Fast Parameter Optimization of Delayed Feedback Reservoir with Backpropagation and Gradient Descent

Sosei Ikeda
Graduate School of Informatics
Kyoto University
Kyoto, Japan
sikeda@easter.kuee.kyoto-u.ac.jp

Hiromitsu Awano
Graduate School of Informatics
Kyoto University
Kyoto, Japan
awano@i.kyoto-u.ac.jp

Takashi Sato
Graduate School of Informatics
Kyoto University
Kyoto, Japan
takashi@i.kyoto-u.ac.jp

***Abstract*—A delayed feedback reservoir (DFR) is a reservoir computing system well-suited for hardware implementations. However, achieving high accuracy in DFRs depends heavily on selecting appropriate hyperparameters. Conventionally, due to the presence of a non-linear circuit block in the DFR, the grid search has only been the preferred method, which is computationally intensive and time-consuming and thus performed offline. This paper presents a fast and accurate parameter optimization method for DFRs. To this end, we leverage the well-known backpropagation and gradient descent framework with the state-of-the-art DFR model for the first time to facilitate parameter optimization. We further propose a truncated backpropagation strategy applicable to the recursive dot-product reservoir representation to achieve the highest accuracy with reduced memory usage. With the proposed lightweight implementation, the computation time has been significantly reduced by up to 1/700 of the grid search.**

I. Introduction

Reservoir computing (RC) is a machine learning method closely associated with the neuromorphic concept [1]. It uses a reservoir, which nonlinearly transforms inputs into a high-dimensional space. The reservoir weights are fixed in RC; only the output layer weights are trained [2].

A delayed feedback reservoir (DFR) [3] is a specific type of RC system. It consists of a nonlinear element and a feedback loop. The nonlinear element often adopts a neuromorphic Mackey–Glass equation, which models cells in the blood [4]. The structural simplicity of DFR makes it an attractive candidate for embedded hardware implementation, with low power consumption being a notable advantage among RCs [1]. Various implementation methods for RC have been proposed, including electronic circuits [3], optical elements [5], etc.

The performance of RC heavily depends on the reservoir weights [1]. In the case of DFR, grid search has been commonly used to optimize the weights or parameters [3]. However, grid search faces a significant scalability problem. In the case of DFR, both the regularization parameter of the linear regression at the output layer and the reservoir parameters must be optimized as hyperparameters, resulting in a high–dimensional search space. In addition, this optimization process can be highly nonlinear, necessitating a finely divided grid, leading to a significant increase in computation time as the number of optimizing parameters grows. In the context of embedded applications, online learning is crucial as it aligns well with the low-power implementation that DFR enables. However, the challenges associated with training DFR hinder its realization in online learning scenarios. Fast learning methods are essential overcome these obstacles and achieve online learning in DFR.

Backpropagation and gradient descent [6] are widely used and proven effective training methods for neural networks. However, to the best of the authors' knowledge, no prior study has applied these methods to DFRs. One of the reasons for this absence is that DFRs use a nonlinear element often based on physical phenomena, making it challenging to backpropagate residues or requiring complex calculations [7].

To address this limitation, the authors propose a parameter optimization method for DFRs based on backpropagation and gradient descent, leveraging the concept of modular DFR [8]. The modular DFR model reduces the number of parameters in the reservoir layer and divides the nonlinear element into multiple blocks, facilitating the application of backpropagation. With this approach, the authors demonstrate that fast learning is achievable for DFRs. As only the regularization parameter of linear regression in the output layer is treated as a hyperparameter, and the recursively defined reservoir state of the DFR, which serves as a feature for the output layer, is approximated, the fixed number of epochs is required. The major contributions of this study can be summarized as follows:

1) Proposal of a method to optimize reservoir parameters using backpropagation and gradient descent based on the modular DFR model that allows for selecting a nonlinear function whose derivatives can be efficiently obtained while maintaining accuracy.
2) Simplification of the backpropagation calculation by reducing the number of reservoir features used retrospectively. This enhancement leads to faster computation and can reduce memory usage required for backpropagation.
3) Demonstration of the faster learning of the proposed

979-8-3503-4859-0/DATE24/© 2024 EDAA

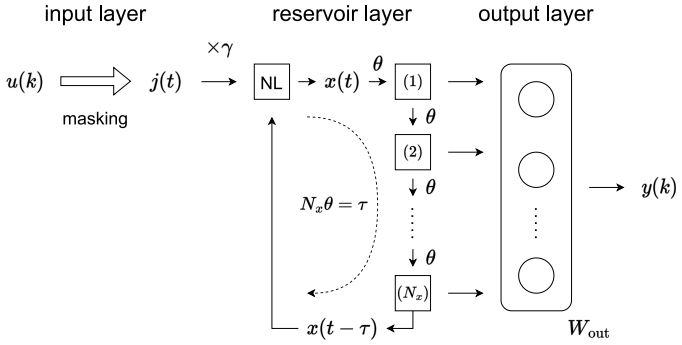

Fig. 1. Conceptual diagram of DFR. The reservoir consists of a nonlinear element (NL) and a feedback loop with a total delay τ. The feedback loop comprises N_x virtual nodes with a time interval θ.

method using multiple classification tasks. The method achieves comparable accuracy to traditional grid search with magnitude, up to 700x, reduced computation time.

II. DFR

A. Concept of DFR

The DFR is a specific implementation of RC, utilizing a nonlinear element combined with a delayed feedback loop. RC has a structure that reflects past input, making it suitable for processing time series [1]. This approach proves to be more straightforward for hardware implementation than other RC schemes due to its simplistic structure [1]. Typically, hardware realizations of DFRs employ analog circuits in the reservoir layer [3], [9].

We explain the standard analog implementation of DFR to demonstrate the operation and the challenges in applying efficient learning methods. Fig. 1 illustrates its conceptual diagram. Initially, the digital time series input, $u(k)$, sampled at a period τ, is converted to an analog signal, $i(t)$. This signal, $i(t)$, undergoes masking wherein it is multiplied by the mask signal with a faster sample rate θ. The resulting signal $j(t)$ is subsequently scaled by γ, added to the reservoir feedback, and fed into the nonlinear element (NL) to generate the output, $x(t)$. The NL is mathematically represented by the following delay differential equation:

$$\frac{\mathrm{d}}{\mathrm{d}t}x(t) = F(x(t-\tau), \gamma j(t)). \tag{1}$$

The Mackey–Glass (MG) model [4] is a commonly used NL in various studies [3], [9], [10]. Its operation is expressed by the following equations:

$$\frac{\mathrm{d}}{\mathrm{d}t}x(t) = -x(t) + \eta f(x(t-\tau), \gamma j(t)), \tag{2}$$

$$f(x(t-\tau), \gamma j(t)) = \frac{[x(t-\tau) + \gamma j(t)]}{1 + [x(t-\tau) + \gamma j(t)]^p}. \tag{3}$$

Here, p is an adjustable parameter. In this context, the virtual nodes are connected at a time interval θ on the feedback loop with a total delay of τ. The signal values at these virtual nodes serve as the features and are collected as a vector of N_x elements, referred to as the reservoir state, where $N_x\theta = \tau$.

The reservoir state is defined as follows:

$$\boldsymbol{x}(k) \equiv [x(k\tau - \theta), x(k\tau - 2\theta), \ldots, x(k\tau - \tau)]. \tag{4}$$

The output is subsequently obtained by applying a linear transformation.

Fully digital versions of DFRs have been proposed to enhance the ease of implementation [3], [10]. In these digital DFRs, the signal, $\boldsymbol{j}(k)$ after the masking process is expressed as $\boldsymbol{j}(k) = \boldsymbol{m}u(k)$ where $\boldsymbol{m}$ is a mask vector of length N_x and its elements are determined randomly and digitally [3].

After masking, the nonlinear element and feedback loop of the DFR determine the reservoir state, $\boldsymbol{x}(k)$. Assuming that f in Eq. (2) is constant for a short time θ, which is the time interval between the virtual nodes, the differential equation in Eq. (2) is solved for $x(t)$ as follows:

$$x(t) = x_0 \mathrm{e}^{-t} + \eta(1 - \mathrm{e}^{-t}) f(x(t-\tau) + \gamma j(t)), \tag{5}$$

where x_0 is the initial value of $x(t)$ at each θ [3]. The value of the next virtual node can be expressed as $x(\theta)$. Assuming that the components of $\boldsymbol{x}(k)$ and $\boldsymbol{j}(k)$ are

$$\boldsymbol{x}(k) \equiv [x(k)_1, x(k)_2, \ldots, x(k)_{N_x}], \tag{6}$$

$$\boldsymbol{j}(k) \equiv [j(k)_1, j(k)_2, \ldots, j(k)_{N_x}], \tag{7}$$

$\boldsymbol{x}(k)$ is obtained recurrently as follows:

$$x(k)_n = x(k)_{n-1}\mathrm{e}^{-\theta} + (1 - \mathrm{e}^{-\theta}) f(x(k-1)_n, j(k)_n). \tag{8}$$

Here, the initial value of the reservoir state is set to 0. The construction and operation of the output layer are the same as in an analog implementation.

In either implementation, the recursive nature of the network construction and the differentially defined NL make grid search the only effective method for parameter optimization.

B. DPRR

In classification tasks, a single output is expected for multiple time-series inputs. The number of features obtained by the reservoir layer depends on the length of the input series, necessitating the features converted into an intermediate representation of fixed length called the *reservoir representation* [11]. This conversion allows the features to be processed by an output layer of a fixed size [12].

Numerous reservoir representations have been proposed [3], [12]–[15]. Currently, DPRR is considered to be the best in terms of both accuracy and circuit size [11]. Let the elements of $\boldsymbol{x}(k)$ (i.e., the features) be $[x(k)_1, x(k)_2, \ldots, x(k)_{N_x}]$. The vector of a node in the reservoir:

$$\boldsymbol{x}_n \equiv [x(1)_n, x(2)_n, \ldots, x(T)_n], \tag{9}$$

can be considered to be the time evolution of the node state. Here, T denotes the length of a time series. Taking the dot product of these vectors for two nodes yields:

$$\boldsymbol{x}_i \cdot \boldsymbol{x}_j^- = \sum_{k=1}^{T} x(k)_i x(k-1)_j \ (i, j = 1, 2, \ldots, N_x). \tag{10}$$

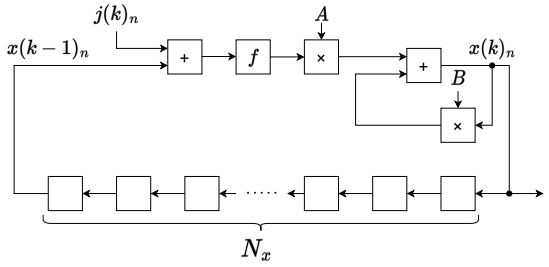

Fig. 2. Block diagram of reservoir processing in the modular DFR model. The block labeled "f" operates as a one-input, one-output function f in Eq. (2). Only two parameters, A and B, have to be optimized.

In addition, the reservoir state itself is used as a feature:

$$\boldsymbol{x}_i \cdot \mathbf{1} = \sum_{k=1}^{T} x(k)_i \ (i = 1, 2, \ldots, N_x). \tag{11}$$

The DPRR is defined as a vector $\boldsymbol{r}$ by concatenating Eqs. (10) and (11), which consists of $N_x \cdot (N_x + 1)$ values, i.e., $\boldsymbol{r} \equiv \mathrm{vec}\left(\sum_{i=1}^{T} \boldsymbol{x}(k)\boldsymbol{x'}(k-1)\right)$, where $\boldsymbol{x'}(k-1) = [\boldsymbol{x}(k), 1]$. The output is obtained using the output layer W_{out}, where N_r represents the number of features in a reservoir representation:

$$\boldsymbol{y} = W_{\mathrm{out}}\boldsymbol{r} + \boldsymbol{b} = \sum_{n=1}^{N_r} \boldsymbol{w}_n \boldsymbol{r}_n + \boldsymbol{b}. \tag{12}$$

For each of the components of r, multiple reservoir state features are used in the derivation, making backpropagation complicated and challenging.

C. Modular DFR

The challenge in designing a DFR lies in determining the design of the NL. In analog implementations utilize elements obey a delay differential equation. In digital implementations, the goal is often to replicate the behavior of elements used in existing analog implementations.

The modular DFR presents a model that simplifies the design of the NL in the digital implementation of DFR [8]. First, the NL operation is decomposed into multiple blocks, simplifying the design target from a delay differential equation to a one-input, one-output function f. This reduction also decreases the number of parameters to be adjusted from 3 to 2 while preserving the same solution space. Fig. 2 illustrates the block diagram of the modular DFR model obtained. Here, $x(k)_n$ can be represented as follows:

$$x(k)_n = Af(j(k)_n + x(k-1)_n) + Bx(k)_{n-1}. \tag{13}$$

In Eq. (13), the parameters subject to optimization are A and B. In total, three parameters need to be optimized: these two and the regularization parameter for linear regression in the output layer. However, using grid search for optimization is challenging, even after the number of parameters is reduced, owning to the need to explore the three-dimensional parameter space. Moreover, improper selection of these three parameters could lead to insufficient accuracy. Therefore, an efficient optimization strategy is essential to ensure effective parameter tuning.

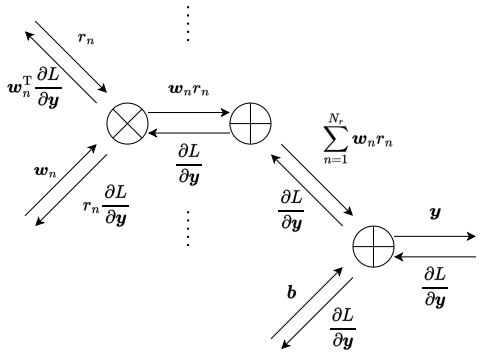

Fig. 3. Computation graph of forward and backward propagation in the output layer.

III. Parameter Optimization of DFR by Backpropagation and Gradient Descent

In this section, we present the formulation of backpropagation for DFR. We choose the modular DFR for its network structure and the task-specific extensibility of the NL. The backpropagation process is explained in reverse order, starting from the output layer, then the DPRR layer, and finally the reservoir layer, tracing back from the error in the output. Special attention is given to the DPRR layer, where backpropagation occurs from multiple roots for a single value, and the reservoir layer with a recursive structure. Subsequently, we propose a truncation approach to the backpropagation to enable a more lightweight implementation.

A. Backpropagation in the output layer

Fig. 3 illustrates the computation graph summarizing the backpropagation in the output layer. First, the output $\boldsymbol{y}$ and the target output $\boldsymbol{d}$ are represented as the following vectors:

$$\boldsymbol{y} = [y_1, y_2, \ldots, y_{N_y}] \text{ and } \boldsymbol{d} = [d_1, d_2, \ldots, d_{N_y}]. \tag{14}$$

Here, N_y represents the number of classes, and $\boldsymbol{d}$ is represented in a one-hot encoding. The loss function is formulated using cross entropy error as:

$$L = -\sum_{k=1}^{N_y} d_k \log y_k. \tag{15}$$

Then the partial derivative of L with respect to $\boldsymbol{y}$ is given by:

$$\frac{\partial L}{\partial \boldsymbol{y}} = \boldsymbol{y} - \boldsymbol{d}. \tag{16}$$

This derivative is propagated backwards through the network.

The computation in the output layer is expressed as Eq. (12). The partial derivatives of L with respect to $\boldsymbol{b}$, $\boldsymbol{r}_n$, and $\boldsymbol{w}_n$ are respectively expressed as:

$$\frac{\partial L}{\partial \boldsymbol{b}} = \frac{\partial L}{\partial \boldsymbol{y}}, \ \frac{\partial L}{\partial \boldsymbol{r}_n} = \boldsymbol{w}_n^{\mathrm{T}}\frac{\partial L}{\partial \boldsymbol{y}}, \text{ and } \frac{\partial L}{\partial \boldsymbol{w}_n} = \boldsymbol{r}_n\frac{\partial L}{\partial \boldsymbol{y}}. \tag{17}$$

B. Backpropagation in the DPRR layer

Fig. 4 illustrates the computation graph summarizing the backpropagation in the DPRR layer. First, each feature in the

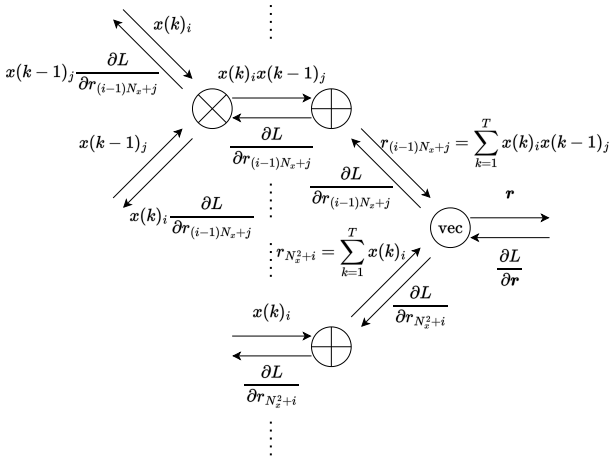

Fig. 4. Computation graph of forward and backward propagation in the DPRR layer.

DPRR is defined as follows.

$$r_{(i-1)N_x+j} = \sum_{k=1}^{T} x(k)_i x(k-1)_j \ (i,j = 1,\ldots,N_x), \quad (18)$$

$$r_{N_x^2+i} = \sum_{k=1}^{T} x(k)_i \ (i = 1,\ldots,N_x). \quad (19)$$

For $r_{(i-1)N_x+j}$, the following is backpropagated.

$$\frac{\partial L}{\partial x(k)_i} = x(k-1)_j \frac{\partial L}{\partial r_{(i-1)N_x+j}}, \quad (20)$$

$$\frac{\partial L}{\partial x(k-1)_j} = x(k)_i \frac{\partial L}{\partial r_{(i-1)N_x+j}}. \quad (21)$$

Subsequently, for $r_{N_x^2+i}$, the following is backpropagated.

$$\frac{\partial L}{\partial x(k)_i} = \frac{\partial L}{\partial r_{N_x^2+i}}. \quad (22)$$

From the above, the following value (bpv) is backpropagated from the DPRR layer for the partial derivative of L with respect to $x(k)_n$.

$$\begin{aligned}(\text{bpv}) = &\sum_{j=1}^{N_x} x(k-1)_j \frac{\partial L}{\partial r_{(n-1)N_x+j}} \\ &+ \sum_{i=1}^{N_x} x(k+1)_i \frac{\partial L}{\partial r_{(i-1)N_x+n}} + \frac{\partial L}{\partial r_{N_x^2+n}}. \end{aligned} \quad (23)$$

C. Backpropagation in the reservoir layer

Fig. 5 illustrates the computation graph summarizing the backpropagation in the reservoir layer. Note that f has a constant multiplication parameter A. First, the partial derivative of L with respect to $Bx(k)_{n-1}$ is

$$\frac{\partial L}{\partial (Bx(k)_{n-1})} = \frac{\partial L}{\partial x(k)_n}. \quad (24)$$

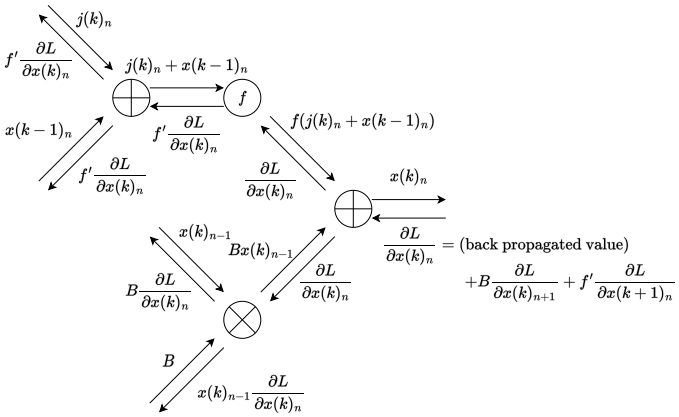

Fig. 5. Computation graph of forward and backward propagation in the reservoir layer.

It is then expressed as follows.

$$\frac{\partial L}{\partial x(k)_{n-1}} = B\frac{\partial L}{\partial x(k)_n}, \quad (25)$$

$$\frac{\partial L}{\partial B} = x(k)_{n-1}\frac{\partial L}{\partial x(k)_n}. \quad (26)$$

Next, the partial derivative of L with respect to $f(j(k)_n + x(k-1)_n)$ is

$$\frac{\partial L}{\partial f(j(k)_n + x(k-1)_n)} = \frac{\partial L}{\partial x(k)_n}. \quad (27)$$

The partial derivative of L with respect to the constant multiplication parameter A of f is

$$\frac{\partial L}{\partial A} = \frac{\partial f(j(k)_n + x(k-1)_n)}{\partial A}\frac{\partial L}{\partial x(k)_n}. \quad (28)$$

Then, the following holds.

$$\begin{aligned}\frac{\partial L}{\partial x(k-1)_n} &= \frac{\partial L}{\partial (j(k)_n + x(k-1)_n)} \\ &= \frac{\partial f(j(k)_n + x(k-1)_n)}{\partial (j(k)_n + x(k-1)_n)}\frac{\partial L}{\partial x(k)_n}. \end{aligned} \quad (29)$$

The partial derivative of L with respect to $x(k)_n$ is recursively given as:

$$\frac{\partial L}{\partial x(k)_n} = (\text{bpv}) + B\frac{\partial L}{\partial x(k)_{n+1}} + f'\frac{\partial L}{\partial x(k+1)_n}. \quad (30)$$

Therefore, the partial derivative of L with respect to the constant multiplication parameter A of f and B are respectively given by adding up for all past times as follows:

$$\frac{\partial L}{\partial A} = \sum_{k=1}^{T} \frac{\partial f(j(k)_n + x(k-1)_n)}{\partial A}\frac{\partial L}{\partial x(k)_n}, \quad (31)$$

$$\frac{\partial L}{\partial B} = \sum_{k=1}^{T} x(k)_{n-1}\frac{\partial L}{\partial x(k)_n}. \quad (32)$$

D. Truncated Backpropagation

For (bpv) in Eq. (23), the value of $\frac{\partial L}{\partial \boldsymbol{r}}$ is determined at time T, so the reservoir state must be stored for the number of times used for backpropagation plus one time. Therefore, to use the entire time series, $(T+1)$ reservoir states must be stored, which consumes quadratically larger memory space as T becomes larger. To alleviate the memory usage while maintaining optimization accuracy, we propose a *truncated backpropagation* method where only a limited number of reservoir states, e.g., the last time only, are used. This approximation is based on the observation that the last reservoir state cumulatively reflects past reservoir states, and the impact of past states gradually attenuates. With this approximation, Eqs. (23),(30),(31), and (32) are respectively simplified as follows:

$$(\text{bp value}) = \sum_{j=1}^{N_x} x(T-1)_j \frac{\partial L}{\partial r_{(n-1)N_x+j}} + \frac{\partial L}{\partial r_{N_x^2+n}}, \quad (33)$$

$$\frac{\partial L}{\partial x(T)_n} = (\text{bp value}) + B\frac{\partial L}{\partial x(T)_{n+1}}, \quad (34)$$

$$\frac{\partial L}{\partial A} = \frac{\partial f(j(T)_n + x(T-1)_n)}{\partial A}\frac{\partial L}{\partial x(T)_n}, \quad (35)$$

$$\frac{\partial L}{\partial B} = x(T)_{n-1}\frac{\partial L}{\partial x(T)_n}. \quad (36)$$

By comparing Eqs. (31),(32) and Eqs. (35),(36), it becomes evident that the computational effort is significantly reduced by about $1/T$. This simplification allows us to store only two reservoir states, i.e., $\boldsymbol{x}(T-1)$ and $\boldsymbol{x}(T)$. Specifically, for many datasets with a time series length T greater than 100, the memory requirement for the reservoir state can be decreased to less than 2%. Considering the overall memory usage, which includes output weights and other factors, let us consider a scenario that a DFR solves a three-class classification task with a time series length of $T = 500$ and $N_x = 30$, the reduction in memory usage would be approximately 80%.

IV. Evaluation

In this section, we evaluate the proposed parameter optimization method for DFR using the same datasets (npz files) as in [15]. Throughout the evaluation, the optimization of the reservoir function f in modular DFR was not performed; $f(x) = Ax$ was used consistently for all datasets, as suggested in [8]. Also, the reservoir size, N_x, was set to 30. The evaluation was performed in the following environment: AMD Ryzen 7 5825U CPU with 16.0 GB of memory. All programs were written using Python 3.9.6 and the numpy library version 1.23.0.

In the proposed method, the reservoir parameters, A and B, and the output parameters, W_{out} and b, are optimized by backpropagation. The initial value of $[A, B]$ is $[0.01, 0.01]$, and the output parameters are initialized to zeros. The stochastic gradient descent method was run over 25 epochs. The learning rate starts at 1 and is multiplied by 0.1 for the parameters of the reservoir layer at epochs 5, 10, 15, and 20. For the parameters of the output layer, the learning rate is multiplied by 0.1 at epochs 10, 15, and 20. Once the optimization with backpropagation is completed, the output layer undergoes training using ridge regression with the regularization parameter β. For β, we try $[10^{-6}, 10^{-4}, 10^{-2}, 10^{0}]$ and choose the one with the smallest loss L.

TABLE I
Runtime comparison between the proposed backpropagation (bp) and grid search (gs). Column "gs divs" represents the number of grid divisions required to achive equal accuracy as the proposed method (bp acc).

data set	bp acc	bp time (s)	gs divs	gs time (s)	(gs time)/ (bp time)
ARAB	0.981	245	8	25,040	102.2
AUS	0.954	54	8	5,535	102.5
CHAR	0.918	44	10	4,820	109.5
CMU	0.931	4	1	3	0.8
ECG	0.850	11	16	4,977	452.5
JPVOW	0.978	4	4	106	26.5
KICK	0.800	7	1	2	0.3
LIB	0.806	12	18	8,423	701.9
NET	0.783	45	1	49	1.1
UWAV	0.850	65	10	6,322	97.3
WAF	0.983	14	3	188	13.4
WALK	1.000	4	1	3	0.8

A. Comparison with grid search

First, we compared the proposed method with grid search, executed in three dimensions, A, B, and β. The ranges for A and B were set to $[10^{-3.75}, 10^{-0.25}]$, $[10^{-2.75}, 10^{-0.25}]$, respectively. These ranges were determined to be able to find the optimal parameters for all the datasets used in this evaluation. In these ranges, the number of grid divisions is increased from 1 until the accuracy reaches that of the proposed method. Note that the grid divisions are performed equally; e.g., if the number is 2, the range is divided into two equal sections. β is searched in the same way as the proposed method.

The comparison results for accuracy and computation time are summarized in Table I. Remarkably, the accuracy achieved by the parameters optimized with backpropagation is equal to or even superior to the state-of-the-art study [11] for most datasets. The proposed method significantly reduces the computation time to approximately 1/700th of the grid search. This reduction is particularly significant for datasets where the grid search requires long computation times. For datasets requiring fewer grid divisions, the parameter space yielding good results is larger, or the initial coarse grid produces satisfactory outcomes coincidentally. However, early stopping of grid search is practically challenging, as it is difficult to know whether optimal accuracy has been attained. Conversely, datasets with a small area of optimal parameters require numerous grid divisions for accurate optimization. Grid search, in the worst case, demands exponential computation time. An alternative method to explore a finer parameter space is to recursively dig the best region in the coarser search, which results in linear time complexity. However, this approach may fail for certain datasets if the coarse grid search does not

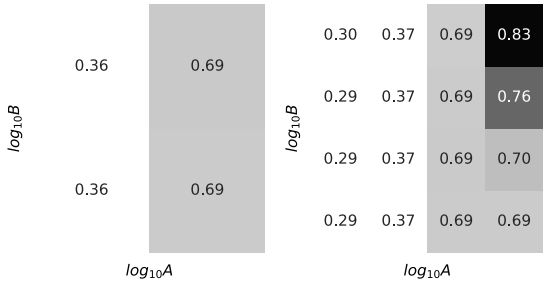

Fig. 6. Grid search example where the recursively narrowing the finer grids is challenging. The dataset used is CHAR. Left: grid level 1, right: grid level 2.

TABLE II
STORAGE REDUCTION BY TRUNCATED BACKPROPAGATION. "NAIVE" AND "SIMPLIFIED" REPRESENT THE TOTAL NUMBER OF STORED VALUES OF THE RESERVOIR STATE, RESERVOIR REPRESENTATION, AND RESERVOIR WEIGHT, BEFORE AND AFTER TRUNCATING THE COMPUTATION OF BACKPROPAGATION.

dataset	naive (a)	simplified (b)	(a-b)/a
ARAB	13030	10300	21 %
AUS	93455	89435	4 %
CHAR	25700	19610	24 %
CMU	20192	2852	86 %
ECG	7352	2852	61 %
JPVOW	10179	9369	8 %
KICK	28022	2852	90 %
LIB	16245	14955	8 %
NET	42853	13093	69 %
UWAV	17828	8438	53 %
WAF	8732	2852	67 %
WALK	60332	2852	95 %

yield globally optimal parameters, as illustrated in Fig. 6. Conversely, the proposed method successfully found optimal values with a fixed number of epochs for all datasets used in this study. Although further improvements can be explored by attempting different initial values or increasing the number of epochs, these did not improve the accuracy of the presented datasets.

B. Memory usage reduction by truncated backpropagation

Table II lists the memory savings resulting from the truncation of the backpropagation calculations shown in Section III.III-D. While the reduction in memory is relatively smaller for datasets with a large number of classes and short series lengths, more than half of the data sets still achieve a reduction larger than 50%. In addition, for all datasets, including those with small memory reductions, the computational effort of backpropagation has been reduced to about $1/T$, resulting in a considerable reduction in computation time.

V. CONCLUSION

We proposed a fast and accurate parameter optimization method for DFR utilizing backpropagation and gradient descent. Backpropagation in the reservoir layer, which was particularly challenging and computationally intensive, was addressed by introducing the modular DFR model. Using the model, we formulated the backpropagation of DPRR, which effectively improves accuracy but poses difficulties owing to long dependence on past reservoir states. Additionally, we presented a truncated backpropagation method that takes advantage of the reservoir state's characteristics. As a result, the proposed method significantly reduced the computation time by up to about 1/700. Moreover, the truncation technique successfully reduces overall memory usage by half for most datasets.

ACKNOWLEDGEMENTS

This work was supported in part by the Japan Society for the Promotion of Science KAKENHI under Grant 23H03362.

REFERENCES

[1] G. Tanaka, T. Yamane, J. B. Héroux, R. Nakane, N. Kanazawa, S. Takeda, H. Numata, D. Nakano, and A. Hirose, "Recent advances in physical reservoir computing: A review," *Neural Netw.,* 115, pp. 100-–123, 2019.

[2] M. Lukoševičius and H. Jaeger, "Reservoir computing approaches to recurrent neural network training," *Computer Science Review,* 3(3), pp. 127-–149, 2009.

[3] L. Appeltant, M. C. Soriano, G. Van der Sande, J. Danckaert, S. Massar, J. Dambre, B. Schrauwen, C. R. Mirasso, and I. Fischer, "Information processing using a single dynamical node as complex system," *Nat. Commun.,* 2(1), pp. 1-–6, 2011.

[4] M. C. Mackey and L. Glass, "Oscillation and chaos in physiological control systems," *Science,* 197(4300), pp. 287—289, 1977.

[5] L. Larger, M. C. Soriano, D. Brunner, L. Appeltant, J. M. Gutiérrez, L. Pesquera, C. R. Mirasso, and I. Fischer, "Photonic information processing beyond Turing: an optoelectronic implementation of reservoir computing," *Optics Express,* vol. 20, no. 3, pp. 3241—3249, 2012

[6] S. Amari, "Backpropagation and stochastic gradient descent method," *Neurocomputing,* vol. 5, no. 4–5, pp. 185-–196, 1993.

[7] M. Nakajima, K. Inoue, K. Tanaka, Y. Kuniyoshi, T. Hashimoto, and K. Nakajima, "Physical deep learning with biologically inspired training method: gradient-free approach for physical hardware," *Nature Commun.,* 13(1), p. 7847, 2022.

[8] S. Ikeda, H. Awano, and T. Sato, "Modular dfr: digital delayed feedback reservoir model for enhancing design flexibility," *ACM Trans. Embed. Comput. Syst.,* 22(5s), 2023.

[9] M. C. Soriano, S. Ortín, L. Keuninckx, L. Appeltant, J. Danckaert, L. Pesquera, and G. Van der Sande, "Delay-based reservoir computing: noise effects in a combined analog and digital implementation," *IEEE Trans. Neural Netw. Learn. Syst.,* 26(2), pp. 388-–393, 2014.

[10] M. L. Alomar, M. C. Soriano, M. Escalona-Morán, V. Canals, I. Fischer, C. R. Mirasso, and J. L. Rosselló, "Digital implementation of a single dynamical node reservoir computer," *IEEE Trans. Circuits Syst. II: Express Br.,* 62(10), pp. 977-–981, 2015.

[11] S. Ikeda, H. Awano, and T. Sato, "Hardware-friendly delayed-feedback reservoir for multivariate time-series classification," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.,* 41(11), pp. 3650-–3660, 2022.

[12] H. Chen, F. Tang, P. Tino, and X. Yao, "Model-based kernel for efficient time series analysis," *Proc. ACM SIGKDD Int. Conf.,* pp. 392—400, 2013.

[13] R. Pascanu, J. W. Stokes, H. Sanossian, M. Marinescu, and A. Thomas, "Malware classification with recurrent networks," *Proc. IEEE Int. Conf. Acoust. Speech Signal Process.,* pp. 1916-–1920, 2015.

[14] J. Cabessa, H. Hernault, H. Kim, Y. Lamonato, and Y. Z. Levy, "Efficient text classification with echo state networks," *Proc. Int. Jt. Conf. Neural Netw.,* pp. 1-–8, 2021.

[15] F. M. Bianchi, S. Scardapane, S. Løkse, and R. Jenssen, "Reservoir computing approaches for representation and classification of multivariate time series," *IEEE Trans. Neural Netw. Learn. Syst.,* pp. 2169-–2179, 2020.