numerous types of implementation employing physical systems, such as photonics[32–34], spintronics[35], mechanical oscillators[36], and analog integrated electronic circuits[37,38], have been demonstrated[39]. Although these implementations have exhibited the superiority of RC in computational speed and energy efficiency, the maximum size of the reservoir and, in turn, the forecasting accuracy, is limited by the physical size of the hardware.

In this study, we propose three methods that reduce the size of the reservoir without any performance impairment. The three methods share the concept that the number of the effective dimension of the reservoir is increased by allowing additional connections from the reservoir layer at multiple time steps to the output layer at the current time step. We analyze the mechanism of the proposed methods based on the *information processing capacity* (IPC) proposed by Dambre et al.[40]. We also demonstrate how the proposed methods reduce the size of the reservoir in the generalized Hénon-map and NARMA tasks.

Results

RC framework. In the mathematical representation of RC, four vector variables are defined as follows: $\boldsymbol{u}(t) \in \mathbb{R}^{N^{\text{in}}}$ for the inputs, $\boldsymbol{x}(t) \in \mathbb{R}^{N^{\text{res}}}$ for the states of the reservoir, $\boldsymbol{y}(t) \in \mathbb{R}^{N^{\text{out}}}$ for the outputs, and $\boldsymbol{y}^{\text{tc}}(t) \in \mathbb{R}^{N^{\text{out}}}$ for the teaching signals. The constants N^{in}, N^{res}, and N^{out} are the dimensions of the inputs, states of the reservoir, and outputs, respectively. The updates of the reservoir states are given by

$$\boldsymbol{x}(t) = \tanh\left(W^{\text{res}}\boldsymbol{x}(t-1) + W^{\text{in}}\boldsymbol{u}(t)\right), \tag{1}$$

where $W^{\text{in}} \in \mathbb{R}^{N^{\text{res}} \times N^{\text{in}}}$ is a weight matrix representing the connections from the neurons in the input layer to those in the reservoir layer. Its elements are independently drawn from uniform distribution $U(-\rho^{\text{in}}, \rho^{\text{in}})$, where ρ^{in} is a positive constant. Another weight matrix $W^{\text{res}} \in \mathbb{R}^{N^{\text{res}} \times N^{\text{res}}}$ represents the connections among the neurons in the reservoir layer. Its elements are initialized by drawing values from uniform distribution $U(-1, 1)$ and subsequently divided by a positive value to ensure that the spectral radius of W^{res} is ρ^{res}. Note that elements in matrices W^{in} and W^{res} are fixed to the initialized values. The outputs are obtained by

$$\boldsymbol{y}(t) = W^{\text{out}}\boldsymbol{x}(t), \tag{2}$$

where $W^{\text{out}} \in \mathbb{R}^{N^{\text{out}} \times N^{\text{res}}}$ is a weight matrix representing connections from the neurons in the reservoir layer to those in the output layer. The output weight matrix W^{out} is trained in the offline learning process of RC by using the pseudoinverse (see "Methods" section).

Proposed methods. We propose three methods that modify the connections between the reservoir and output layers. We call these three methods (i) *delay-state concatenation*, (ii) *drift-state concatenation*, and (iii) *delay-state concatenation with transient states*. These methods share the idea that the number of the effective dimension of the reservoir is increased by allowing additional connections from the reservoir layer at multiple time steps to the output layer at the current time step. For the delay-state concatenation and delay-state concatenation with transient states, additional connections are formed from the past states of the reservoir layer to the current output layer, as illustrated in Fig. 2a,d. On the other hand, for the drift-state concatenation, additional connections are formed from newly introduced states of the reservoir, called drifting states, to the current output layer, as illustrated in Fig. 2b. The drifting states are obtained by updating the current states of the reservoir layer without input signals. In what follows, we mathematically formulate these three proposed methods.

First, we formulate the delay-state concatenation with concatenated states of the reservoir given by

$$\hat{\boldsymbol{x}}(t) := \begin{pmatrix} \boldsymbol{x}(t) \\ \boldsymbol{x}(t-Q) \\ \vdots \\ \boldsymbol{x}(t-PQ) \end{pmatrix}. \tag{3}$$

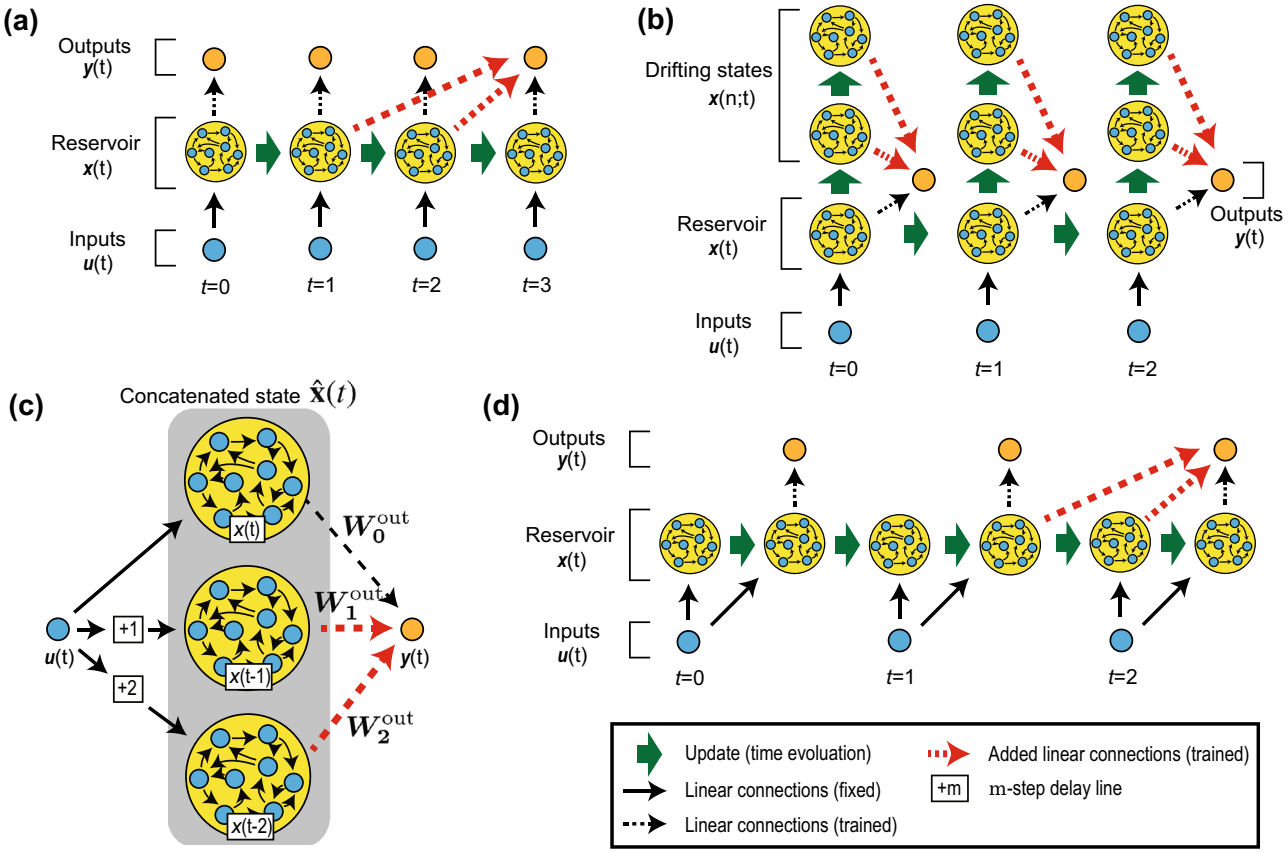

Figure 2. Schematics of the proposed methods. (**a**) Delay-state concatenation when the number of additional connections P is two and the unit of delay Q is one. (**b**) Drift-state concatenation when P is two and Q is one. (**c**) Another view of delay-state concatenation. The reservoir consists of three identical dynamical systems and delay lines. The added dynamical systems have $+1$ delay lines and $+2$ delay lines, respectively. (**d**) Delay-state concatenation with one transient state when P is two and Q is one.

Note that $\boldsymbol{x}(t)$ is a column vector and the number of neurons in the reservoir does not change. A positive integer Q represents the unit of delays. Another positive integer P represents the number of past states that are concatenated to the current states. The outputs are obtained with the concatenated states $\hat{\boldsymbol{x}}(t)$ and the corresponding output-weight matrix $\hat{W}^{\text{out}}$ as follows:

$$\hat{W}^{\text{out}} := \left(W_0^{\text{out}}\ W_1^{\text{out}}\ \cdots\ W_P^{\text{out}}\right), \tag{4}$$

$$\begin{aligned}\boldsymbol{y}(t) &= \sum_{i=0}^{P} W_i^{\text{out}} \boldsymbol{x}(t - iQ) \\ &= \hat{W}^{\text{out}} \hat{\boldsymbol{x}}(t).\end{aligned} \tag{5}$$

Here, $\hat{\boldsymbol{x}}(t)$ and $\hat{W}^{\text{out}}$ are defined in $\mathbb{R}^{(P+1)N^{\text{res}}}$ and $\mathbb{R}^{N^{\text{out}} \times (P+1)N^{\text{res}}}$, respectively. Figure 2a shows a schematic of this method illustrating the prediction of $\boldsymbol{y}(3)$ when $Q = 1$ and $P = 2$. One can see that there are additional connections from the past states of the reservoir $\boldsymbol{x}(1)$ and $\boldsymbol{x}(2)$ to output $\boldsymbol{y}(3)$, as indicated by red dashed arrows. From a different point of view, this model can be illustrated using the concatenated states of the reservoir $\hat{\boldsymbol{x}}(t)$, as in Fig. 2c, where the reservoir consists of three identical smaller reservoirs, each with a different time delay of 0, 1, and 2 from the inputs. Evidently, the effective dimension of the concatenated reservoir is three times larger than that of the original reservoir. Because the learning performance is enhanced by using a larger reservoir[9], the proposed method should be able to increase the computing capability without needing to add neurons in the reservoir.

Second, we formulate drift-state concatenation, as illustrated in Fig. 2b by introducing the drifting states of the reservoir given by

$$\boldsymbol{x}^{\text{drift}}(t'; t) = \begin{cases} \tanh\left(W^{\text{drift}} \boldsymbol{x}(t)\right), & (\text{if } t' = 1), \\ \tanh\left(W^{\text{drift}} \boldsymbol{x}^{\text{drift}}(t' - 1; t)\right), & (\text{if } t' \geq 2), \end{cases} \tag{6}$$

(a)

Time step	t	$t+1$	$t+2$
States	$\boldsymbol{x}(t)$	$\boldsymbol{x}(t+1)$	$\boldsymbol{x}(t+2)$
Output components	$W_0^{\text{out}}\boldsymbol{x}(t)$	$W_0^{\text{out}}\boldsymbol{x}(t+1)$	$W_0^{\text{out}}\boldsymbol{x}(t+2)$
	$W_1^{\text{out}}\boldsymbol{x}(t)$	$W_1^{\text{out}}\boldsymbol{x}(t+1)$	$W_1^{\text{out}}\boldsymbol{x}(t+2)$
	$W_2^{\text{out}}\boldsymbol{x}(t)$	$W_2^{\text{out}}\boldsymbol{x}(t+1)$	$W_2^{\text{out}}\boldsymbol{x}(t+2)$

(b)

Time step	t	$t+1$	$t+2$
States	$\boldsymbol{x}(t)$	$\boldsymbol{x}(t+1)$	$\boldsymbol{x}(t+2)$
Output components	$W_0^{\text{out}}\boldsymbol{x}(t)$	$W_0^{\text{out}}\boldsymbol{x}(t+1)$	$W_0^{\text{out}}\boldsymbol{x}(t+2)$
	$W_1^{\text{out}}\boldsymbol{x}^{\text{drift}}(1;t)$	$W_1^{\text{out}}\boldsymbol{x}^{\text{drift}}(1;t+1)$	$W_1^{\text{out}}\boldsymbol{x}^{\text{drift}}(1;t+2)$
	$W_2^{\text{out}}\boldsymbol{x}^{\text{drift}}(2;t)$	$W_2^{\text{out}}\boldsymbol{x}^{\text{drift}}(2;t+1)$	$W_2^{\text{out}}\boldsymbol{x}^{\text{drift}}(2;t+2)$

(c)

Method	Reservoir states	Output components	Reservoir weights	Output weights
(i)	$(P+1)N^{\text{res}}$	N/A	$\left((P+1)N^{\text{res}}\right)^2$	$(P+1)N^{\text{res}}N^{\text{out}}$
(ii)	N^{res}	$(P+1)(PQ+2)N^{\text{out}}/2$	$(N^{\text{res}})^2$	$(P+1)N^{\text{res}}N^{\text{out}}$
(iii)	N^{res}	$(P+1)(PQ+2)N^{\text{out}}/2$	$(N^{\text{res}})^2$	$(P+1)N^{\text{res}}N^{\text{out}}$
(iv)	$2N^{\text{res}}$	$(P+1)N^{\text{out}}$	$2(N^{\text{res}})^2$	$(P+1)N^{\text{res}}N^{\text{out}}$

Figure 3. Memory requirements. (**a**) Memory requirement for delay-state concatenation when P is two and Q is one. Output components that are stored in memory at time step $t+2$ for computing outputs at this time step or future time steps are masked with blue color. (**b**) Memory requirement for drift-state concatenation when P is two. Output components that are stored in memory at time step $t+2$ for computing outputs at this time step are masked with blue color. (**c**) Comparison of memory requirement for (i) standard RC[8], (ii) delay-state concatenation, (iii) delay-state concatenation with transient states, and (iv) drift-state concatenation. Values in each column represent the amounts of memory required to store vectors. The effective dimension of RC is $(P+1)N^{\text{res}}$ for all the cases.

where $\boldsymbol{x}^{\text{drift}}(t';t)$ represents the drifting states of the reservoir and t' is the time step after the current time step t. Using the drifting states, we redefine the concatenated states of the reservoir and the corresponding output matrix as follows:

$$\hat{\boldsymbol{x}}(t) := \begin{pmatrix} \boldsymbol{x}(t) \\ \boldsymbol{x}^{\text{drift}}(1;t) \\ \boldsymbol{x}^{\text{drift}}(2;t) \\ \vdots \\ \boldsymbol{x}^{\text{drift}}(P;t) \end{pmatrix}, \tag{7}$$

$$\hat{W}^{\text{out}} := \begin{pmatrix} W_0^{\text{out}} & W_1^{\text{out}} & \cdots & W_P^{\text{out}} \end{pmatrix}, \tag{8}$$

$$\begin{aligned} \boldsymbol{y}(t) &= W_0^{\text{out}}\boldsymbol{x}(t) + \sum_{i=1}^{P} W_i^{\text{out}}\boldsymbol{x}^{\text{drift}}(i;t) \\ &= \hat{W}^{\text{out}}\hat{\boldsymbol{x}}(t). \end{aligned} \tag{9}$$

Here, W^{drift} is a matrix representing the connections within the reservoir to obtain the drifting states, and its elements are drawn from uniform distribution $U(-1,1)$ divided by a positive value to ensure that the spectral radius of W^{drift} is ρ^{drift}.

Third, delay-state concatenation with transient states introduces transient states to the delay-state concatenation, as illustrated in Fig. 2d, where the states of the reservoir update twice (so it has one transient state) during the inputs and the outputs update once (see "Methods" section).

Although we have shown that the proposed methods can increase the effective dimension of the reservoir without adding neurons, one potential drawback of the methods is the cost of the memory required to store the past reservoir states. However, the proposed methods are very memory efficient. To carry out RC with delay-state concatenation, the output components

$$W_0^{\text{out}}\boldsymbol{x}(t),\ W_1^{\text{out}}\boldsymbol{x}(t),\ldots,\text{ and } W_P^{\text{out}}\boldsymbol{x}(t) \tag{10}$$

are computed and stored in memory at time step t. The dimensions of these vectors are all N^{out}. As shown in Fig. 3a, the vector $W_i^{\text{out}}\boldsymbol{x}(t)$ must be stored until they are used for calculating the outputs at time steps $t+iQ$. Therefore, the total memory cost is obtained as follows:

$$\begin{aligned} &N^{\text{out}} + (Q+1)N^{\text{out}} + (2Q+1)N^{\text{out}} + \cdots + (PQ+1)N^{\text{out}} \\ &= \frac{(P+1)(PQ+2)N^{\text{out}}}{2}. \end{aligned} \tag{11}$$

Note that the memory required to store the states of the reservoir is proportional to N^{res}. The memory required to store the weights within the reservoir and the output weights is proportional to $(N^{\mathrm{res}})^2$ and $(P+1)N^{\mathrm{res}}N^{\mathrm{out}}$, respectively, given that the reservoir is fully connected. Although delay-state concatenation with transient states requires additional computation for transient states, the cost of memory is the same as that for delay-state concatenation.

To carry out RC with drift-state concatenation, the output components

$$W_0^{\mathrm{out}}\boldsymbol{x}(t),\ W_1^{\mathrm{out}}\boldsymbol{x}^{\mathrm{drift}}(1;t),\ldots,\ \text{and}\ W_P^{\mathrm{out}}\boldsymbol{x}^{\mathrm{drift}}(P;t) \tag{12}$$

are computed and stored in memory at time step t, leading to the memory cost of $N^{\mathrm{out}}(P+1)$ as shown in Fig. 3b. After calculating all the drifting states at time step t, the states $\boldsymbol{x}(t+1)$ at the next time step are calculated using the states $\boldsymbol{x}(t)$ at the current time step (see Fig. 1b). Therefore, this method requires additional memory to store $\boldsymbol{x}(t)$ corresponding to the memory cost of N^{res}. Note that this memory cost is independent of the number of P. The cost of the memory required to store the weights within the reservoir including drifting states is $2(N^{\mathrm{res}})^2$ and that required to store the output weights is $(P+1)N^{\mathrm{res}}N^{\mathrm{out}}$, given that the reservoir is fully connected.

If the number of neurons is increased by $(P+1)$ times using standard RC[8], the memory cost to store the reservoir states becomes $(P+1)N^{\mathrm{res}}$ and the memory cost required to store the weights within the reservoir and output weights becomes $((P+1)N^{\mathrm{res}})^2$ and $(P+1)N^{\mathrm{res}}N^{\mathrm{out}}$, respectively, given that the reservoir is fully connected. The memory required for different methods is compared in Fig. 3c.

According to Fig. 3c, for moderate values of P and Q (typically less than 5), the total memory cost for the proposed methods is much less than that for standard RC because $N^{\mathrm{out}} \ll N^{\mathrm{res}}$ for typical RC applications. Therefore, the proposed methods can increase the dimensions of the reservoir more efficiently than by simply increasing the number of neurons in the reservoir. It should be noted that the proposed methods increase the effective dimension of the reservoir state by a factor of $(P+1)$, but the number of the neurons in the reservoir is not increased. We also note that the discussion of memory reduction only applies to the inference phase because states of the reservoir at all time steps must be stored in the training phase.

Quantitative analysis based on IPC. Before benchmarking the proposed RC, we quantitatively analyze the learning capacity of the RC to elucidate how the proposed methods work.

The *memory capacity* (MC) is a performance measure commonly used in the RC research community[41]. The MC represents how precisely the RC system can reproduce the past inputs. A number of studies have shown that the MC is theoretically bounded by the number of neurons in the reservoir, and the MC can reach this bound in some situations[25,26,42]. Boedecker et al.[43] evaluated the MC at the edge of chaos, which is a region in the model parameter space where RC is stable but near to unstable. Farkaš et al.[44] have evaluated the MC for various model parameters. Ganguli et al.[45] extended the concept of MC using Fischer information. However, the MC does not evaluate how well RC processes information in a nonlinear way. Because many tasks in the real world targeted by RC are nonlinear problems, the MC is not a suitable measure for analyzing the proposed methods in this sense. Therefore, to elucidate the mechanism of the methods proposed in this study, we employed another criterion[40] called the *information processing capacity* (IPC), which handles nonlinear tasks.

The IPC is a measure that integrates both memory and information processing performance. By employing an orthogonal basis set that spans the Hilbert space, one IPC can be obtained from one corresponding basis. The IPC can be interpreted as a quantity that represents not only how well the network can memorize past inputs but also how precisely the network can convert inputs into the target outputs in a nonlinear manner given the basis set. Dambre et al.[40] showed that the total IPC C^{total}, which is a sum of all IPCs, is identical to the number of neurons in the reservoir, provided (I) the inputs are independent and identically distributed (*i.i.d.*), (II) the fading memory condition is satisfied, and (III) all the neurons are linearly independent (see Theorem 7 and its proof in Ref.[40]). By analyzing the IPCs, one can obtain a large amount of information about how RC processes input data. For example, the degree of nonlinearity of the information processing carried out in RC can be analyzed by calculating multi-order IPCs. The kth-order IPC $C^{k\mathrm{th}}$ is defined as the sum of the IPCs corresponding to the subset of a basis with kth-order nonlinearity. Based on the IPC, informative results such as the memory–nonlinearity tradeoff have been obtained[40]. Therefore, using the IPC, we can analyze how RC stores and processes information in the reservoir as well as how the proposed methods affect the way information is processed.

We calculated the IPCs for the standard RC and for RC with the proposed methods (see "Methods" section). Figure 4 shows the IPCs when $N^{\mathrm{res}}=12$ and $N^{\mathrm{res}}=24$ for various values of ρ^{in} and ρ^{res}. Note that only odd-order IPCs were observed because of the symmetry. In each setting, we calculated the IPCs for the standard RC (left columns), those for the RC with delay-state concatenation with $P=Q=1$ (center columns), and those for the RC with drift-state concatenation with $P=Q=1$ (right columns). One can find that the value of $C^{\mathrm{total}}/(P+1)$ almost reaches the number of neurons N^{res} in reservoir, except when $\rho^{\mathrm{res}}=1.05$. This result indicates that the proposed methods actually increase the total IPC by $(P+1)$ times. The observed lower values of the total IPCs for the case of $\rho^{\mathrm{res}}=1.05$ can be attributed to the failure-of-fading-memory condition. In the RC research community, it is well known that the dynamics of RC is more likely to be chaotic when ρ^{res} increases (typically occurring when ρ^{res} is larger than 1)[46], and this corresponds to the failure of fading memory. For all cases, as ρ^{in} increases, the third-order IPC $C^{3\mathrm{rd}}$ and the fifth-order IPC $C^{5\mathrm{th}}$ tend to increase, which reflects the increase in nonlinearity in the reservoir[40] given the selected basis set. Note that as ρ^{in} increases, the total IPC C_{total} tends to decrease, which is attributed to increases in the importance of higher-order IPCs (e.g., seventh-order and ninth-order IPCs).

In Fig. 5a, we show the IPCs for delay-state concatenation with $P=1$ for several values of the unit of delay Q. As Q increases, the first-order IPC increases as well. This result may be trivial because RC with large Q can access the past states of the reservoir, rendering the reproduction of the past inputs easy. To investigate the effects

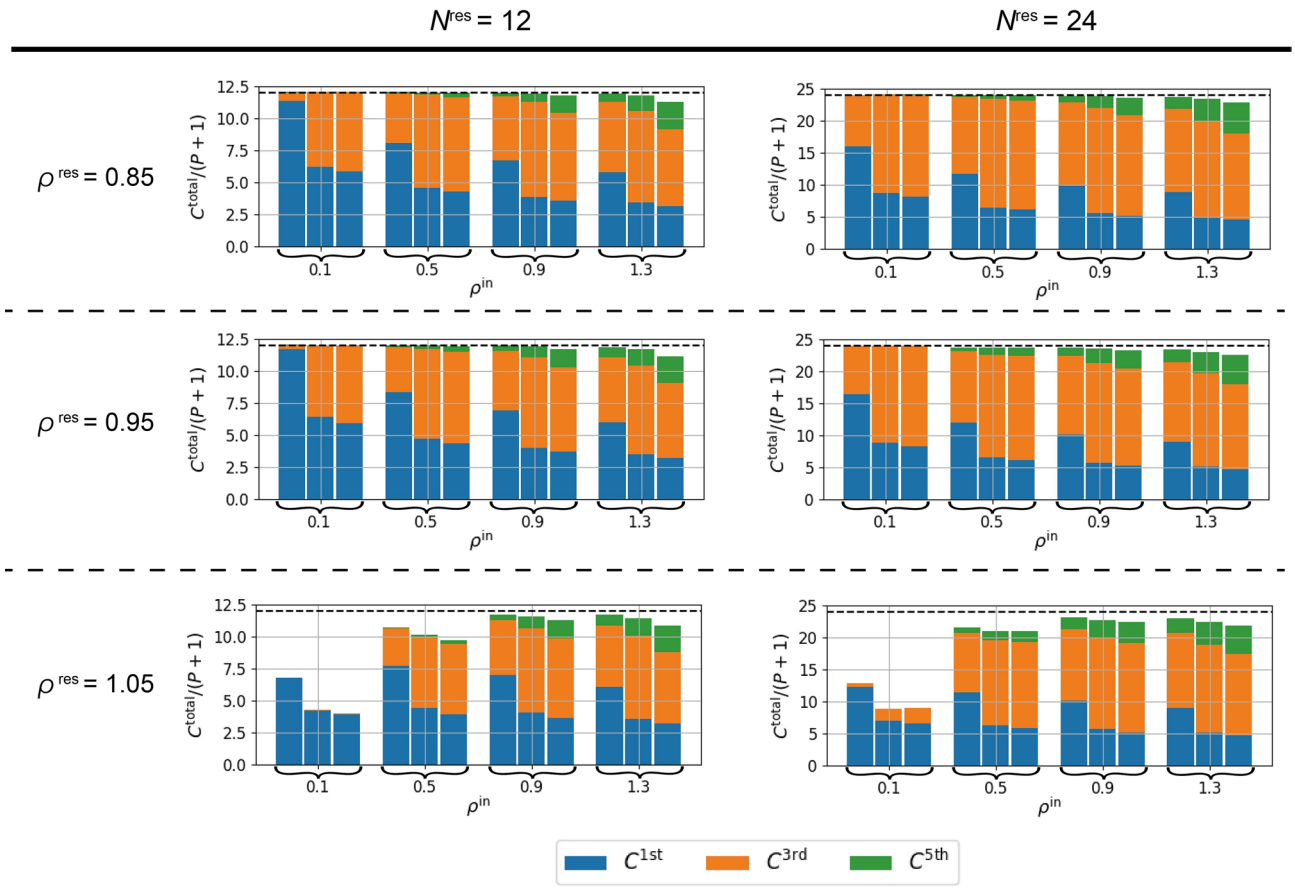

Figure 4. Analysis of IPCs. IPCs of standard RC and RC with the proposed methods for various values of input weight strength ρ^{in} and spectral radius ρ^{res}. The left and right panels show the results for $N^{res} = 12$ and $N^{res} = 24$, respectively. The dashed horizontal line within each subgraph represents the value of N^{res}. For each value of ρ^{in}, the left column presents the standard RC with $P = 0$, the center column presents delay-state concatenation with $P = Q = 1$, and the right column presents drift-state concatenation with $P = 1$.

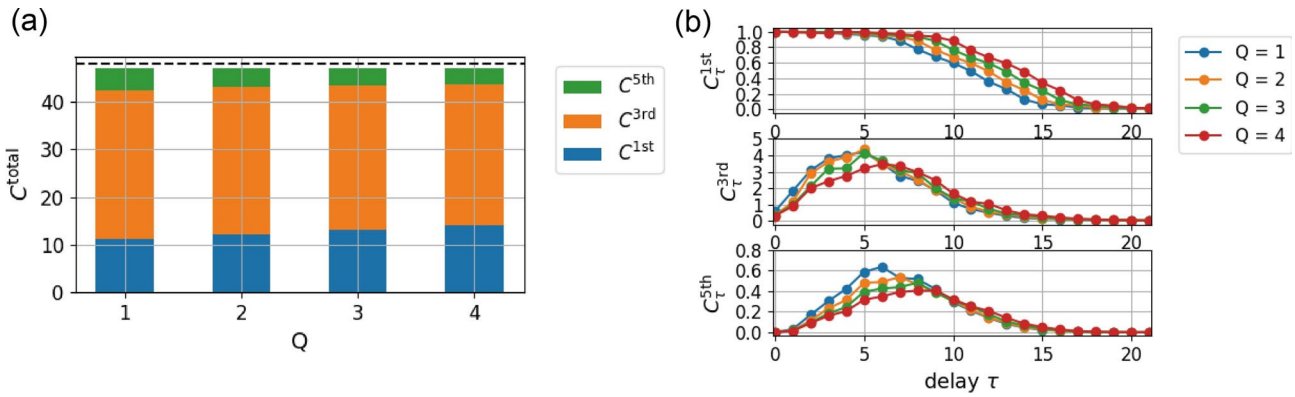

Figure 5. Q-dependence of IPCs. (**a**) IPCs of the RC with delay-state concatenation for various values of Q. (**b**) Delay structures of the IPCs for various values of Q. From top to bottom, the first-, third-, and fifth-order IPCs are shown. The parameters are set as follows: $N^{res} = 24$, $\rho^{in} = 0.9$, $\rho^{res} = 0.95$, and $P = 1$.

of the unit of delay Q on IPCs, we decomposed the kth-order IPC C^{kth} into components in terms of their delay such as C_0^{kth}, C_1^{kth}, and C_2^{kth}, which correspond to a different subset of the basis. Figure 5b shows the distributions of the delay components for four values of Q under the same experimental conditions. As the values of Q increase, the distribution tends to shift to the right (larger delays) for each order of IPC. This fact indicates that, as demonstrated in the subsequent section, one can tune RC models by adjusting the value of Q according to the delay structure of the target tasks.

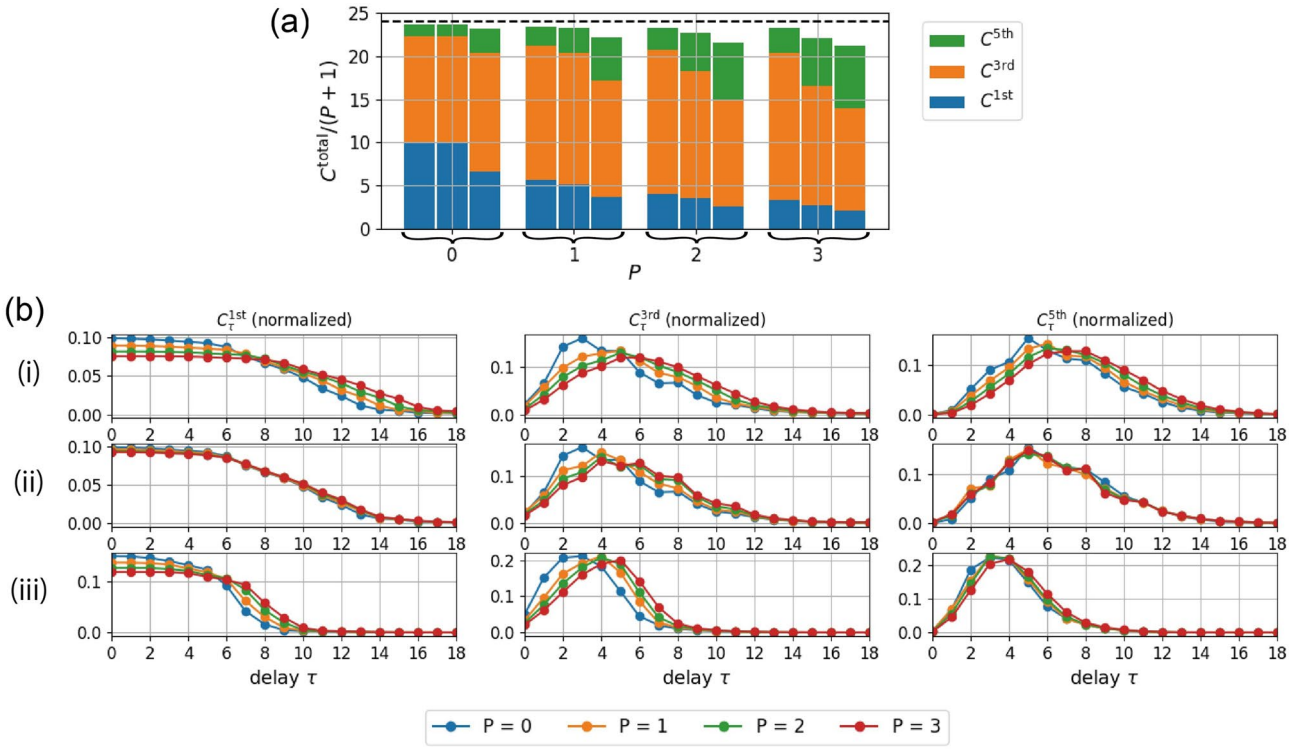

Figure 6. *P*-dependence of IPCs. (**a**) IPCs with the proposed methods for various values of *P*. Left columns: IPCs for delay-state concatenation, center columns: IPCs for drift-state concatenation, and right columns: IPCs for delay-state concatenation with one transient state. (**b**) Comparison of the delay structures of the IPCs for the three proposed methods. Top panels (i): delay structures for delay-state concatenation, middle panels (ii): delay structures for drift-state concatenation, and bottom panels (iii): delay structures for delay-state concatenation with one transient state. To highlight the difference in the distribution of the delay structures, the IPCs for each order are normalized. From left to right, the first-, third-, and fifth-order IPCs are shown. The parameters are set as follows: $N^{\text{res}} = 24$, $\rho^{\text{in}} = 0.9$, $\rho^{\text{res}} = 0.95$, and $Q = 1$.

Next, for various values of *P*, we show the IPCs in Fig. 6a for delay-state concatenation, drift-state concatenation, and delay-state concatenation with one transient state. For all three proposed methods, the contributions of higher-order IPCs tend to be dominant in the total IPCs as *P* increases. In Fig. 6b, we show the delay structures of the IPCs. Note that to clarify how the distributions of the delay structure change as the value of *P* changes, we used the normalized IPC $C_\tau^{n\text{th}}/(P+1)\sum_\tau C_\tau^{n\text{th}}$. The top panels in Fig. 6b show that, as *P* increases, the distribution of the delay structure of the IPCs for delay-state concatenation tends to shift to the right (larger delays). Conversely, in the middle panels, the IPCs for drift-state concatenation do not change significantly. These results may be explained as follows: the increase in *P* for delay-state concatenation increases the memory of past inputs because of the additional connections from the past states of the reservoir, whereas the increase in *P* for drift-state concatenation does not increase the memory of the past inputs because drifting states are obtained from the current states of the reservoir. The bottom panels of this figure show that the distribution of IPCs for delay-state concatenation with one transient state tends to shift to larger delays as *P* increases. However, the delays in this distribution are smaller than the delays in the distribution obtained using delay-state concatenation. This difference may stem from the fact that the information of past states stored in the reservoir is more likely to be thrown away and to be replaced with that of more recent states in delay-state concatenation with one transient state because the RC model in this case carries out nonlinear transformation twice for each input (see Fig. 2d).

Here, we present a short summary of the above experiments. We have numerically shown that the total IPCs divided by $P+1$ are almost independent of the values of *Q* and *P*, which is consistent with the theory in Ref.[40]. We note that when the value of *P* approaches the number of simulation time steps, the IPCs do not effectively increase because the internal states are no longer linearly independent. Furthermore, we have found that the importance among IPC components and the delay structure of IPCs can be modified by selecting the values of *Q* and *P*. These findings indicate that the learning performance on real-world tasks may be enhanced by selecting appropriate values of *Q* and *P* adjusted to a target task with a specific temporal structure.

Effectiveness on complex data. Although we have shown that the proposed methods can increase the IPCs efficiently, the conditions assumed above are not always guaranteed in real-world applications; for example, inputs may not be drawn from *i.i.d.* data, and neurons in the reservoir may not be linearly independent. Therefore, the IPCs are just a guide that help us understand the mechanisms of the proposed methods. In this section, to evaluate the effectiveness of the proposed methods on complex data, we applied them to two prediction tasks: generalized Hénon-map tasks and NARMA tasks (see Eqs. (25) and (26) in "Methods" section). In the following

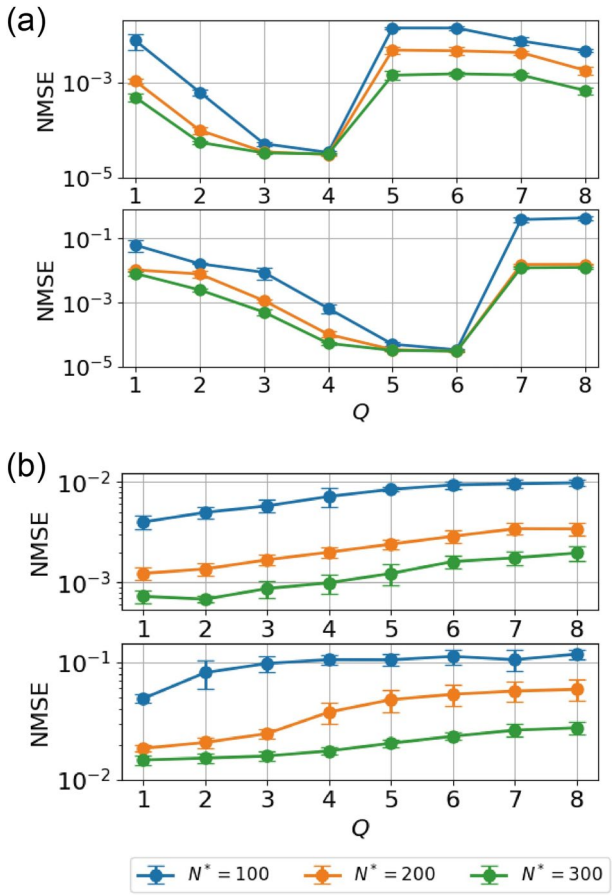

Figure 7. Q-dependence of the learning performance. (**a**) NMSE for a sixth-order Hénon-map task (the upper panel) and for an eighth-order Hénon-map task (the lower panel) for various values of Q with $P = 1$. (**b**) NMSE for NARMA5 (the upper panel) task and for NARMA10 (the lower panel) task for various values of Q with $P = 1$. Error bars show the standard deviation of the results of 10 trials.

experiments, the dimensions of the reservoir are approximately given by an integer N^*, and the actual number of neurons in the reservoir is given by $N^{\mathrm{res}} = \lfloor N^*/(P+1) \rfloor$, where $\lfloor m \rfloor := \max\{q \in \mathbb{Z} | q \leq m\}$. We optimized the model parameters, ρ^{in}, ρ^{res}, and ρ^{drift} with Bayesian optimization[47,48].

We first investigated the effects of the value of Q in the delay-state concatenation method. Figure 7a shows the normalized mean-squared errors (NMSEs) for the sixth-order and eighth-order Hénon-map tasks for various values of Q with $P = 1$. As the value of Q increases, the NMSE first decreases, but abruptly increases when Q is larger than 4 in the sixth-order Hénon-map and larger than 6 in the eighth-order Hénon-map. Considering the fact that to predict the output, the mth-order Hénon-map has two informative inputs at the mth and $(m-1)$th previous steps, these increases in performance are reasonable because an RC system with the appropriate values of Q can possess the information needed from past inputs. We also note that similar results were obtained recently in Ref.[49]. In contrast, the NMSE monotonically increases as the value of Q increases for NARMA5 and NARMA10, as shown in Fig. 7b. These results imply that simply adjusting the value of Q is not effective for tasks such as NARMA5 and NARMA10 with complicated temporal structures.

We next investigated the effects of the value of P for the NARMA tasks in which adjusting the value of Q was not effective. Figure 8 shows the NMSE against the values of P for the NARMA5 task (the top panel) and NARMA10 task (the bottom panel) for three settings of N^*. Note that N^* is fixed as P is varied to ensure that the size of reservoir N^{res} reduces to approximately $1/(P+1)$ times as P varies. We found that the NMSEs were almost constant up to specific values of P. For example, for the case of the NARMA10 task with $N^* = 300$, the NMSE was almost constant up to $P = 5$, indicating that the number of neurons in the reservoir can be reduced from 300 to 50 without impairing performance.

Finally, we compared the proposed methods, delay-state concatenation with and without one transient state, and drift-state concatenation on the NARMA10 task. Figure 9 shows the NMSE as functions of P values with $N^* = 200$ and 300 for the proposed methods. We found that the NMSEs for delay-state concatenation with one transient state and drift-state concatenation were lower than those for delay-state concatenation when P was larger than 7. For the NARMA10 task, inputs more than 10 steps in the past are not very informative for predicting one step forward. The lack of information in the past steps may explain the increase in NMSE for delay-state concatenation when P is larger than 7. The observed lower NMSEs for delay-state concatenation with

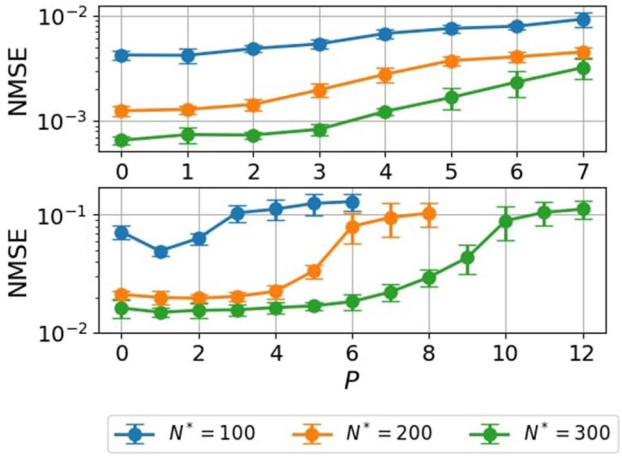

Figure 8. *P*-dependence of the learning performance. NMSEs for the NARMA5 (the top panel) and NARMA10 (the bottom panel) tasks with delay-state concatenation as functions of P for various values of N^*. Error bars show the standard deviation of the results of 10 trials.

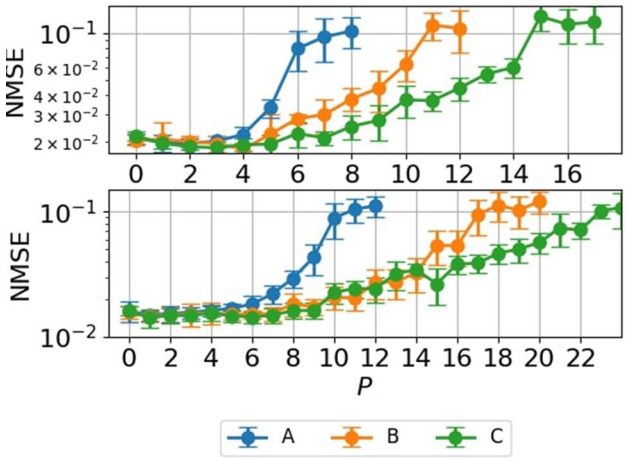

Figure 9. Comparison of the proposed methods. Results of regression performance on the NARMA10 task for various values of P with $N^* = 200$ (the top panel) and $N^* = 300$ (the bottom panel) for (**A**) delay-state concatenation, (**B**) delay-state concatenation with one transient state, and (**C**) drift-state concatenation. Error bars show the standard deviation of the results of 10 trials.

one transient state and drift-state concatenation are also reasonable because (i) the former method uses the past states of the reservoir, which contain more recent input information for the same value of P (see Fig. 2d), and (ii) the latter method uses the states of the reservoir, which contain current input information.

Discussion

In this study, we proposed three methods to reduce the size of an RC reservoir without impairing performance.

To elucidate the mechanism of the proposed methods, we analyzed the IPC. We found that the value of the total IPCs almost reaches $N^{\mathrm{res}}(P+1)$ using the proposed methods, whereas the importance of their components (the first-, third-, and fifth-order IPCs) changes drastically. We also found that the delay structures of the IPCs depend on the values of Q and P. To investigate the applicability of the proposed methods on complex data, we presented the experimental results on generalized Hénon-map and NARMA tasks. We found that when the target task has a relatively simple temporal structure, as demonstrated with the Hénon-map tasks, selecting an appropriate value of Q enhances the performance substantially. In contrast, when the target task contains complex temporal structure, as demonstrated in the NARMA tasks, adjusting the value of Q does not enhance the performance. However, in those cases, we found that increasing the value of P can reduce the size of the reservoir without impairing performance. We have demonstrated that the number of neurons in the reservoir can be reduced by up to one tenth in the NARMA10 task.

Here, we note the relationship between our work and the relevant previous works[23,49]. In Ref.[23], the authors proposed the time-delay reservoir that consists of a single node with delayed feedback. The states of the single node at multiple time steps correspond to virtual nodes, which may look similar to the delay-state concatenation. However, these two methods are different because the virtual nodes have output connections only to the current outputs, whereas the delay-state concatenation adds output connections from the nodes in the reservoir not only to the current outputs but also to future outputs. We also note that the delay-state concatenation can be also applied to the time-delay reservoir. In Ref.[49], the authors proposed a method that is similar to delay-state concatenation. Their proposed model corresponds to the case when the number of additional connected past states is one (i.e., $P = 1$). They observed that performance enhancement depends on the value of Q, as we showed in this paper. However, to the best of our knowledge, the dependence of the performance on the value of P has not been reported. In addition, the other two proposed methods, drift-state concatenation and delay-state concatenation with transient states, are introduced for the first time in this paper. Moreover, the authors of Ref.[49] explained the mechanism of their proposed method in terms of the delayed embedding theorem[50]. In contrast, we have provided a more intuitive explanation based on the IPC[40].

Because the proposed methods do not assume a specific topology for the reservoir, they can readily be implemented in FPGAs and physical reservoir systems, such as photonic reservoirs[39]. Therefore, the proposed methods could be an important set of techniques that facilitates the introduction of RC in edge computing.

Methods

Training output weights. The training procedures are the same as those for standard RC models[9]. The output weights are trained by minimizing

$$\sum_{t=1}^{T} ||\boldsymbol{y}(t) - \boldsymbol{y}^{\mathrm{tc}}(t)||_2^2. \tag{13}$$

Adding a regularization term $\|\hat{W}^{\mathrm{out}}\|_2^2$ did not improve the performance in our case.

Delay-state concatenation with transient states. We inserted transient states in the RC system with delay-state concatenation as follows:

$$\begin{aligned}\boldsymbol{x}(t) = \tanh\Bigg(& W^{\mathrm{res}}\boldsymbol{x}(t-1) \\ & + W^{\mathrm{in}}\boldsymbol{u}\left(\left\lfloor \frac{t}{N^{\mathrm{tran}}+1} \right\rfloor\right)\Bigg),\end{aligned} \tag{14}$$

$$\hat{\boldsymbol{x}}(t) = \begin{pmatrix} \boldsymbol{x}(t) \\ \boldsymbol{x}(t-Q) \\ \vdots \\ \boldsymbol{x}(t-QP) \end{pmatrix}, \tag{15}$$

$$\boldsymbol{y}(t) = \hat{W}^{\mathrm{out}}\hat{\boldsymbol{x}}\big((N^{\mathrm{tran}}+1)t + N^{\mathrm{tran}}\big), \tag{16}$$

where N^{tran} is the number of inserted transient states.

IPC. Following the same procedure given in Ref.[40], the IPCs are calculated as follows: The total IPC is defined as

$$C^{\mathrm{total}} = \sum_{\{d_i\}} C(\{d_i\}), \tag{17}$$

where C^{total} is the total IPC and $C(\{d_i\})$ is the IPC for a basis represented with a list $\{d_i\} = \{d_0, d_1, \dots\}$. The list represents an orthogonal basis in Hilbert space. We employed the following Legendre polynomials as the orthogonal basis:

$$y_{\{d_i\}}(t) = \prod_{i=0}^{\tau_{\max}} P_{d_i}(u(t-i)), \tag{18}$$

where $P_{d_i}(\cdot)$ is the d_ith-order Legendre polynomial and $u(t)$ is drawn from uniform distribution on $[-1, 1]$. A constant $\tau_{\max}$ is the maximum delay, which must be large enough to converge the calculation. In our simulations, we set $\tau_{\max}$ to 50 for $\rho^{\mathrm{in}} = 0.1$ or otherwise to 25. Then, the IPC $C(\{d_i\})$ can be calculated as

$$C(\{d_i\}) = 1 - \frac{\left\langle |y(t) - y_{\{d_i\}}(t)|^2 \right\rangle}{\left\langle |y(t) - \langle y(t)\rangle|^2 \right\rangle}, \tag{19}$$

where $\langle y(t)\rangle := \frac{1}{T}\sum_{t=1}^{T} y(t)$. We set the simulation steps T to 10^6 in all experiments. To avoid overestimation, the value of $C(\{d_i\})$ was set to zero when the value is less than threshold of $7N^{\mathrm{res}}(P+1) \times 10^{-5}$.

We define the kth-order IPCs as

$$C^{k\text{th}} = \sum_{\{d_i\}\in\Gamma^k} C(\{d_i\}), \tag{20}$$

$$\Gamma^k = \left\{ \{d_i\} \middle| \sum_{i=0}^{\tau_{\max}} d_i = k \right\}. \tag{21}$$

The kth-order IPC was decomposed into components corresponding to a subset of a basis whose maximum delay is τ as follows:

$$C^{k\text{th}} = \sum_{\tau=0}^{\tau_{\max}} C_\tau^{k\text{th}}, \tag{22}$$

$$C_\tau^{k\text{th}} = \sum_{\{d_i\}\in\Gamma_\tau^k} C(\{d_i\}), \tag{23}$$

$$\Gamma_\tau^k = \left\{ \{d_i\} \middle| \sum_{i=0}^{\tau_{\max}} d_i = k, \max\{i|d_i \geq 1\} = \tau \right\}. \tag{24}$$

Dataset. The mth-order generalized Hénon-map[51] is given by

$$y^{\text{tc}}(t) = 1.76 - y^{\text{tc}}(t-m+1)^2 - 0.1y^{\text{tc}}(t-m) + \sigma(t)\ (m \geq 2), \tag{25}$$

where σ is Gaussian noise with zero mean and standard deviation of 0.05. The inputs and outputs of the RC are the time series of an n-dimensional generalized Hénon-map. The task is to predict one step forward $y(t+1)$ with past inputs $y(t), y(t-1), \ldots$. The NARMA time series is obtained with a nonlinear auto-regressive moving average as follows:

$$\begin{aligned} y^{\text{tc}}(t) = {} & 0.3y^{\text{tc}}(t-1) + 0.05y^{\text{tc}}(t-1)\sum_{i=1}^{m} y^{\text{tc}}(t-i) \\ & + 1.5s(t-m+1)s(t) + 0.1, \end{aligned} \tag{26}$$

where $s(t)$ is drawn from uniform distribution of $[0, 0.5]$. The NARMA5 and NARMA10 time series correspond to the case when $m = 5$ and $m = 10$, respectively. The inputs of the RC are $s(t)$. The task is to predict $y(t)$ from the inputs $s(t)$.

For both tasks, we used 2000 steps as a training dataset and used 3000 steps as a test dataset. We removed the first 200 steps (free run) both during the training and test phases to avoid the effects of the initial conditions in the reservoirs[9]. We evaluated the performance based on the normalized mean-squared error (NMSE) during the test phase following:

$$\text{NMSE} = \frac{\left\langle |y(t) - y^{\text{tc}}(t)|^2 \right\rangle}{\left\langle |y(t) - \langle y(t)\rangle|^2 \right\rangle}, \tag{27}$$

where $\langle y(t)\rangle = \frac{1}{T}\sum_{t=1}^{T} y(t)$. We averaged the NMSEs over 10 trials. For each iteration, the dataset and connection matrix of the reservoir were generated using their corresponding probabilistic distributions.

Received: 7 September 2020; Accepted: 23 November 2020
Published online: 11 December 2020

References

1. Greff, K., Srivastava, R. K., Koutnk, J., Steunebrink, B. R. & Schmidhuber, J. LSTM: A search space odyssey. *IEEE Trans. Neural Netw. Learn. Syst.* **28**, 2222–2232 (2017).
2. Werbos, P. J. Backpropagation through time: What it does and how to do it. *Proc. IEEE* **78**, 1550–1560 (1990).
3. Lillicrap, T. P. & Santoro, A. Backpropagation through time and the brain. *Curr. Opin. Neurobiol.* **55**, 82–89 (2019).
4. Shi, W., Cao, J., Zhang, Q., Li, Y. & Xu, L. Edge computing: Vision and challenges. *IEEE Internet Things J.* **3**, 637–646 (2016).
5. Zhou, Z. *et al.* Edge intelligence: Paving the last mile of artificial intelligence with edge computing. *Proc. IEEE* **107**, 1738–1762 (2019).
6. Jaeger, H. The echo state approach to analysing and training recurrent neural networks. *Technical Report GMD Report 148, German National Research Center for Information Technology* (2001).
7. Maass, W., Natschlger, T. & Markram, H. Real-time computing without stable states: A new framework for neural computation based on perturbations. *Neural Comput.* **14**, 2531–2560 (2002).
8. Lukoševičius, M. & Jaeger, H. Reservoir computing approaches to recurrent neural network training. *Comput. Sci. Rev.* **3**, 127–149 (2009).
9. Lukoševičius, M. A practical guide to applying echo state networks. In *Neural Networks: Tricks of the Trade* 659–686, (2012).
10. Scardapane, S. & Wang, D. Randomness in neural networks: An overview. *WIREs Data Min. Knowl. Discov.* **7**, e1200 (2017).

11. Soures, N., Merkel, C., Kudithipudi, D., Thiem, C. & McDonald, N. Reservoir computing in embedded systems: Three variants of the reservoir algorithm. *IEEE Consumer Electron. Mag.* **6**, 67–73 (2017).
12. Jaeger, H. & Haas, H. Harnessing nonlinearity: Predicting chaotic systems and saving energy in wireless communication. *Science* **304**, 78–80 (2004).
13. Pathak, J., Hunt, B., Girvan, M., Lu, Z. & Ott, E. Model-free prediction of large spatiotemporally chaotic systems from data: A reservoir computing approach. *Phys. Rev. Lett.* **120**, 024102 (2018).
14. Pathak, J. *et al.* Hybrid forecasting of chaotic processes: Using machine learning in conjunction with a knowledge-based model. *Chaos* **28**, 041101 (2018).
15. McDermott, P. L. & Wikle, C. K. An ensemble quadratic echo state network for non-linear spatio-temporal forecasting. *Stat* **6**, 315–330 (2017).
16. Tian, Z., Wang, G. & Ren, Y. Short-term wind speed forecasting based on autoregressive moving average with echo state network compensation. *Wind Eng.* **44**, 152–167 (2020).
17. Lin, X., Yang, Z. & Song, Y. Short-term stock price prediction based on echo state networks. *Expert Syst. Appl.* **36**, 7313–7317 (2009).
18. Tsai, C.-Y., Dutoit, X., Song, K.-T., Van Brussel, H. & Nuttin, M. Robust face tracking control of a mobile robot using self-tuning Kalman filter and echo state network. *Asian J. Control* **12**, 488–509 (2010).
19. Antonelo, E. A. & Schrauwen, B. On learning navigation behaviors for small mobile robots with reservoir computing architectures. *IEEE Trans. Neural Netw. Learn. Syst.* **26**, 763–780 (2015).
20. Jalalvand, A., Wallendael, G. V. & Walle, R. V. D. Real-time reservoir computing network-based systems for detection tasks on visual contents. In *2015 7th International Conference on Computational Intelligence, Communication Systems and Networks*, 146–151 (2015).
21. Buteneers, P. *et al.* Real-time detection of epileptic seizures in animal models using reservoir computing. *Epilepsy Res.* **103**, 124–134 (2013).
22. Panda, P. & Srinivasa, N. Learning to recognize actions from limited training examples using a recurrent spiking neural model. *Front. Neurosci.* **12**, 126 (2018).
23. Appeltant, L. *et al.* Information processing using a single dynamical node as complex system. *Nat. Commun.* **2**, 468 (2011).
24. Ozturk, M. C., Xu, D. & Príncipe, J. C. Analysis and design of echo state networks. *Neural Comput.* **19**, 111–138 (2007).
25. Rodan, A. & Tino, P. Minimum complexity echo state network. *IEEE Trans. Neural Netw.* **22**, 131–144 (2011).
26. Strauss, T., Wustlich, W. & Labahn, R. Design strategies for weight matrices of echo state networks. *Neural Comput.* **24**, 3246–3276 (2012).
27. Katori, Y., Tamukoh, H. & Morie, T. Reservoir computing based on dynamics of pseudo-billiard system in hypercube. In *2019 International Joint Conference on Neural Networks (IJCNN)*, 1–8 (2019).
28. Alomar, M. L. *et al.* Digital implementation of a single dynamical node reservoir computer. *IEEE Trans. Circuits Syst. II Exp. Briefs* **62**, 977–981 (2015).
29. Loomis, L., McDonald, N. & Merkel, C. An FPGA implementation of a time delay reservoir using stochastic logic. *J. Emerg. Technol. Comput. Syst.* **14**, 46 (2018).
30. Alomar, M. L. *et al.* Efficient parallel implementation of reservoir computing systems. *Neural Comput. Appl.* **32**, 2299–2313 (2018).
31. Penkovsky, B., Larger, L. & Brunner, D. Efficient design of hardware-enabled reservoir computing in FPGAs. *J. Appl. Phys.* **124**, 162101 (2018).
32. Brunner, D. *et al.* Tutorial: Photonic neural networks in delay systems. *J. Appl. Phys.* **124**, 152004 (2018).
33. de Lima, T. F., Shastri, B. J., Tait, A. N., Nahmias, M. A. & Prucnal, P. R. Progress in neuromorphic photonics. *Nanophotonics* **6**, 577–599 (2017).
34. Peng, H., Nahmias, M. A., de Lima, T. F., Tait, A. N. & Shastri, B. J. Neuromorphic photonic integrated circuits. *IEEE J. Select. Top. Quantum Electron.* **24**, 1–15 (2018).
35. Torrejon, J. *et al.* Neuromorphic computing with nanoscale spintronic oscillators. *Nature* **547**, 428–431 (2017).
36. Dion, G., Mejaouri, S. & Sylvestre, J. Reservoir computing with a single delay-coupled non-linear mechanical oscillator. *J. Appl. Phys.* **124**, 152132 (2018).
37. Bauer, F. C., Muir, D. R. & Indiveri, G. Real-time ultra-low power ECG anomaly detection using an event-driven neuromorphic processor. *IEEE Trans. Biomed. Circuits Syst.* **13**, 1575–1582 (2019).
38. Yamaguchi, M., Katori, Y., Kamimura, D., Tamukoh, H. & Morie, T. A chaotic Boltzmann machine working as a reservoir and its analog VLSI implementation. In *2019 International Joint Conference on Neural Networks (IJCNN)*, 1–7 (2019).
39. Tanaka, G. *et al.* Recent advances in physical reservoir computing: A review. *Neural Netw.* **115**, 100–123 (2019).
40. Dambre, J., Verstraeten, D., Schrauwen, B. & Massar, S. Information processing capacity of dynamical systems. *Sci. Rep.* **2**, 514 (2012).
41. Jaeger, H. Short term memory in echo state networks. *Technical Report GMD Report 152, German National Research Center for Information Technology* (2002).
42. White, O. L., Lee, D. D. & Sompolinsky, H. Short-term memory in orthogonal neural networks. *Phys. Rev. Lett.* **92**, 148102 (2004).
43. Boedecker, J., Obst, O., Lizier, J. T., Mayer, N. M. & Asada, M. Information processing in echo state networks at the edge of chaos. *Theory Biosci.* **131**, 205–213 (2012).
44. Farkaš, I., Bosák, R. & Gerge, P. Computational analysis of memory capacity in echo state networks. *Neural Netw.* **83**, 109–120 (2016).
45. Ganguli, S., Huh, D. & Sompolinsky, H. Memory traces in dynamical systems. *Proc. Natl. Acad. Sci.* **105**, 18970–18975 (2008).
46. Yildiz, I. B., Jaeger, H. & Kiebel, S. J. Re-visiting the echo state property. *Neural Netw.* **35**, 1–9 (2012).
47. Snoek, J., Larochelle, H. & Adams, R. P. Practical Bayesian optimization of machine learning algorithms. *Adv. Neural Inf. Process. Syst.* **25**, 2951–2959 (2012).
48. Frazier, P. I. A tutorial on Bayesian optimization. arXiv:1807.02811 (2018).
49. Marquez, B. A., Suarez-Vargas, J. & Shastri, B. J. Takens-inspired neuromorphic processor: A downsizing tool for random recurrent neural networks via feature extraction. *Phys. Rev. Research* **1**, 033030 (2019).
50. Takens, F. Detecting strange attractors in turbulence. In *Dynamical Systems and Turbulence, Lecture Notes in Mathematics* (Springer, Berlin, 1981).
51. Richter, H. The generalized Hénon maps: Examples for higher-dimensional chaos. *Int. J. Bifurcation Chaos* **12**, 1371–1384 (2002).

Acknowledgements

The authors would like to thank Makoto Ikeda, Hiromitsu Awano, and Gouhei Tanaka for the fruitful discussion. This work was partially supported by the "Brain-Morphic AI to Resolve Social Issues" project at UTokyo, the NEC Corporation, and AMED (JP20dm0307009).

Author contributions

All the authors designed the research. Y.S. performed all simulations, and all the authors confirmed the theory. Further, they all wrote the paper.

Competing interests

Y.S., K.M., and K.A. have applied for a patent related to the proposed methods (Japanese Patent Application No. 2019-206438). The remaining author has no conflict of interest to declare.

Additional information

Correspondence and requests for materials should be addressed to Y.S.

Reprints and permissions information is available at www.nature.com/reprints.

Publisher's note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.