2023 International Symposium on Nonlinear Theory and Its Applications
NOLTA2023, September 26-29, 2023, Catania and Online

Reservoir computing with high-order polynomial activation functions and regenerative internal weights for enhancing nonlinear capacity and hardware resource efficiency

Yuki Abe[†], Kohei Nishida[††], Megumi Akai-Kasaya[‡,‡†], and Tetsuya Asai[‡]

†Graduate School of IST / ‡Faculty of IST , Hokkaido University
Kita 14, Nishi 9, Kita-ku, Sapporo, Hokkaido 060-0814, Japan
††Faculty of Engineering, Hokkaido University
Kita 13, Nishi 8, Kita-ku, Sapporo, Hokkaido 060-8628, Japan
‡†Department of Chemistry, Graduate School of Science, Osaka University
1-1 Machikaneyama, Toyonaka, Osaka 560-0043, Japan

Email: abe.yuki.cx@ist.hokudai.ac.jp

Abstract— This report describes a method to achieve resource efficiency and accurate prediction in hardware implementation of Reservoir Computing. We discuss the performance of the proposed method and architecture in terms of their impact on resource efficiency and various benchmark scores.

1. Introduction

Reservoir Computing (RC) is a field of research that connects nonlinear science and artificial intelligence (AI) [**?**]. While conventional AI approaches intelligence from the perspective of the brain structure, RC is a machine learning framework that leverages the in/out characteristics of nonlinear dynamical systems to process information. Due to its unique approach, various implementations have been proposed, including physical implementations [2, 3]. Regardless of the implementation target, RC has the ability to evaluate the information processing capacity of any system, making it a significant area of research in nonlinear science [4]. Furthermore, RC is expected to provide a solution for edge computing due to its low operating resource requirements [5]. This study proposes a method to reduce resource usage in hardware implementations of RC by utilizing regenerative internal weights, as well as a method to expand nonlinear capacity through high-order polynomials. Using the proposed architecture, we have confirmed that the hardware resources required for network weights can be reduced to less than 1% of those required by conventional methods. Additionally, our system has demonstrated superior performance, achieving 6-9th order nonlinear capacity, which is difficult to achieve with conventional methods.

ORCID iDs Yuki Abe: 0009-0001-8793-3203, Kohei Nishida: 0000-0001-6064-5917, Megumi Akai-Kasaya: 0000-0003-2217-9382, Tetsuya Asai: 0000-0003-1158-9810

2. Reservoir Computing

RC is a type of Recurrent Neural Network (RNN) model known for its lower computational resource requirements for learning compared to existing RNN models[1]. This model differs from existing RNN models in that the neural network is separated from the learning system. In this model, the fixed structure network is called the ”Reservoir”, and the separated learning system is called the ”Readout”, as shown in Figure 1. The Reservoir acts as

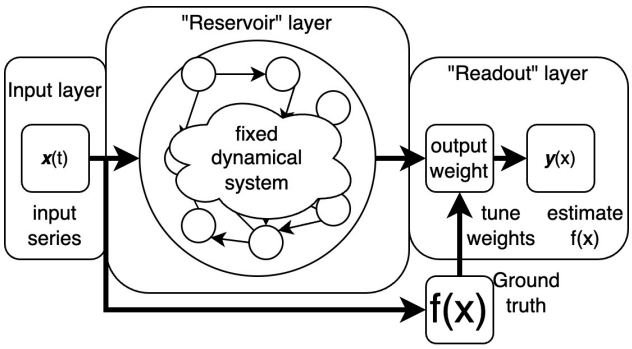

Figure 1: Basic RC model

a kernel for Support Vector Machines, making complex supervisors linearly separable. The Readout is the learning system, and most of them are implemented as linear regression. In this paper, we use the classical Echo State Network model as the basic RC architecture, which has a 1-dimensional input/output and no output feedback connection [6]. The mathematical RC model is described by the following formula:

$$\mathbf{x}(t) = f(\mathbf{W}_{net}\mathbf{x}(t-1) + \mathbf{W}_{in} \times u(t) + b) \quad (1)$$

$$z(t) = \mathbf{W}_{out} \cdot \mathbf{x}(t) \quad (2)$$

where $\mathbf{W}_{net}$ is a network weight (N,N) matrix, $\mathbf{W}_{in}$ is an input weight (N,1) vector, b is a bias term (fixed to 0.001 in this study), $\mathbf{W}_{out}$ is an output weight (N,1) vector, and N is

the size of the Reservoir. Moreover, $f(x)$ is the activation function, t represents the discrete time, and $\mathbf{X}(t)$, $z(t)$, and $u(t)$ are the state (N,1) vector of the Reservoir, the output scalar, and the input scalar, respectively, at discrete time t.

3. Proposed algorithms

3.1. Nonlinear capacity expansion through high-order polynomial functions

First, we introduce the activation function that enables application-specific capacity in the context of RC. The activation function of RC is usually a Tanh, which can be challenging to implement on hardware. Additionally, Tanh has a strong ratio of 1st-3rd order, indicating low-order capacity based on the Taylor expansion of the function, which is inconvenient when performing high-order capacity. Therefore, we propose an activation function with a "reconfigurable" nonlinearity. The function is defined as follows:

$$f(x) = \sum_{i=1}^{p}(a_{2i-1}x^{2i-1}) + sgn(-x)\sum_{i=1}^{p-1}(a_{2i}x^{2i}) \tag{3}$$

$$\mathbf{a} = (a_1, a_2 \dots a_{2p-1}) \tag{4}$$

$$g(x) = \begin{cases} f(x) & |f(x)| < 1 \\ sgn(x) & 1 \leqq |f(x)| \end{cases} \tag{5}$$

where $g(x)$ is the activation function, and $f(x)$ is the function before output limitation. In this context, p is a constant value that defines the upper limit of the power terms, $\mathbf{a}$ is a coefficient vector of size $(1, 2p-1)$, and a_i is the weight corresponding to each power term. In RC, the activation function is an essential factor that directly affects the dimension of the system's capacity. The activation function can realize the nonlinear capacity of the system according to the application by adjusting the weights that correspond to each power term.

3.2. Main algorithms

Next, we describe the calculation flow. The RC algorithm involves basic matrix operations and activation. The matrix operations consist of the accumulation of reservoir states multiplied by their corresponding weights. Our algorithm is a single-threaded design that processes these operations starting from the top row of the network weight matrix. The calculation flow is as follows:

1. Scan the network weight matrix from the top left to the top right.
2. Add the weighted inputs after scanning the rows.
3. Apply the activation function to the accumulated products to obtain the output of one node.
4. Finally, write the node's output to the RAM.

This calculation flow is repeated for as many nodes as there are. Once all calculations are completed, the previous state $\mathbf{X}(\tau-1)$, which is no longer needed, is overwritten, and the process moves on to the next discrete time.

4. Proposed architecture

4.1. Hardware demand reduction through periodic function

Now we introduce a method to reduce the memory usage required for the network weight matrix in RC. It is well known that the network weight matrix in RC exhibits the following characteristics:

- Each element of the matrix is generated from a uniform distribution of random numbers.
- The spectral radius of the matrix must be less than 1, but is expected to be close to 1[7].

In previous works, the network weight matrix was stored in memory (RAM), as is typical for RNN research. However, if the source of each element is a uniform distribution of random numbers, it should be equivalent to storing the weights in memory and integrating a random number generator into the system. Therefore, in this study, we propose a method to use a regenerative internal weight generator without storing the weights in memory. We employ a Linear Feedback Shift Register (LFSR) as the source of the regenerative internal weights. During the scanning of the network weight matrix, the LFSR is advanced by one step per element, and its output is decoded and used as the value of the weight. This weight generator is implemented as shown in Figure 2. Although this approach alone may

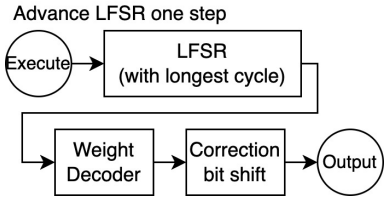

Figure 2: Weight generator

generate a network weight matrix with a spectral radius exceeding 1, potentially corrupting the reservoir, we propose a solution to avoid this problem. Before implementation, we derive the spectral radius of the generated matrix $\rho(\mathbf{W}_{net})$ and right-arithmetic bit shift the weights by $\lceil \log_2(\rho(\mathbf{W}_{net}))\rceil$ bits. This correction bit shift is applied to the weight generator. The same mechanism is used to generate weights for the input weight vectors, in whose case the dynamic range of the input weights is also adjusted by the correction bit shift. With this technique, the memory resources required for a network weight matrix with 100 nodes can be replaced by a register that is only 0.01% of that size.

4.2. Main architecture

The architecture was designed by implementing the algorithm and the hardware resource reduction method. The overall architecture is illustrated in Figure 3, where each colored area corresponds to a process in the calculation

flow. The blue area is designed for network calculation

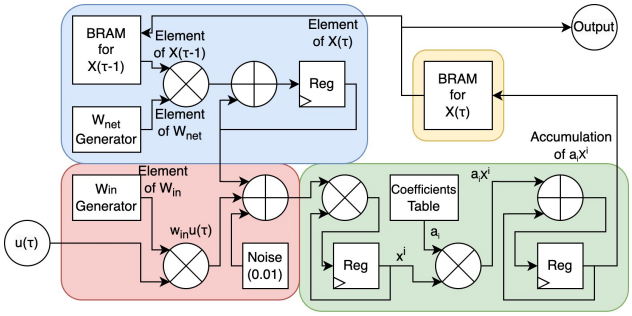

Figure 3: Overall architecture

(flow1), the red area is designed for input addition (flow2), the green area is designed for activation (flow3), and the yellow area is designed for saving the output. Each area operates when the corresponding process is executed. To generate the network and input weights, the weight generators described in the previous section were utilized. The activation function proposed in section 3.1 was implemented by storing the weights for each power term in a register, which is called the Coefficients Table.

5. Evaluation

Finally, we proceed to the evaluation of the proposed system. The Verilog HDL implementation of the proposed system was simulated using Icarus Verilog, and the output of the reservoir was obtained. The system's performance was then evaluated using linear regression. Two architectures with different activation functions were implemented and evaluated separately. The first activation function, denoted as Function (1), used $p = 5$ and $\mathbf{a} = (1, 1, 1, 1, 1, 1, 1, 1, 1, 1)$, whereas the second activation function, denoted as Function (2), used $p = 5$ and $\mathbf{a} = (1/256, 1/256, 1/64, 1/64, 1/16, 1/16, 1/4, 1/4, 1)$. The in/out characteristics of Functions (1) and (2) are shown in Figure 4, and their equations can be written as follows:

$$f(x) = \sum_{i=1}^{4}(x^{2i-1} + sgn(-x) \times x^{2i}) + x^9 \quad (6)$$

$$f(x) = \sum_{i=1}^{4}(\frac{x^{2i-1} + sgn(-x) \times x^{2i}}{4^{5-i}}) + x^9 \quad (7)$$

The reservoir parameters are as follows: the number of nodes is 400 and the spectral radius is 0.735. Regarding the input weight range, the values were adjusted based on the activation function, with the range (-1/128,1/128) used for Function(1), and (-1/16,1/16) used for Function(2). We initially evaluated the proposed system's performance using two standard benchmarks in RC, namely Memory Capacity (MC) and NARMA10-task [8, 9]. NARMA10-task's accuracy was measured using Normalized Root Mean Squared Deviation (NRMSD) and Normalized Root Mean

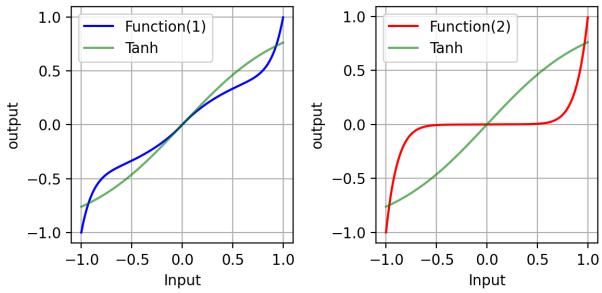

Figure 4: In/Out characteristic of the two activation functions

Squared Error (NRMSE). The benchmark results, along with the plot of NARMA10-task and the forgetting curve of MC, are illustrated in Figure 5. The MC results indicate that the architecture with Function (1) has a superior 1st-order capacity. This can be attributed to the fact that Function (1) places relatively more weight on the 1st-order term when compared to the other activation function. However, in terms of measuring nonlinear capacity, the NARMA10-task produced better results for the architecture with Function (1). We believe that the capacity may be purged to higher-order, which cannot be measured by the NARMA10-task. As a result, we ran a benchmark task, the Information Processing Capacity (IPC) task [10], which can classify and evaluate capacity in various dimensions. In addition, we included the results of the ESN (400 nodes, spectral radius 0.9, input weight range 0.1, and network density 5%) as a comparator in this task. The results are presented in Figure 6. Upon comparison of Figure 6 (a) and (b), it is apparent that the proposed activation function shifted the memory capacity, which was concentrated in the second and third order, to the fourth and fifth orders. Additionally, comparing Figure 6 (b) and (c), it is evident that instead of significantly reducing the low-order capacity, it augmented the capacity beyond the 6th order, which was difficult to achieve with previous models (note: the memory capacity after the 5th order in (b) was 0). These results suggest that it is possible to design an activation function that fits the application by adjusting the coefficients for power terms. Furthermore, we evaluated the system from the architecture viewpoint. After implementing the proposed system, it was synthesized using the FPGA development tool Quartus and evaluated based on the synthesis results. The evaluation results are summarized in Table 1. The number of ALMs utilized in our study is relatively

Table 1: Architecture compilation results

Total ALMs	659
Total DSP Blocks	12
Throughput per loop [1/s]	51.53 (in 50 MHz clk)

small compared to those in similar studies, which is pri-

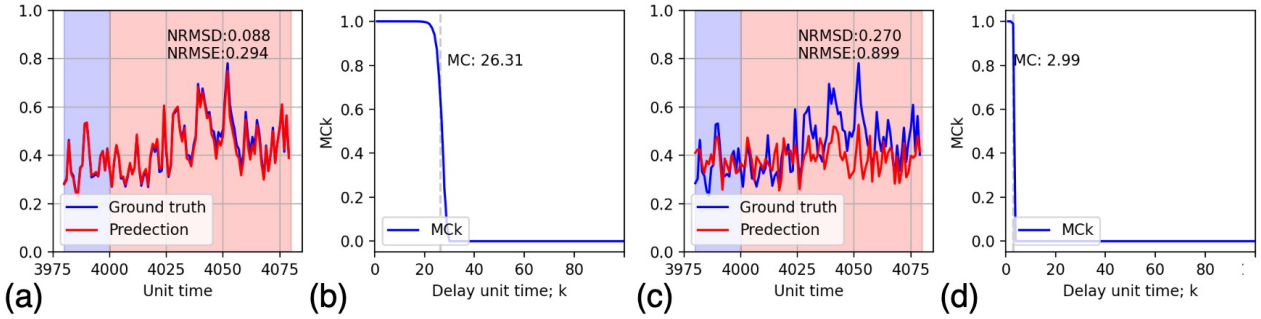

Figure 5: NARMA10-task and MC Results, (a,b) were obtained with Function(1) and (c,d) with Function(2).

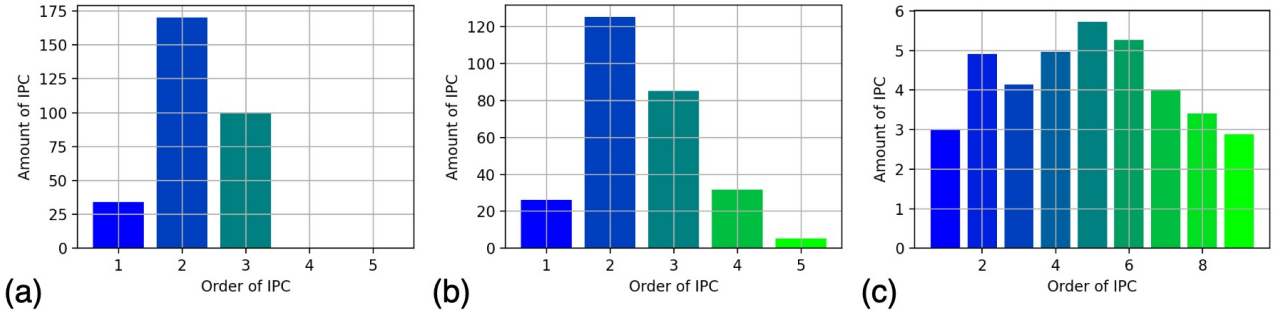

Figure 6: IPC task results obtained with (a) ESN(Comparator), (b) Function (1), and (c) Function (2).

marily due to our hardware resource reduction technique [11, 12].

6. Summary

In this study, we proposed two methods to reduce hardware resources: regenerative internal weights and a method to obtain nonlinear capacity according to the application. We demonstrated that each proposed method can be implemented with significantly fewer registers than conventional methods. Additionally, our proposed activation function was shown to achieve higher-order capacity, which is difficult to attain using conventional models. However, the throughput of the proposed architecture is slow because it is a single-threaded design. Therefore, in future work, we aim to develop a resource-efficient and real-time processable RC architecture by applying multi-threading or pipelining techniques.

Acknowledgments

The authors would like to sincerely thank Tokyo Electron Ltd. for their cooperation in this research.

References

[1] K. Nakajima and I. Fischer, “Reservoir Computing”, 2021, doi: 10.1007/978-981-13-1687-6.

[2] S. Kan, et al., Adv. Sci. 9, pp. 2104076, 2022, doi: 10.1002/advs.202104076.

[3] H. Kubota, et al., NOLTA, vol. E13-N, no. 2, pp. 373-378, 2022, doi: 10.1587/nolta.13.373.

[4] M. Inubushi, et al., Sci Rep, vol. 7, pp. 10199, 2017, doi: 10.1038/s41598-017-10257-6.

[5] G. Tanaka, et al., Neural Networks, vol. 115, pp. 100-123, 2019, doi: 10.1016/j.neunet.2019.03.005.

[6] M. Lukosevicius, et al., Computer Science Review, vol. 3, Issue 3, pp. 127-149, 2009, doi: 10.1016/j.cosrev.2009.03.005.

[7] Chris G. Langton, Physica D: Nonlinear Phenomena, vol. 42, Issues 1-3, pp. 12-37, 1990, doi: 10.1016/0167-2789(90)90064-V.

[8] A. F. Atiya, et al., IEEE TNN, vol. 11, no. 3, pp. 697-709, 2000, doi: 10.1109/72.846741.

[9] H. Jaeger, Short Term Memory in Echo State Networks, 2002.

[10] J. Dambre, et al., Sci Rep, vol. 2, pp. 514, 2012, doi: 10.1038/srep00514.

[11] M. L. Alomar et al., IEEE TCAS II, vol. 62, no. 10, pp. 977-981, 2015, doi: 10.1109/TCSII.2015.2458071.

[12] C. Lin, et al., 2022 23rd ISQED, pp. 1-6, 2022, doi: 10.1109/ISQED54688.2022.9806247.