FastESN: Fast Echo State Network

Hai Wang, *Member, IEEE,* Xingyi Long, and Xue-Xin Liu

***Abstract*—Echo state networks (ESNs) are reservoir computing based recurrent neural networks widely used in pattern analysis and machine intelligence applications. In order to achieve high accuracy with large model capacity, ESNs usually contain a large sized internal layer (reservoir), making the evaluation process too slow for some applications. In this work, we speed up the evaluation of ESN by building a reduced network called the fast echo state network (fastESN), and achieve an ESN evaluation complexity independent of the original ESN size for the first time. FastESN is generated using three techniques. First, the high dimensional state of the original ESN is approximated by a low dimensional state through proper orthogonal decomposition (POD) based projection. Second, the activation function evaluation number is reduced through the discrete empirical interpolation method (DEIM). Third, we show the directly generated fastESN has instability problems and provide a stabilization scheme as a solution. Through experiments on four popular benchmarks, we show that fastESN is able to accelerate the sparse storage based ESN evaluation with a high parameter compression ratio and a fast evaluation speed.**

***Index Terms*—Echo state networks, reservoir computing, recurrent neural networks, complexity reduction, acceleration.**

I. Introduction

The artificial neural network (ANN) has been widely used as a powerful tool in many pattern recognition and machine learning applications. Particularly, deep network, which is ANN with multiple neuron layers, demonstrates superior performances in many computer vision and speech recognition applications [1]. It is even able to handle some complex tasks such as the abstract strategy board game playing [2], real-time strategy video game playing [3], and the very large scale integration (VLSI) circuit design and optimization [4]–[8].

Belonging to the recurrent neural network (RNN) which is the deep network specialized in dealing with temporal data [1], echo state network (ESN) is an important reservoir computing based neural network [9] and is closely related to the liquid state network [10]. ESN is known for its simple structure with a large sized internal recurrent topology called the reservoir. Furthermore, unlike the normal RNNs whose weights are all trainable, only the output weights of an ESN are trained easily using the linear regression method [11]. Thanks to the simple structure and the easy linear regression based training process, ESNs have gained popularity in many time series and nonlinear dynamical system modeling applications [7], [12], [13].

After training, a neural network will be used in an application: the network will take the input data and produce the output data. This is called the evaluation process, or forward propagation since the information flows from the back (input side) to the front (output side). The evaluation speed of an ESN is a major concern because many ESN based applications require quick service response time and/or are deployed on resource constrained devices such as mobile phones, tablets, wearable devices, etc. To make matters worse, ESNs usually need a large reservoir (large internal unit number) in order to capture the complex features in the training data [14]. Such a large reservoir will further slow down the computation in evaluation.

Many research studies have been conducted to speed up the forward propagation of deep networks, most of which focus on the convolutional neural networks (CNNs) by using methods like parameter pruning [15]–[17] and low-rank approximation [18], [19], as briefly surveyed in Section II. However, to the best of our knowledge, there are very few acceleration methods for the evaluation of ESNs, with some principal component analysis (PCA) based methods proposed in [20], [21]. Unfortunately, these methods are only designed to reduce the number of inputs [20] or to reduce the number of state signals transmitted to the special Volterra filter structured output layer [21]. Since the internal units/states (reservoir) are left untouched, they do not work for the general ESNs as discussed in Section II.

In this work, we propose a new compact network called fast echo state network (fastESN) to accelerate the ESN evaluation with three complexity reduction techniques:

1) The high dimensional state of the ESN is approximated by a low dimensional state through the proper orthogonal decomposition (POD) based projection method, which was introduced to analyze turbulent flows [22] and was widely used to reduce dynamical systems [23]. This state approximation accelerates the evaluation of ESN simply because a smaller state will be evaluated in the forward propagation.
2) The activation operation number (i.e., the number of operations by the activation function $\tanh$) of ESN is reduced by using the discrete empirical interpolation method (DEIM) [24]. This technique further increases the evaluation speed, because fewer nonlinear function evaluations and related computations are needed. Thanks to the two techniques above, the evaluation complexity of the fastESN is much lower than that of the ESN and is independent of the ESN's internal unit number.
3) We show theoretically and experimentally that the reduced network directly generated by DEIM, although much faster to evaluate than the original ESN, may be unstable. To solve this problem, a stabilization technique with the idea originally introduced in [25] is utilized, which stabilizes the fastESN to make its evaluation not

This research is supported by National Natural Science Foundation of China under grant No. 61974018.

H. Wang and X. Long are with School of Electronic Science and Engineering, University of Electronic Science and Technology of China, Chengdu, 610054 China.

X.-X. Liu is with Pure Storage, Inc., Mountain View, CA 94041 USA.

only fast but also stable.

The main contribution of this work and the advantages of fastESN are

1) By introducing fastESN, we achieve an ESN evaluation complexity of $\mathcal{O}(q^2+q\cdot n_{\rm in}+q\cdot n_{\rm out})$ for the first time, where q is any user specified number independent of the original ESN size, $n_{\rm in}$ and $n_{\rm out}$ are the input and output numbers, respectively.
2) We demonstrate how to reduce the ESN state size through a state approximation projection framework and also how to reduce the function dimension via a DEIM procedure with stability preservation technique.
3) FastESN greatly reduces the parameter number of the ESN to save storage. It has fewer parameters even compared with the sparsely connected ESN because its weight matrices only have a dimension of $q\times q$.
4) FastESN has a structure similar to the ESN and inherits the stability from the original ESN. As a result, it behaves just like the ESN, but is smaller.
5) FastESN can be generated by collecting samples using the original ESN. Therefore, it can be applied in a wide range of scenarios, even without the training data.

The rest of this article is organized as follows. In Section II, we discuss the related work on the acceleration of neural network evaluation. The preliminaries of ESN are given in Section III, including the structure as well as the basic training and evaluation procedures. Then, we present fastESN in Section IV, covering its structure, its generation and stabilization techniques, the flow of ESN evaluation acceleration using fastESN, and the complexity analysis of fastESN evaluation, etc. Next, the experimental results on four well known benchmarks are given in Section V. The limitations of fastESN and the future work are discussed in Section VI. Finally, Section VII concludes this article.

II. Related work

The research studies on the evaluation acceleration of neural networks mainly focus on CNNs. The acceleration methods can be mainly classified into pruning based and low-rank approximation based. For the pruning based methods, Han *et al.* proposed to prune the weights with small magnitude to reduce the computational complexity of CNN evaluation [15]. The coarse pruning methods like filter pruning and channel pruning were proposed in [17], [26], [27]. The accuracy-sparsity relationship with different granularity for pruning was studied in [28]. Recently, a layer-wise pruning method [16] was introduced by reducing the redundancies within the learned features. For the low-rank approximation based methods, Jaderberg *et al.* proposed to approximate the original filter by a set of rank-1 filters [18]. Zhang *et al.* introduced an acceleration method for very deep CNNs by considering the nonlinear effects in the low-rank approximation [19].

On the ESN side, since its introduction by Jaeger [9], there have been many new techniques proposed to improve the ESNs, including the least square based training [11], the leaky integrator units [29], the reservoir adaptation method by intrinsic plasticity [30], [31], the stacked recurrent layers [32], and an automatic design scheme using multiple subreservoirs [33]. To achieve faster evaluation speed, there are also methods built upon sparse and deterministic reservoir structures, such as the minimum complexity reservoir [14] and the cycle reservoir with jumps (CRJ) [34].

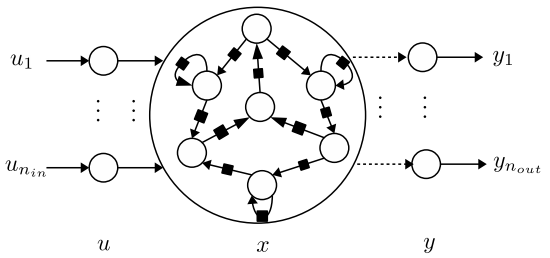


Fig. 1: The structure of an echo state network with $n_{\rm in}$ inputs and $n_{\rm out}$ outputs. It has three layers: the input layer (u), the hidden/internal layer (x) with the hidden units as the neurons in the middle circle, and the output layer (y). The black solid square represents a delay of one time step.

Accelerating the evaluation of ESNs remains a challenging problem, with few related research studies using principal component decomposition (PCA). In [35], PCA is applied to speed up the training process of ESNs, but with the evaluation complexity untouched. Bianchi *et al.* proposed an acceleration method that reduces the number of inputs using PCA for the ESN used for electric load forecasting [20]. This method is designed specifically for applications with a large number of highly correlated inputs. It cannot reduce the high computing cost caused by the large reservoir and has little acceleration effects for the applications with a single or moderate number of inputs. A PCA based ESN method was proposed in [21], where PCA is used to reduce the number of state signals transmitted to the output layer which has a Volterra filter structure. This method is effective because the number of Volterra filter coefficients grows dramatically with the number of signals transmitted to the output [21]. However, for the general ESNs without the Volterra filter structure, such PCA based method will introduce extra storage and computing overhead, making the evaluation even slower.

In summary, there lacks an ESN evaluation acceleration method that reduces the hidden unit number to achieve a computational complexity irrelevant to the original ESN size. In this work, we develop fastESN to solve this problem: fastESN is the first ESN evaluation method with a computational complexity of $\mathcal{O}(q^2+q\cdot n_{\rm in}+q\cdot n_{\rm out})$, where q is any user specified number irrelevant to the original ESN size, $n_{\rm in}$ and $n_{\rm out}$ are the input and output numbers, respectively.

III. Basics of the ESN

In this section, we briefly present the basics of the ESN which will be used in this article. For more details of the ESN, please refer to [9], [29], [36].

A. The structure of ESN

The ESNs are special RNNs, which use the recurrent structure to model the sequence correlation in time. The

architecture of an ESN is given in Fig. 1. It is composed of three layers: the input layer (u), the hidden/internal layer (x), and the output layer (y). The recurrent relations (the time delays) are represented by the black solid squares in the figure, which exist in the internal layer.

An ESN with n_{in} inputs, n_{out} outputs, and n internal units, can be represented by the following nonlinear difference equations in state-space form at a discrete time k:

$$\begin{aligned} x(k+1) &= f(Wx(k) + W_{\text{in}}u(k)), \\ y(k+1) &= W_{\text{out}}x(k+1), \end{aligned} \tag{1}$$

where

- $x(k) \in \mathbb{R}^n$ is the state vector containing the values of the n internal units at time k;
- $u(k) \in \mathbb{R}^{n_{\text{in}}}$ and $y(k) \in \mathbb{R}^{n_{\text{out}}}$ are the input and output vectors at time k;
- $W_{\text{in}} \in \mathbb{R}^{n \times n_{\text{in}}}$ is the input weight matrix for the connections between input units and internal units;
- $W \in \mathbb{R}^{n \times n}$ is the internal weight matrix for the connections from time k to $k+1$ within the internal units;
- $W_{\text{out}} \in \mathbb{R}^{n_{\text{out}} \times n}$ is the output weight matrix for the connections between the internal units and the output units;
- $f(x)$ is called the activation function, which is usually a nonlinear sigmoid function. In this article, we use $f(x) = \tanh(x)$ by default since it is the most popular activation function for ESN.

To simplify presentation, we also call the internal unit number n as the "order" of the ESN.

B. Training and evaluation of ESN

The internal weights in W and input weights in W_{in} of ESN are untrainable: they are randomly generated weights of the reservoir. During the training process, only the output weights in W_{out} are trained through linear regression. It is noteworthy that the training of W_{out} does not need back-propagation through time (BPTT), and hence the problem of learning long-term dependencies in the normal RNNs, which is caused by the BPTT training [37], is avoided [38].

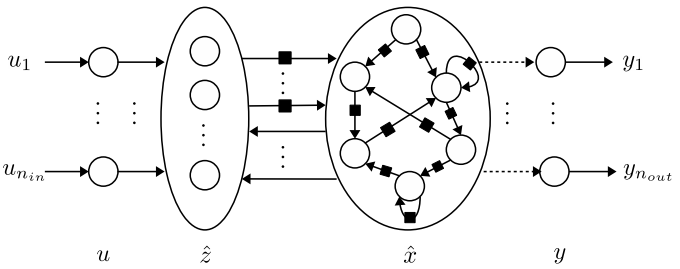
The evaluation (forward propagation) of ESN is straightforward. For each time step k, ESN takes the current input and internal state to compute the new state and output of time step $k+1$ using (1). This procedure will continue iteratively through all the time steps. The evaluation complexities at each time step are

$$\mathcal{O}(n^2 + n \cdot n_{\text{in}} + n \cdot n_{\text{out}}) \tag{2}$$

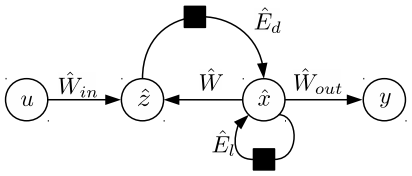
for dense matrix storage of W and

$$\mathcal{O}(nnz + n \cdot n_{\text{in}} + n \cdot n_{\text{out}}) \tag{3}$$

for sparse matrix storage of W [39] (used in the experiments of this work) with nnz as the number of nonzero elements in W, both are governed by the system size. This could make the evaluation too slow for some applications when n and nnz are large. In the next section, we will propose to generate a small network called fastESN, which has an order q with $q \ll n$. Then we can accelerate the ESN evaluation by evaluating fastESN instead, using the same input sequence.



(a) The structure of a fastESN. It contains one more internal/hidden layer ($\hat{z}$) compared with the ESN shown in Fig. 1.



(b) The simplified diagram of a fastESN.

Fig. 2: The structure of a fastESN. The black solid square means a delay of one time step.

IV. Fast echo state network (fastESN)

In this section, we present the fastESN which is generated to speed up the evaluation of ESN. First, we show the structure of fastESN in Section IV-A. Subsequently, we present the three techniques that generate fastESN from the original ESN in the following order: the state approximation technique which reduces the state dimension of the original ESN is shown in Section IV-B, the activation function approximation technique which further lowers the network evaluation complexity is given in Section IV-C, the stabilization technique which stabilizes the reduced network and leads to the final form of fastESN is presented in Section IV-D. Next, we analyze the evaluation complexity of fastESN in Section IV-E and present the extension for the leaky ESN in Section IV-F. Finally, the flow of accelerating ESN evaluation using fastESN is summarized in Section IV-G.

A. The structure of fastESN

First of all, we present the structure of the fastESN to serve as an overview of the new method.

Assume we have built and trained an ESN expressed in (1) with the structure given in Fig. 1. The structure of a fastESN generated from the ESN is shown in Fig. 2. We see that fastESN has a similar structure compared with the ESN. It just contains one extra layer of internal units ($\hat{z}$) and is smaller (with q units in each internal layer) than the ESN (with n units in the internal layer).

The state-space formulation of the fastESN is written as:

$$\begin{aligned} \hat{x}(k+1) &= \hat{E}_l\hat{x}(k) + \hat{E}_d f(\hat{W}\hat{x}(k) + \hat{W}_{\text{in}}u(k)), \\ y(k+1) &= \hat{W}_{\text{out}}\hat{x}(k+1), \end{aligned} \tag{4}$$

where $\hat{x}(k) \in \mathbb{R}^q$, $\hat{W}_{\text{in}} \in \mathbb{R}^{q \times n_{\text{in}}}$, $\hat{W} \in \mathbb{R}^{q \times q}$, $\hat{W}_{\text{out}} \in \mathbb{R}^{n_{\text{out}} \times q}$, $\hat{E}_l \in \mathbb{R}^{q \times q}$, and $\hat{E}_d \in \mathbb{R}^{q \times q}$, with $q \ll n$.

Equation (4) also indicates that fastESN has two internal layers, with q units in each layer. To see this more clearly, we can rewrite (4) into the following form by introducing an extra state vector $\hat{z}$ as

$$\begin{aligned}\hat{z}(k) &= f(\hat{W}\hat{x}(k) + \hat{W}_{\text{in}}u(k)),\\ \hat{x}(k+1) &= \hat{E}_l\hat{x}(k) + \hat{E}_d\hat{z}(k),\\ y(k+1) &= \hat{W}_{\text{out}}\hat{x}(k+1),\end{aligned} \tag{5}$$

where $\hat{z}(k) \in \mathbb{R}^q$ is the state vector representing the extra internal layer with q units.

In the subsequent subsections, we will present the techniques that generate the fastESN.

B. State approximation using projection

This subsection demonstrates the state approximation technique to generate a smaller network with order $q \ll n$ for evaluation (forward propagation), so that the complexity ((2) or (3)) is significantly reduced and evaluation is much faster.

1) The framework of state approximation: The following state approximation

$$V_x\hat{x}(k) \approx x(k), \tag{6}$$

where $V_x \in \mathbb{R}^{n\times q}$ is a constant matrix (called *projection matrix*)[1] and $q \ll n$, allows us to approximate the original n-dim $x(k)$ of the ESN in (1) using a q-dim $\hat{x}(k)$ of fastESN.

The state approximation above leads to a residual R if we plug (6) in (1) as

$$R(k) = f(WV_x\hat{x}(k) + W_{\text{in}}u(k)) - V_x\hat{x}(k+1). \tag{7}$$

Then, by multiplying V_x^{T} to the residual $R(k)$ and forcing the result to be zero as

$$V_x^{\text{T}} f(WV_x\hat{x}(k) + W_{\text{in}}u(k)) - V_x^{\text{T}}V_x\hat{x}(k+1) = 0, \tag{8}$$

we end up with a new state-space model as[2]

$$\begin{aligned}\hat{x}(k+1) &= V_x^{\text{T}} f(WV_x\hat{x}(k) + W_{\text{in}}u(k)),\\ y(k+1) &= W_{\text{out}}V_x\hat{x}(k+1).\end{aligned} \tag{9}$$

The procedure above is known as the Bubnov–Galerkin projection.

The model/network expressed in (9) has the state $\hat{x}(k)$ with order q, which approximates the original n order state $x(k)$ through equation (6). As a result, we call this new network as the *state approximation ESN* in this article. The state approximation ESN is a semi-complete form of fastESN (as in (4)), with only state approximation.

2) Forming the projection matrix via proper orthogonal decomposition: There are many ways to form the projection matrix V_x that leads to the approximation (6). For fastESN, we form the projection matrix V_x using the time domain proper orthogonal decomposition (POD), because the POD samples can be easily obtained: if the training samples (for the ESN training) are available, part of them can be directly used as the POD samples; otherwise, the POD samples can be simply generated by forward propagation of the trained ESN with sample inputs.

POD has been introduced to analyze turbulent flows [22] and widely used to reduce both the linear and nonlinear dynamical systems [23] in a variety of applications including fast simulation of the VLSI systems [40] and runtime thermal estimation of the multi-core integrated systems [41].

The basic idea of POD is to take samples of the original dynamical system state, either in the time domain or in the frequency domain, and form the projection matrix using the principle components of these samples. For fastESN, we will take the original ESN state samples in the time domain as the POD samples.

Specifically, assume we take the original system state samples at n_s different time points (i.e., take $x(k)$ with n_s different k) as $\{x_1, x_2, \ldots, x_{n_s}\}$.

Then, apply singular value decomposition (SVD) to the POD samples as

$$V_x\Sigma_x U_x^{\text{T}} \xleftarrow{\text{SVD}} X, \tag{10}$$

where $X = \left[x_1, x_2, \ldots, x_{n_s}\right] \in \mathbb{R}^{n\times n_s}$, $V_x \in \mathbb{R}^{n\times n_s}$, $\Sigma_x \in \mathbb{R}^{n_s\times n_s}$, and $U_x \in \mathbb{R}^{n_s\times n_s}$.

Finally, to eliminate the redundancies in the POD samples, we truncate V_x matrix by keeping only its first q columns as

$$V_x \leftarrow V_x[:, 1:q], \tag{11}$$

where we used the MATLAB-like notation for matrix truncation. Since the first q columns of the original V_x correspond to the q largest singular values in Σ_x, the most important information in the samples is kept after truncation. Also, notice that V_x matrix is truncated from a unitary matrix according to the property of SVD, so there is $V_x^{\text{T}}V_x = I$.

The truncated V_x will be used as the projection matrix in the state approximation presented in Section IV-B1.

3) Analysis of the state approximation: So far, we have presented the framework of the state approximation and the construction of the projection matrix. It is natural to ask why the projection matrix V_x leads to a good state approximation. In other words, why does the state approximation ESN expressed in (9) have similar outputs compared with the original ESN in (1) (assuming both are driven by the same input sequence)?

To answer the questions above, we will look for the standard projection which connects $V_x\hat{x}(k)$ with $x(k)$ using a projector, then the state approximation can be explained using the standard projection.

First, let us assume the state approximation ESN (9) is fully accurate at time $k-1$, i.e., there is $V_x\hat{x}(k-1) = x(k-1)$.[3]

With the assumption above, from (9) and (1), we have

$$\begin{aligned}\hat{x}(k) &= V_x^{\text{T}} f(WV_x\hat{x}(k-1) + W_{\text{in}}u(k-1))\\ &= V_x^{\text{T}} f(Wx(k-1) + W_{\text{in}}u(k-1))\\ &= V_x^{\text{T}}x(k).\end{aligned} \tag{12}$$

[1]How to form the projection matrix to satisfy the approximation will be presented later in Section IV-B2.

[2]Here $V_x^{\text{T}}V_x$ is an identity matrix as will be shown later in Section IV-B2.

[3]Please note that the only purpose we make this assumption is to show why does the state approximation lead to the similar outputs compared with the original, because we need to isolate the errors from the previous steps to see the error generated only at the current step. In real applications, this assumption does not hold.

By introducing a new variable $\tilde{x}(k) = V_x\hat{x}(k)$ for convenience of presentation, we have

$$\tilde{x}(k) = V_x\hat{x}(k) = V_xV_x^{\mathrm{T}}x(k). \tag{13}$$

Clearly, $V_xV_x^{\mathrm{T}} \in \mathbb{R}^{n\times n}$ is an orthogonal projector (because there is $V_xV_x^{\mathrm{T}} = V_xV_x^{\mathrm{T}} \times V_xV_x^{\mathrm{T}}$), which projects $x(k)$ onto the column space of V_x along the orthogonal direction.

If $x(k)$ lies inside the column space of V, i.e., $x(k) \in \mathrm{colspan}(V_x)$, then there is

$$V_xV_x^{\mathrm{T}}x(k) = x(k), \tag{14}$$

which leads to the exact "approximation":

$$\tilde{x}(k) = V_x\hat{x}(k) = x(k). \tag{15}$$

Since the column space of V_x is spanned by the principle components of the time domain samples $\{x_1, x_2, \ldots, x_{n_s}\}$, $x(k)$ lies close to the column space of V_x if it is close to a sample x_i with $i = 1, 2, \ldots, n_s$, leading to a good state approximation in (6). This concludes our analysis of the state approximation.

4) Computational complexity problem of the state approximation ESN: In the forward propagation of the state approximation ESN (9), computing $WV_x\hat{x}(k-1)$ requires around $n\cdot q$ operations, since WV_x is an $n\times q$ matrix. In addition, the activation function f needs to be evaluated n times and computing $V_x^{\mathrm{T}}f(\cdot)$ needs $n\cdot q$ operations. What is more, although the operation number of the output side $W_{\mathrm{out}}V_x\hat{x}(k+1)$ has been reduced to $q\cdot n_{\mathrm{out}}$, the input side $W_{\mathrm{in}}u(k)$ still requires $n\cdot n_{\mathrm{in}}$ operations. As a result, the evaluation complexity of the state approximation ESN is

$$\mathcal{O}(n\cdot q + n\cdot n_{\mathrm{in}} + q\cdot n_{\mathrm{out}}). \tag{16}$$

Even though this is already lower than that of the original ESN ((2) and (3)), it still involves the original ESN internal unit number n. This problem must be solved to deliver impressive speedup.

Next, we will present the second fastESN construction technique, called activation function approximation, which further reduces the complexity by replacing the remaining n with q in (16).

C. Activation function approximation

By analyzing the computational complexity problem of the state approximation ESN, we find the major cause of the problem is that the activation function f has to operate on the n-dim vector $WV_x\hat{x}(k) + W_{\mathrm{in}}u(k)$ because $V_x^{\mathrm{T}} \in \mathbb{R}^{q\times n}$ is multiplied to the results of $f(\cdot)$.

An intuitive idea to solve this problem is to move V_x^{T} inside the activation function f, forming a $q\times q$ matrix like $V_x^{\mathrm{T}}WV_x$ to reduce the computational complexity. However, this straightforward idea does not work since the activation function f is usually nonlinear in the artificial neural networks, meaning $V_x^{\mathrm{T}}f(WV_x\hat{x}(k) + W_{\mathrm{in}}u(k)) \neq f(V_x^{\mathrm{T}}WV_x\hat{x}(k) + V_x^{\mathrm{T}}W_{\mathrm{in}}u(k))$.

Luckily, if we can find a selection matrix $P_g \in \mathbb{R}^{n\times q}$ whose columns are drawn from the $n\times n$ identity matrix, then there is $P_g^{\mathrm{T}}f(WV_x\hat{x}(k)+W_{\mathrm{in}}u(k)) = f(P_g^{\mathrm{T}}WV_x\hat{x}(k)+P_g^{\mathrm{T}}W_{\mathrm{in}}u(k))$. This is because the activation function f is applied to each element of a vector independently, so it does not matter whether we select the vector element using a selection matrix P_g^{T} before or after the activation operation. This indicates a possibility that we can form a network with an evaluation time complexity independent of n, since the new weight matrix $P_g^{\mathrm{T}}WV_x$ is only $q\times q$. This idea can be realized by the activation function approximation via the discrete empirical interpolation method (DEIM) [24] (a discrete version of the empirical interpolation method [42]) as presented in the following.

1) The framework of activation function approximation: We will first present the framework of activation function approximation. For convenience, we also write the activation function as $g(x(k)) = f(Wx(k) + W_{\mathrm{in}}u(k))$.

The essence of the activation function approximation is to find a q-dim function $\hat{g}(x)$ to approximate the original n-dim function $g(x)$, in order to reduce the activation function evaluation from n times to q times. This function approximation is expressed as:

$$V_g\hat{g}(x) \approx g(x), \tag{17}$$

where $V_g \in \mathbb{R}^{n\times q}$ and $\hat{g}(x) \in \mathbb{R}^q$.

By multiplying P_g^{T} on the residual of the approximation in (17) and making the result to be zero, we have

$$P_g^{\mathrm{T}}V_g\hat{g}(x) = P_g^{\mathrm{T}}g(x), \tag{18}$$

where $P_g \in \mathbb{R}^{n\times q}$ is a selection matrix as introduced previously. Since multiplying P_g^{T} to an n-dim vector is to select q elements (from the n elements of the vector) according to the positions of the q ones in P_g, the physical meaning of (18) is to have the exact "approximation" at the q equations determined by the selection matrix P_g.

From (18), $\hat{g}(x)$ can be uniquely determined as

$$\hat{g}(x) = (P_g^{\mathrm{T}}V_g)^{-1}P_g^{\mathrm{T}}g(x). \tag{19}$$

By combining (9), (17), and (19), we readily have

$$\begin{aligned}\hat{x}(k+1) &= V_x^{\mathrm{T}}V_g(P_g^{\mathrm{T}}V_g)^{-1}\\ &\quad\cdot f(P_g^{\mathrm{T}}WV_x\hat{x}(k) + P_g^{\mathrm{T}}W_{\mathrm{in}}u(k)),\\ y(k+1) &= W_{\mathrm{out}}V_x\hat{x}(k+1),\end{aligned} \tag{20}$$

whose computing complexity of forward propagation is independent of n now. Please note that we have used the property $P_g^{\mathrm{T}}f(WV_x\hat{x}(k)+W_{\mathrm{in}}u(k)) = f(P_g^{\mathrm{T}}WV_x\hat{x}(k)+P_g^{\mathrm{T}}W_{\mathrm{in}}u(k))$ since P_g is a selection matrix.

2) Forming the activation function approximation matrices: As presented previously, we need to build two matrices, P_g and V_g for the activation function approximation.

We look for the V_g matrix first. Similar to the theory of the state approximation presented in Section IV-B3, it is the column space of V_g that determines the activation function approximation accuracy in (17). As a result, in order to get a good approximation, we would like the subspace spanned by the columns of V_g to cover the main information of $g(x(k))$. This is still achieved through a POD process.

In the POD process of the activation function approximation, we take samples of $g(x(k))$ at n_s different

time points (i.e., take $g(x(k))$ with n_s different k) as $\{g_1, g_2, \ldots, g_{n_s}\}$. Then, apply SVD to the sample matrix $G = \left[g_1, g_2, \ldots, g_{n_s}\right] \in \mathbb{R}^{n \times n_s}$ as

$$V_g \Sigma_g U_g^{\mathrm{T}} \xleftarrow{\mathrm{SVD}} G, \tag{21}$$

where $V_g \in \mathbb{R}^{n \times n_s}$, $\Sigma_g \in \mathbb{R}^{n_s \times n_s}$, and $U_g \in \mathbb{R}^{n_s \times n_s}$. The final projection matrix in (17) is obtained by truncating V_g to preserve its first q columns as

$$V_g \leftarrow V_g[:, 1:q]. \tag{22}$$

Now we are ready to construct the selection matrix P_g. Let us denote $P_g = [e_{s_1}, e_{s_2}, \ldots, e_{s_q}] \in \mathbb{R}^{n \times q}$, where e_{s_i} is the s_i-th column of the $n \times n$ identity matrix. Then, the task of constructing P_g can be simplified to determining the index sequence $S = \{s_1, s_2, \ldots, s_q\}$. The basic idea is to find the indices $s_1, s_2, \ldots, s_q$ one by one, such that the growth of the error bound of the approximation (17) is limited in a greedy manner, where the error bound is expressed as

$$\|g(x) - V_g \hat{g}(x)\|_2 \leq \|(P_g^{\mathrm{T}} V_g)^{-1}\|_2 \cdot \|(I - V_g V_g^{\mathrm{T}}) g(x)\|_2. \tag{23}$$

Such procedure is described in Algorithm 1 of [24], which will not be presented here repeatedly.

D. Stabilization of fastESN

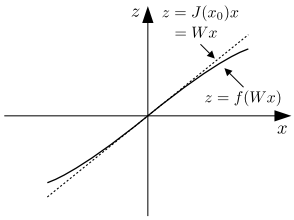
By applying the state approximation and activation function approximation, we have successfully generated a network in (20) which is much smaller in size compared with the original ESN and can be evaluated with a complexity independent of the large original order n. Unfortunately, this small network may not be asymptotically stable even when the original ESN is. In this subsection, we will analyze the instability problem, and provide a simple stabilization solution which leads to the final form of fastESN.

1) The instability problem: The stability of a nonlinear dynamical system at an equilibrium x_0 is determined by its Jacobian at the equilibrium $J(x_0)$: the system is asymptotically stable at x_0 if there is $\|J(x_0)\|_2 < 1$. Since the Jacobian of the original ESN at the equilibrium $x_0 = 0$ is W for $f(x) = \tanh(x)$, the weight matrix is scaled to satisfy $\|W\|_2 < 1$ in order to make the ESN stable [9] as demonstrated graphically in Fig. 3a.

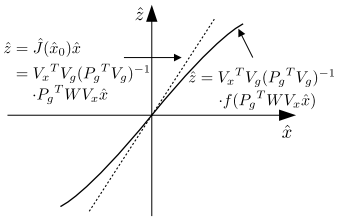
Unfortunately, the activation function approximation through DEIM presented in Section IV-C does not preserve the asymptotic stability property of the original ESN. For the reduced system expressed in (20), its stability at the equilibrium $\hat{x}_0 = 0$ is determined by its Jacobian

$$\hat{J}(\hat{x}_0) = \hat{J}(0) = V_x^{\mathrm{T}} V_g (P_g^{\mathrm{T}} V_g)^{-1} P_g^{\mathrm{T}} W V_x. \tag{24}$$

Please note that when the original ESN is stable ($\|W\|_2 < 1$), it is still possible to have $\|\hat{J}(\hat{x}_0)\|_2 > 1$, because there is $\|(P_g^{\mathrm{T}} V_g)^{-1}\|_2 > 1$. It means that using the DEIM procedure to approximate the activation function magnifies the Jacobian at the equilibrium, which may cause asymptotic instability of the reduced network. This problem will become even more severe as the reduced order q grows. This is because $\|(P_g^{\mathrm{T}} V_g)^{-1}\|_2$ (which is also a component of the error bound given in (23)) grows with the order q, leading to an increased possibility of $\|\hat{J}(\hat{x}_0)\|_2 > 1$. Please see Fig. 3b for a graphical view of this problem.

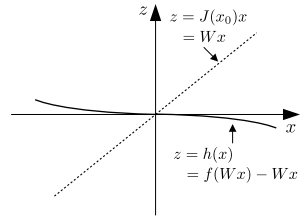


(a) The original ESN is asymptotically stable at the equilibrium $x_0 = 0$ as long as $\|J(x_0)\|_2 = \|W\|_2 < 1$.

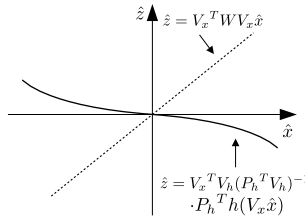


(b) The 2-norm of the Jacobian is magnified to $\|\hat{J}(\hat{x}_0)\|_2 > 1$ by $(P_g^{\mathrm{T}} V_g)^{-1}$ during the activation function approximation, causing the instability problem.

Fig. 3: Illustration of the instability problem of fastESN.



(a) Generate new nonlinear part $h(x)$ by subtracting Wx from $f(Wx)$. $h(x)$ has zero Jacobian at the equilibrium. The linear part Wx inherits the Jacobian from the original nonlinear part of ESN in Fig. 3a.



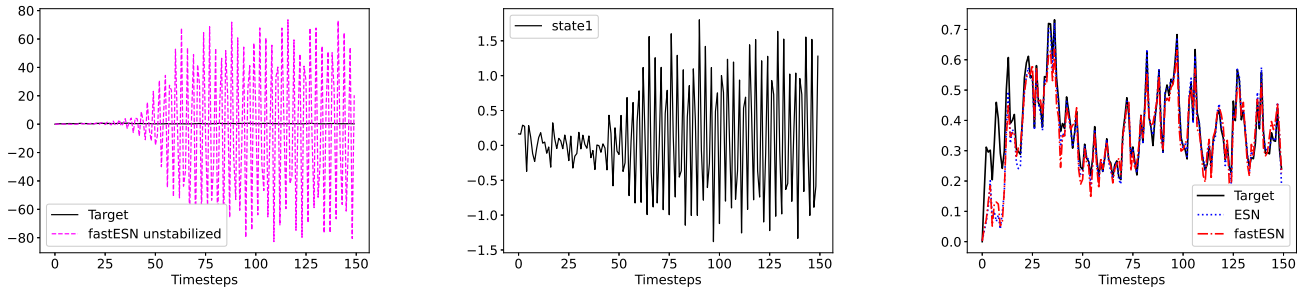
(b) Performing function approximation to $h(x)$ does not magnify the 2-norm of its Jacobian at the equilibrium, since it is zero. In addition, the 2-norm of linear term Jacobian remains the same: $\|V_x^{\mathrm{T}} W V_x\|_2 = \|W\|_2 < 1$.

Fig. 4: Illustration of the stabilization of fastESN.

Experimentally, we also give one example of the instability problem in Fig. 5a and Fig. 5b. This example is from a 50-order network generated using the activation function approximation through DEIM (called "fastESN unstabilized") from a 200-order ESN. Both the training and testing data are the 10th order NARMA data (NARMA10). The experimental settings follow the ones presented later in Section V-A, except for the dropping of the washout data. From Fig. 5a, we can see that the reduced network is not asymptotically stable, generating an output which oscillates around the target with a large amplitude. To analyze the problem, the value of the reduced network's first state (the first element of the state vector $\hat{x}(k)$) is plotted in Fig. 5b. It is observed that the state diverges after around 50 time steps, because it is self-magnified by the large Jacobian $\|\hat{J}(\hat{x}_0)\|_2 > 1$ after the activation function approximation, as shown in Fig. 3b.

2) Stabilizing fastESN: In order to obtain a reduced network that is asymptotically stable, we can stabilize the DEIM procedure by using the idea presented in [25].

For fastESN, the stabilization procedure can be very simple. The basic idea is to decompose the nonlinear part of ESN (i.e., $f(Wx(k) + W_{\mathrm{in}} u(k))$) into two parts (illustrated in Fig. 4a): a nonlinear part $h(x(k))$ with zero Jacobian at the equilibrium and a linear part which inherits the Jacobian of the original nonlinear part. Then, the function approximation is performed on $h(x(k))$ instead of the original nonlinear part.



(a) The output of the fastESN directly generated by DEIM. The network is not asymptotically stable.

(b) Plot of the first element of $\hat{x}(k)$ (state1) of the unstabilized fastESN. It is magnified by the Jacobian with $\|\hat{J}(\hat{x}_0)\|_2 > 1$, causing the instability problem.

(c) The output of the stabilized fastESN (with label "fastESN") with the same input. After the stabilization, fastESN is asymptotically stable.

Fig. 5: Experimental example of the instability problem of the fastESN on the NARMA10 test bench. The network has 50-order ($q = 50$) generated from an ESN network with 200 internal units ($n = 200$).

Since $h(x(k))$ has zero Jacobian at the equilibrium (Fig. 4b), the 2-norm of its Jacobian will not be magnified to cause the instability problem.

To realize the idea above, we perform Taylor expansion of the original nonlinear part of ESN at the equilibrium $x_0 = 0$ as

$$\begin{aligned} f(Wx(k) + W_{\text{in}}u(k)) &= J(x_0)(x(k) - x_0) + h(x(k)) \\ &= Wx(k) + h(x(k)). \end{aligned} \tag{25}$$

Then, the linear part can be easily obtained as $J(x_0)(x(k) - x_0)$ or $Wx(k)$ which inherits the original Jacobian $J(x_0) = W$. The new nonlinear part $h(x(k))$ with zero Jacobian at the equilibrium is readily solved from (25) as

$$h(x(k)) = f(Wx(k) + W_{\text{in}}u(k)) - Wx(k). \tag{26}$$

With the nonlinear function decomposition described above, the original ESN in (1) can be rewritten using $h(x)$ as

$$\begin{aligned} x(k+1) &= Wx(k) + h(x(k)), \\ y(k+1) &= W_{\text{out}}x(k+1). \end{aligned} \tag{27}$$

Next, for the activation function approximation, we apply the approximation to $h(x(k)) = f(Wx(k) + W_{\text{in}}u(k)) - Wx(k)$ instead of $g(x(k)) = f(Wx(k) + W_{\text{in}}u(k))$, using the function samples $\{h_1, h_2, \ldots, h_{n_s}\}$, to obtain

$$V_h\hat{h}(x) \approx h(x), \tag{28}$$

and

$$\hat{h}(x) = (P_h^{\text{T}}V_h)^{-1}P_h^{\text{T}}h(x), \tag{29}$$

where P_h and V_h are the DEIM matrices generated for the new nonlinear function $h(x)$ by following the same steps presented in Section IV-C2.

The final fastESN is derived by combining (27), (28), and (29) as

$$\begin{aligned} \hat{x}(k+1) &= (V_x^{\text{T}}WV_x - \hat{E}_dP_h^{\text{T}}WV_x)\hat{x}(k) \\ &\quad + \hat{E}_df(P_h^{\text{T}}WV_x\hat{x}(k) + P_h^{\text{T}}W_{\text{in}}u(k)), \\ y(k+1) &= W_{\text{out}}V_x\hat{x}(k+1), \end{aligned} \tag{30}$$

where $\hat{E}_d = V_x^{\text{T}}V_h(P_h^{\text{T}}V_h)^{-1}$.

Experimentally, if we perform the experiment on the stabilized fastESN with the same settings presented in Section IV-D1, the new system is asymptotically stable: its output (with the label "fastESN") tightly follows the target as given in Fig. 5c.

By further defining

$$\begin{aligned} \hat{E}_l &= V_x^{\text{T}}WV_x - \hat{E}_dP_h^{\text{T}}WV_x, & \hat{W} &= P_h^{\text{T}}WV_x, \\ \hat{W}_{\text{in}} &= P_h^{\text{T}}W_{\text{in}}, & \hat{W}_{\text{out}} &= W_{\text{out}}V_x, \end{aligned} \tag{31}$$

the model in (30) becomes the standard fastESN model (4) presented earlier in Section IV-A, which concludes the fastESN generation steps.

E. Evaluation complexity analysis of fastESN

Analysis of the evaluation complexity of fastESN expressed in (4) (or equivalently in (30)) is straightforward as follows.

First, computing $\hat{W}\hat{x}(k)$ and $\hat{W}_{\text{in}}u(k)$ needs q^2 and $q \cdot n_{\text{in}}$ operations, respectively. Then, the nonlinear activation function will be evaluated q times for $f(\cdot)$. Next, there are q^2 operations for $\hat{E}_df(\cdot)$ and another q^2 operations for $\hat{E}_l\hat{x}(k)$. Finally, $\hat{W}_{\text{out}}x(k)$ requires $q \cdot n_{\text{out}}$ operations.

In summary, the evaluation complexity of fastESN is only

$$\mathcal{O}(q^2 + q \cdot n_{\text{in}} + q \cdot n_{\text{out}}), \tag{32}$$

which is independent of the original order n. Since there is $q \ll n$, the evaluation of fastESN can be much faster than the evaluation of the original ESN.

F. Extension for the leaky ESN

To improve learning ability in some applications, the ESN may contain leaky integrator units (we call such ESN as leaky ESN in short) [29]. A leaky ESN is modeled as

$$\begin{aligned} x(k+1) &= (1-\alpha)x(k) + \alpha f(Wx(k) + W_{\text{in}}u(k)), \\ y(k+1) &= W_{\text{out}}x(k+1), \end{aligned} \tag{33}$$

where $\alpha \in (0, 1]$ is the leaky rate [36].

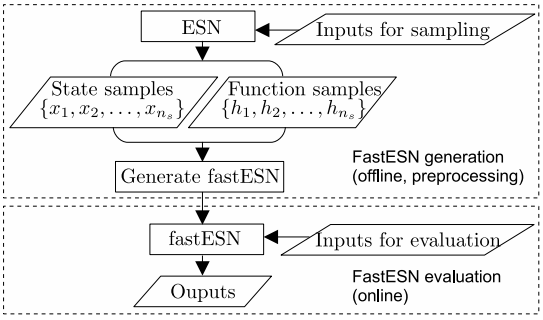


Fig. 6: The flowchart diagram of accelerating ESN evaluation using fastESN.

FastESN is fully compatible with the leaky ESN, which can be easily derived as

$$\begin{aligned}\hat{x}(k+1) &= (1-\alpha)\hat{x}(k) \\ &\quad + \alpha(V_x^{\mathrm{T}} W V_x - \hat{E}_d P_h^{\mathrm{T}} W V_x)\hat{x}(k) \\ &\quad + \alpha \hat{E}_d f(P_h^{\mathrm{T}} W V_x \hat{x}(k) + P_h^{\mathrm{T}} W_{\mathrm{in}} u(k)), \\ y(k+1) &= W_{\mathrm{out}} V_x^{\mathrm{T}} \hat{x}(k+1),\end{aligned} \tag{34}$$

where V_x, P_h, V_h (contained in $\hat{E}_d$) are generated from the state samples and function samples of the leaky ESN in (33) by following the same generation procedures as the standard fastESN.

G. The flow of accelerating ESN evaluation using fastESN

Algorithm 1 The fastESN generation algorithm

Input: samples $\{x_1, x_2, \ldots, x_{n_s}\}$ and $\{h_1, h_2, \ldots, h_{n_s}\}$, ESN matrices W, W_{in}, W_{out}, reduce order q

Output: fastESN matrices $\hat{E}_d$, $\hat{E}_l$, $\hat{W}$, $\hat{W}_{\mathrm{in}}$, $\hat{W}_{\mathrm{out}}$

▷ *Obtain the state approximation matrix V_x*

$X \leftarrow \left[x_1, x_2, \ldots, x_{n_s}\right]$

$V_x, \Sigma_x, U_x \leftarrow \mathrm{SVD}(X)$

$V_x \leftarrow V_x[:, 1:q]$

▷ *Obtain the function approximation matrices V_h and P_h*

$H \leftarrow \left[h_1, h_2, \ldots, h_{n_s}\right]$

$V_h, \Sigma_h, U_h \leftarrow \mathrm{SVD}(H)$

$V_h \leftarrow V_h[:, 1:q]$

$P_h \leftarrow \mathrm{DEIM}(V_h)$ ▷ *DEIM process in Algorithm 1 of [24]*

▷ *Generate fastESN matrices*

$\hat{E}_d \leftarrow V_x^{\mathrm{T}} V_h (P_h^{\mathrm{T}} V_h)^{-1}$

$\hat{E}_l \leftarrow V_x^{\mathrm{T}} W V_x - \hat{E}_d P_h^{\mathrm{T}} W V_x$

$\hat{W} \leftarrow P_h^{\mathrm{T}} W V_x$

$\hat{W}_{\mathrm{in}} \leftarrow P_h^{\mathrm{T}} W_{\mathrm{in}}$

$\hat{W}_{\mathrm{out}} \leftarrow W_{\mathrm{out}} V_x$

The flowchart diagram of accelerating ESN evaluation using fastESN is given in Fig. 6. The flow mainly contains two parts: the offline preprocessing part, as summarized in Algorithm 1, which generates the fastESN from the original ESN, and the online part which evaluates the fastESN instead of the original ESN to gain speed.

V. Experimental results

A. Experimental settings

The original ESNs used in the experiments are generated by TensorFlow v2.3 using the ESN functions from TensorFlow Addons (TFA) v0.11.2. Then we extract the original ESN matrices and perform the rest of the experiments in Python 3 with NumPy v1.18.3, including the building of fastESN and the evaluation of both the original ESN and fastESN. The runtime data (fastESN generation time and network evaluation time) are collected on a Mac computer with Intel i5-8500 CPU (3.0 GHz) and 8 GB memory using a single thread.

We test fastESN on popular benchmarks used previously in the recurrent neural network/ESN research studies: a complex second-order problem, a two-input/two-output nonlinear system, and two NARMA systems (NARMA10 and NARMA30). We also test fastESN for the weather prediction task, with the Jena climate dataset [43].

All test cases share the following experimental settings. The connectivity ratio of the ESN internal layer is 0.1. We use the compressed sparse row (CSR) matrix storage in SciPy to store the sparse weight matrices (W) of ESNs and perform the sparse matrix-vector multiplication (SpMV) using the default SpMV routine in SciPy. It is known that sparse matrix computations only show speed advantages for the large sparse matrices. In our test, the sparse matrix computation is faster than the dense matrix computation when the order of the original ESN exceeds 200. Since the smallest original ESN used in this experiment has an order of 250, we use the SpMV routine for the evaluation of all original ESNs to gain speed.

For all the experiments, the sample number to train the original ESN is 1×10^5. To obtain the samples for the fastESN, the state samples and function samples are directly drawn from the forward propagation results during the ESN training: we take one state sample and one function sample for every 50 time steps in training, collecting a total of 2000 state samples and 2000 function samples (i.e., $n_s = 2000$). The first 50 data in both training and testing samples are washout data (the data points at the beginning of the time series which will not be used). All the error and runtime data are recorded by running the experiments 10 times and taking the average measurements, unless noted explicitly.

B. Results on a complex second-order problem

First, we test fastESN using a complex second-order problem which is also used in [44]:

$$y(k+1) = \frac{y(k)y(k-1)(y(k)+0.25)}{1+y^2(k)+y^2(k-1)} + u(k), \tag{35}$$

where the input $u(k)$ is generated randomly from a uniform distribution in $[0, 0.5]$.

The accuracy results of the fastESN on the test data set are plotted in Fig. 7, with the orders of the ESN and fastESN as 1000 and 80, respectively. The result from the unstabilized fastESN is not plotted because it may not be stable. We see that fastESN, with only 80-order, is able to track the target accurately with observable errors at only very few places. It is also interesting to notice that the outputs of the state

TABLE I: The error, size, and runtime comparisons of the original ESN, state approximation ESN, and fastESN tested on the complex second-order problem in (35). In this table and also Table II, III, IV, and V, MSE_t is the mean square error against the target, MSE_o is the mean square error against the original ESN output, and time is the evaluation time spent on $10{,}000$ time steps.

ESN (sparse)				state approximation ESN							fastESN						
order	MSE_t $\times 10^{-3}$	time (s)	param #	order	MSE_t $\times 10^{-3}$	MSE_o $\times 10^{-3}$	time (s)	param #	speed up	comp ratio	order	MSE_t $\times 10^{-3}$	MSE_o $\times 10^{-3}$	time (s)	param #	speed up	comp ratio
500	0.0035	0.47	26k	10	0.35	0.35	0.37	11k	1.3×	42%	10	0.35	0.35	0.17	320	2.8×	1.2%
				20	0.25	0.25	0.42	21k	1.1×	81%	20	0.25	0.25	0.18	1240	2.7×	4.8%
				40	0.20	0.20	0.54	41k	0.87×	158%	40	0.20	0.20	0.19	4880	2.6×	19%
				80	0.027	0.024	0.77	81k	0.61×	312%	80	0.032	0.029	0.22	19k	2.2×	73%
1000	0.0022	1.3	102k	10	0.34	0.34	0.55	22k	2.3×	22%	10	0.34	0.34	0.17	320	7.5×	0.3%
				20	0.25	0.25	0.63	42k	2.0×	41%	20	0.25	0.25	0.18	1240	7.2×	1.2%
				40	0.22	0.22	0.87	82k	1.5×	80%	40	0.22	0.22	0.19	4880	6.8×	4.8%
				80	0.026	0.024	1.4	162k	0.92×	159%	80	0.030	0.028	0.22	19k	5.7×	19%
2000	0.0016	4.5	404k	10	0.35	0.35	0.90	44k	5.0×	11%	10	0.35	0.35	0.17	320	26.4×	0.08%
				20	0.25	0.25	1.1	84k	4.1×	21%	20	0.25	0.25	0.18	1240	25.3×	0.3%
				40	0.23	0.22	1.5	164k	2.9×	41%	40	0.23	0.22	0.19	4880	23.3×	1.2%
				80	0.035	0.033	2.7	324k	1.7×	80%	80	0.041	0.039	0.22	19k	19.9×	4.7%

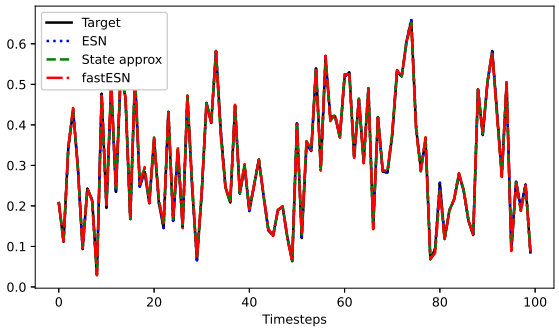


Fig. 7: The accuracy demonstration of fastESN on the complex second-order problem in (35). The orders of the ESN and fastESN are 1000 and 80, respectively.

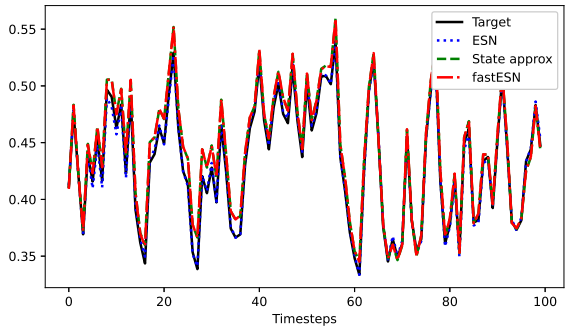


Fig. 8: The accuracy demonstration of fastESN on the two-input/two-output nonlinear system in (36). The orders of the ESN and fastESN are 1000 and 80, respectively.

approximation ESN (the one presented in Section IV-B) and the fastESN are basically indistinguishable. This means the approximation error mainly comes from the state approximation, and the DEIM procedure does not introduce significant error by further reducing the function dimension.

The accuracy, size, and evaluation speed results of the ESN, the state approximation ESN, and fastESN, with different orders, are summarized in Table I.

On the accuracy side, we observe from the table that fastESN, with a much smaller size (parameter number), achieves a similar accuracy compared with the state approximation ESN of the same order. Both of them are able to keep a good evaluation accuracy, with approximation error decreases as the order increases.

For the network size, we see that the ESN contains a large number of parameters, even when we only count the nonzero elements in its internal weight matrix W stored in sparse matrix format. State approximation ESN reduces the parameter count of ESN when its order is small enough: for the original ESN with order 500, a 42% parameter compression ratio is achieved by the state approximation ESN with order 10. However, as its order increases, the size of state approximation ESN grows fast: beyond the order 40, it even contains more parameters than the original ESN. This is because the state approximation ESN only reduces the dimension of the states to the order q, resulting in $n \times q$ weight matrices which may even require more storage than the sparsely stored $n \times n$ weight matrix in ESN. Compared with the state approximation ESN, the fastESN achieves a much higher parameter compression ratio to ESN, because it has $q \times q$ weight matrices thanks to the function dimension reduction via DEIM.

On the evaluation speed side, for some cases, it is even slower to evaluate the state approximation ESN than the original ESN. For example, the speedup of evaluating an 80-order state approximation ESN, generated from a 500-order original ESN, is $0.61\times$ of evaluating the original ESN. This is due to

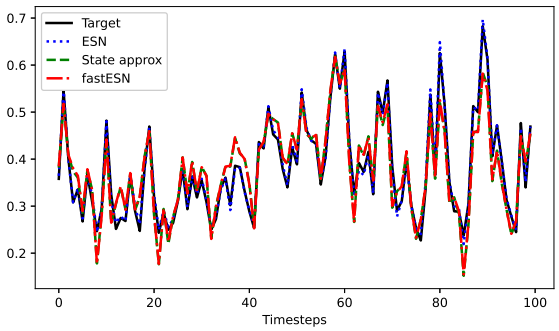


Fig. 9: The accuracy demonstration of fastESN on the NARMA10 system in (37). The orders of the ESN and fastESN are 1000 and 80, respectively.

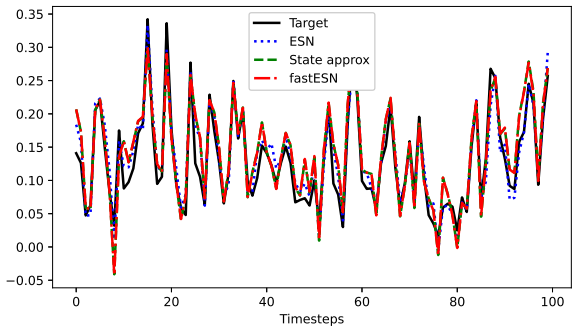


Fig. 10: The accuracy demonstration of fastESN on the NARMA30 system in (38). The orders of the ESN and fastESN are 1000 and 80, respectively.

the fact that the state approximation ESN cannot reduce the function dimension as explained previously. On the other hand, evaluating the fastESN is faster than evaluating the original ESN. As expected, smaller order q used in fastESN always brings forth larger speedup, because the evaluation complexity of fastESN (expressed in equation (32)) only relates to q. The largest speedup in the table is $26.4\times$, when we compare the 10-order fastESN against the 2000-order ESN.

C. Results on a two-input/two-output nonlinear system

Next, we test fastESN on a multiple-input and multiple-output (MIMO) nonlinear system: here we use a two-input/two-output nonlinear system described as

$$\begin{aligned} x_1(k+1) &= 0.5x_1^{2/3}(k) + 0.3x_2(k)x_3(k) + 0.2u_1(k), \\ x_2(k+1) &= 0.5x_2^{2/3}(k) + 0.3x_3(k)x_1(k) + 0.5u_1(k), \\ x_3(k+1) &= 0.5x_3^{2/3}(k) + 0.3x_1(k)x_1(k) + 0.5u_2(k), \\ y_1(k+1) &= 0.7(x_1(k+1) + x_2(k+1)), \\ y_2(k+1) &= 1.5x_1^2(k+1), \end{aligned} \tag{36}$$

where x_1, x_2, x_3 are the three states, u_1 and u_2 are the two inputs, y_1 and y_2 are the two outputs. Each input is generated randomly from a uniform distribution in $[0, 0.5]$. This system is also used in [44].

The evaluation results on the two-input/two-output nonlinear system with 1000-order ESN and 80-order fastESN are plotted in Fig. 8. The accuracy and evaluation speed results of the ESN, the state approximation ESN, and fastESN, with different orders, are summarized in Table II.

Since this test system is a MIMO system, fastESN needs to capture the system information for all input-output pairs, which is more difficult than the previous task. From Fig. 8 and Table II, we see very similar results except for slightly larger errors for both ESN and fastESN, compared with the previous results for the complex second-order problem. This indicates that fastESN is compatible with the ESN for the MIMO system.

D. Results on two NARMA systems

We also test these networks on two NARMA systems: a 10th order one (NARMA10) and a 30th order one (NARMA30). The NARMA10 system is given as

$$\begin{aligned} y(k+1) = 0.3y(k) + 0.05y(k)\sum_{i=0}^{9} y(k-i) \\ + 1.5u(k-9)u(k) + 0.1, \end{aligned} \tag{37}$$

and the NARMA30 system is expressed as

$$\begin{aligned} y(k+1) = 0.2y(k) + 0.04y(k)\sum_{i=0}^{29} y(k-i) \\ + 1.5u(k-29)u(k) + 0.001, \end{aligned} \tag{38}$$

From (37) and (38), we see the order of a NARMA system is defined as the number of its largest delay time steps. Both of the two NARMA systems feature a long delay in time, thus they are often used to test the learning ability of long-term time dependencies. They also appear in [31], [38], [44].

The evaluation results on NARMA10 and NARMA30 are plotted in Fig. 9 and Fig. 10, respectively. The accuracy and evaluation speed results of different networks with different orders are summarized in Table III and Table IV.

As expected, we see that the NARMA systems, due to their long time delay, are more difficult to learn than the previous systems (35) and (36). On NARMA10, the original ESN has larger errors than on the previous systems. The errors become even larger on NARMA30 because of its even longer time delay.

Except for the shared similarities with the previous results, we see that the fastESN achieves similar accuracy against target compared with the original ESN on the NARMA30 benchmark. The reason for this phenomenon is found by further looking at the errors against the original ESN (MSE_o): as the fastESN order grows, MSE_o drops fast and may become the secondary error source against the target. In this situation, the main component of the error against the target for fastESN is the error of the original ESN.

It is also interesting to see that the fastESN of a fixed order is usually less accurate against the original (with larger

TABLE II: The error, size, and runtime comparisons of the original ESN, state approximation ESN, and fastESN tested on the two-input/two-output nonlinear system in (36).

ESN (sparse)				state approximation ESN							fastESN						
order	MSE_t $\times 10^{-3}$	time (s)	param #	order	MSE_t $\times 10^{-3}$	MSE_o $\times 10^{-3}$	time (s)	param #	speed up	comp ratio	order	MSE_t $\times 10^{-3}$	MSE_o $\times 10^{-3}$	time (s)	param #	speed up	comp ratio
500	0.011	0.49	27k	10	0.32	0.31	0.38	12k	1.3×	44%	10	0.32	0.31	0.17	340	2.8×	1.3%
				20	0.25	0.24	0.43	22k	1.1×	81%	20	0.25	0.24	0.18	1280	2.7×	4.7%
				40	0.16	0.14	0.55	42k	0.89×	156%	40	0.16	0.14	0.19	4960	2.5×	18%
				80	0.032	0.022	0.77	82k	0.63×	304%	80	0.034	0.024	0.22	20k	2.1×	74%
1000	0.0066	1.3	104k	10	0.32	0.31	0.55	24k	2.3×	23%	10	0.32	0.31	0.17	340	7.3×	0.3%
				20	0.22	0.22	0.63	44k	2.0×	42%	20	0.22	0.22	0.18	1280	7.1×	1.2%
				40	0.15	0.15	0.88	84k	1.4×	81%	40	0.15	0.15	0.19	4960	6.7×	4.8%
				80	0.081	0.075	1.3	164k	0.94×	158%	80	0.082	0.075	0.22	20k	5.7×	19%
2000	0.0045	4.6	408k	10	0.33	0.32	0.89	48k	5.1×	12%	10	0.33	0.32	0.18	340	25.7×	0.08%
				20	0.24	0.24	1.1	88k	4.3×	22%	20	0.24	0.24	0.18	1280	24.7×	0.3%
				40	0.15	0.15	1.6	168k	2.9×	41%	40	0.15	0.15	0.20	4960	23.1×	1.2%
				80	0.088	0.083	2.7	328k	1.7×	80%	80	0.088	0.084	0.23	20k	20.0×	4.9%

TABLE III: The error, size, and runtime comparisons of the original ESN, state approximation ESN, and fastESN tested on the NARMA10 system in (37).

ESN (sparse)				state approximation ESN							fastESN						
order	MSE_t $\times 10^{-3}$	time (s)	param #	order	MSE_t $\times 10^{-3}$	MSE_o $\times 10^{-3}$	time (s)	param #	speed up	comp ratio	order	MSE_t $\times 10^{-3}$	MSE_o $\times 10^{-3}$	time (s)	param #	speed up	comp ratio
500	0.13	0.48	26k	10	4.9	4.8	0.37	11k	1.3×	42%	10	4.9	4.8	0.17	320	2.8×	1.2%
				20	1.7	1.6	0.42	21k	1.1×	81%	20	1.7	1.6	0.17	1240	2.7×	4.8%
				40	1.6	1.5	0.54	41k	0.88×	158%	40	1.6	1.5	0.18	4880	2.6×	19%
				80	1.3	1.2	0.78	81k	0.61×	312%	80	1.3	1.2	0.21	19k	2.1×	73%
1000	0.069	1.2	102k	10	4.9	4.8	0.55	22k	2.3×	22%	10	4.9	4.8	0.17	320	7.5×	0.3%
				20	1.7	1.6	0.64	42k	1.9×	41%	20	1.7	1.6	0.17	1240	7.3×	1.2%
				40	1.6	1.5	0.87	82k	1.4×	80%	40	1.6	1.5	0.18	4880	6.8×	4.8%
				80	1.4	1.3	1.4	162k	0.90×	159%	80	1.4	1.3	0.22	19k	5.8×	19%
2000	0.037	4.6	404k	10	4.9	4.8	0.92	44k	4.9×	11%	10	4.9	4.8	0.18	320	25.7×	0.08%
				20	1.7	1.6	1.1	84k	4.3×	21%	20	1.7	1.6	0.19	1240	23.7×	0.3%
				40	1.6	1.6	1.6	164k	2.9×	41%	40	1.6	1.6	0.18	4880	24.7×	1.2%
				80	1.4	1.4	2.7	324k	1.7×	80%	80	1.4	1.4	0.22	19k	20.7×	4.7%

TABLE IV: The error, size, and runtime comparisons of the original ESN, state approximation ESN, and fastESN tested on the NARMA30 system in (38).

ESN (sparse)				state approximation ESN							fastESN						
order	MSE_t $\times 10^{-3}$	time (s)	param #	order	MSE_t $\times 10^{-3}$	MSE_o $\times 10^{-3}$	time (s)	param #	speed up	comp ratio	order	MSE_t $\times 10^{-3}$	MSE_o $\times 10^{-3}$	time (s)	param #	speed up	comp ratio
500	1.2	0.48	26k	10	4.9	3.8	0.37	11k	1.3×	42%	10	4.9	3.8	0.17	320	2.8×	1.2%
				20	4.9	3.7	0.41	21k	1.1×	81%	20	4.9	3.7	0.17	1240	2.8×	4.8%
				40	1.6	0.51	0.54	41k	0.88×	158%	40	1.6	0.51	0.18	4880	2.6×	19%
				80	1.3	0.21	0.77	81k	0.62×	312%	80	1.4	0.21	0.21	19k	2.2×	73%
1000	0.69	1.3	102k	10	4.9	4.3	0.56	22k	2.3×	22%	10	4.9	4.3	0.17	320	7.5×	0.3%
				20	5.0	4.3	0.62	42k	2.0×	41%	20	5.0	4.3	0.17	1240	7.5×	1.2%
				40	1.7	1.0	0.86	82k	1.5×	80%	40	1.7	1.0	0.18	4880	7.1×	4.8%
				80	1.3	0.69	1.4	162k	0.88×	159%	80	1.3	0.69	0.21	19k	6.0×	19%
2000	0.18	4.5	404k	10	4.9	4.7	0.90	44k	5.0×	11%	10	4.9	4.7	0.18	320	25.4×	0.08%
				20	4.9	4.7	1.1	84k	4.3×	21%	20	4.9	4.7	0.17	1240	26.0×	0.3%
				40	1.7	1.5	1.6	164k	2.9×	41%	40	1.7	1.5	0.18	4880	24.7×	1.2%
				80	1.3	1.2	2.7	324k	1.7×	80%	80	1.3	1.2	0.21	19k	21.0×	4.7%

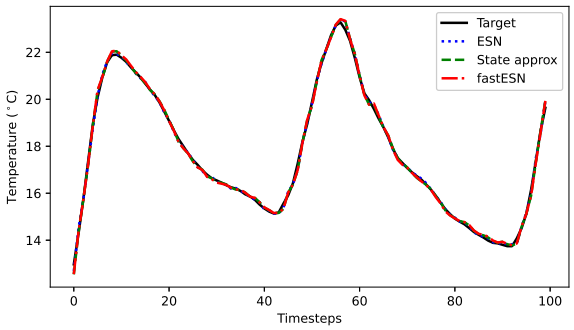


Fig. 11: The accuracy demonstration of fastESN for the weather prediction task on the Jena climate dataset. The orders of the ESN and fastESN are 1000 and 80, respectively.

MSE_o), when it is generated from a larger ESN. This is simply caused by the fact that the fastESN (of a fixed order) generated from a larger ESN has a higher parameter compression ratio, and a higher compression ratio leads to more accuracy loss.

E. Results on the Jena Climate Dataset

We also test fastESN for the weather prediction task on the widely used Jena Climate Dataset [43] recorded by the Max Planck Institute for Biogeochemistry. The data used in this experiment are recorded every ten minutes from January 10, 2009 to December 31, 2016. We take 7 features (pressure, temperature, saturation vapor pressure, vapor pressure deficit, specific humidity, airtight and wind speed) sampled every 30 minutes as the inputs of the ESNs. The data of each feature are normalized to $[-1, 1]$ and smoothed using a 5-step sliding window. The output of the ESNs is the temperature at the next time step.

The weather prediction results on the Jena climate dataset with 1000-order ESN and 80-order fastESN are plotted in Fig. 11. We see that fastESN produces a prediction result almost indistinguishable from that of the original ESN, indicating the high accuracy of the fastESN.

The accuracy and evaluation speed results of the ESN, the state approximation ESN, and fastESN, with different orders on the Jena climate dataset, are summarized in Table V. Please note that the errors appear larger on this benchmark than on the previous benchmarks, only because the absolute target values (temperatures with the largest value over 20) are larger compared with the previous benchmarks (with the largest target value below 1). The errors of fastESN decrease fast with the increase of its order. At the order of 80, fastESN has a mean squared prediction error as small as 0.40×10^{-2} against the 1000-order original ESN. In this case, the mean squared error of the 80-order fastESN against the target ($(\mathrm{MSE}_t = 1.1 \times 10^{-2})$) is only slightly larger than that of the 1000-order original ESN ($\mathrm{MSE}_t = 0.74 \times 10^{-2}$), because the main error source is the error of the original ESN against the target. With this order 80, fastESN achieves a $5.9\times$ speed up and a 19% compression ratio against the 1000-order original ESN, indicating a good performance of fastESN on this real-world benchmark.

F. Comparison between fastESN and ESN of the same size

One advantage of fastESN is that its generation process does not require training data, such that it can accelerate the evaluation of a trained large ESN deployed to hardware without any training data available. But here we still compare the performance of fastESN and a re-trained ESN of the same order, by assuming the training data is available.

This test is conducted on all benchmarks mentioned above, and the order of the trained original ESN (which is only used to build the fastESN here) is 500. The re-trained ESN (of the same order as fastESN) is trained using the same training data used to train the original ESN. All accuracy results in this part are recorded by running the experiments 50 times and taking the average measurements.

The accuracy comparison results of fastESN and ESN with orders 10, 20, 40, and 80, are shown in Fig. 12. We see that fastESN is more accurate than the ESN of the same order in most cases. There are some cases where the re-trained ESN achieves a higher accuracy such as for the orders 20 and 40 on the second-order problem benchmark, but the errors are already extremely small ($\mathrm{MSE}_t < 3\times10^{-4}$) for both networks.

G. The fastESN generation results

In the end, we experimentally analyze the fastESN generation process, i.e. the procedure performed offline, as listed in Algorithm 1. The results of generating fastESNs from a 1000-order ESN on the NARMA10 system with different settings are collected in Table VI. Since the fastESN generation in Algorithm 1 mainly contains the state approximation (from line 1 to line 3) and function approximation (from line 4 to line 7), we also record their runtimes individually.

From Table VI, we can see that the fastESN generation time mainly depends on the sample number used: using more samples will slow down the fastESN generation speed significantly. This is because more samples will make the sample matrices X and H wider, leading to slower SVD processes (line 2 and line 5 in Algorithm 1). We also notice that overly increasing the sample number does not necessarily lead to a significant improvement in accuracy: we hardly get any accuracy improvements when we use over 1000 samples as shown in Table VI. This indicates a good trade-off: we can reduce the sample number to gain speed with limited accuracy sacrifice if the original sample number is too large.

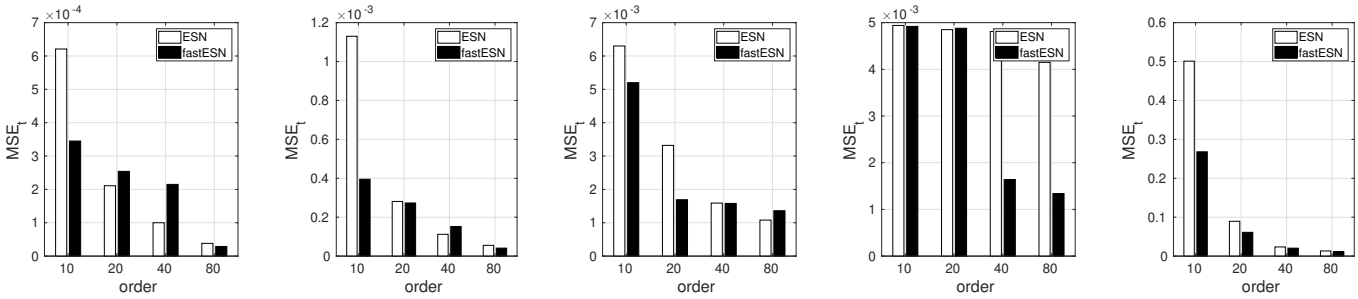
On the other hand, we observe that although fastESN can be generated slightly faster with a smaller order, the fastESN order has a smaller impact on the network generation speed than the sample number. Since a larger order of fastESN means slower online evaluation and higher approximation accuracy, a proper order should be chosen mainly by balancing the network evaluation speed and accuracy, instead of considering the network generation speed.

H. Summary of results

In summary, fastESN shows a high parameter compression ratio and a significant evaluation speedup on all test benchmarks. This result is achieved because fastESN not only

TABLE V: The error, size, and runtime comparisons of the original ESN, state approximation ESN, and fastESN tested on the Jena climate dataset.

ESN (sparse)				state approximation ESN							fastESN						
order	MSE_t $\times 10^{-2}$	time (s)	param #	order	MSE_t $\times 10^{-2}$	MSE_o $\times 10^{-2}$	time (s)	param #	speed up	comp ratio	order	MSE_t $\times 10^{-2}$	MSE_o $\times 10^{-2}$	time (s)	param #	speed up	comp ratio
250	0.90	0.32	8250	10	20.5	19.7	0.29	7000	1.1×	85%	10	24.2	23.4	0.18	380	1.8×	4.6%
				20	4.6	3.7	0.34	12k	0.92×	145%	20	6.0	5.0	0.18	1360	1.8×	16%
				40	1.7	0.76	0.41	22k	0.77×	267%	40	2.2	1.3	0.19	5120	1.6×	62%
				80	1.1	0.20	0.53	42k	0.59×	509%	80	1.2	0.27	0.23	20k	1.4×	242%
500	0.81	0.53	29k	10	21.3	20.6	0.41	14k	1.3×	48%	10	24.0	23.3	0.18	380	2.9×	1.3%
				20	4.4	3.7	0.45	24k	1.2×	83%	20	5.0	4.3	0.18	1360	2.9×	4.7%
				40	1.9	1.1	0.57	44k	0.90×	152%	40	2.2	1.4	0.19	5120	2.7×	18%
				80	1.1	0.30	0.80	84k	0.64×	290%	80	1.2	0.41	0.22	20k	2.3×	68%
1000	0.74	1.3	108k	10	23.0	22.4	0.62	28k	2.2×	26%	10	25.4	24.8	0.18	380	7.6×	0.4%
				20	5.5	4.8	0.69	48k	2.0×	44%	20	6.3	5.6	0.18	1360	7.5×	1.3%
				40	1.8	1.0	0.93	88k	1.4×	81%	40	1.9	1.2	0.19	5120	7.0×	4.7%
				80	1.1	0.35	1.4	168k	0.95×	156%	80	1.1	0.40	0.22	20k	5.9×	19%



(a) Second-order. (b) Two-input/two-output. (c) NARMA10. (d) NARMA30. (e) Climate.

Fig. 12: The accuracy demonstration of fastESN and ESN with the same order on different benchmarks. Here, the fastESNs are generated from 500-order original ESNs.

reduces the state dimension but also reduces the function dimension. By generating a fastESN with $q \times q$ weight matrices from an original $n \times n$ ESN, we can lower the memory requirement and reach the time complexity of $\mathcal{O}(q^2 + q \cdot n_{\text{in}} + q \cdot n_{\text{out}})$ for the first time in ESN evaluation.

VI. Limitations and future work

FastESN is not perfect and the research of ESN acceleration has a lot of opportunities. Here we list some known limitations of fastESN and the corresponding research directions that the community and we can explore in the future.

First, right now we just use POD with simple time domain sampling strategies to generate the projection matrices for both state approximation and function approximation. In the future, we would like to find a new method with an advanced sampling scheme to increase the fastESN accuracy.

Second, fastESN is a fully connected network, which limits its acceleration ratio. In the future, we want to find a new scheme to generate fastESN that preserves the connectivity ratio of the original ESN, or propose a sparsification technique to further accelerate fastESN.

Third, fastESN has an extra layer compared with the original ESN, which introduces two additional matrix multiplications $\hat{E}_l \hat{x}(k)$ and $\hat{E}_d f(\cdot)$. In the future, we wish to find a reduced network which has the same structure as the standard ESN, just purely smaller.

Fourth, there are many new muti-layered ESN structures proposed recently [45], [46]. We think extending fastESN to support these new ESN structures is a very important research direction.

Last but not least, although fastESN works for multi-input applications, it does not work well for the applications with extremely high-dimensional inputs. As a result, developing an evaluation acceleration method which compresses both the internal state dimension and the input dimension of ESNs is desired.

VII. Conclusion

In this article, we have presented the fast echo state network (fastESN), which enables an ESN evaluation complexity of $\mathcal{O}(q^2 + q \cdot n_{\text{in}} + q \cdot n_{\text{out}})$ for the first time. FastESN is generated from an ESN using three techniques including the state approximation, the activation function approximation, and a stabilization technique. The new network is smaller in size than the ESN, thus it can be evaluated with a lower computational complexity independent of the original ESN

TABLE VI: The comparison of generation time and approximation error of fastESNs with different fastESN orders and sample numbers. All of these fastESNs are generated from a 1000-order ESN on NARMA10 data from (37). The time measurements of the state approximation step and the activation function approximation step are recorded in columns "state" and "function" respectively.

fastESN order	sample #	generation time (s)			MSE_o $\times 10^{-3}$
		state	function	total	
10	2000	0.72	0.74	1.5	3.2
	1000	0.36	0.37	0.73	3.2
	500	0.098	0.098	0.20	3.3
	250	0.027	0.032	0.059	3.3
	125	0.010	0.015	0.024	4.1
20	2000	0.71	0.72	1.4	1.6
	1000	0.36	0.37	0.73	1.6
	500	0.10	0.10	0.20	1.7
	250	0.029	0.033	0.062	1.7
	125	0.011	0.018	0.028	1.8
40	2000	0.71	0.71	1.4	1.6
	1000	0.36	0.37	0.73	1.6
	500	0.098	0.11	0.20	1.7
	250	0.026	0.039	0.065	1.8
	125	0.012	0.021	0.033	1.9
80	2000	0.70	0.73	1.4	1.3
	1000	0.37	0.38	0.75	1.3
	500	0.096	0.12	0.22	1.4
	250	0.028	0.052	0.080	1.6
	125	0.010	0.039	0.049	2.5

order. Experiments on four popular test benchmarks reveal that fastESN is able to accelerate the evaluation of sparsely stored ESN with a high parameter compression ratio and a fast evaluation speed.

REFERENCES

[1] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. The MIT Press, 2016.

[2] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis, "Mastering the game of Go with deep neural networks and tree search," *Nature*, vol. 529, pp. 484–489, January 2016.

[3] O. Vinyals, I. Babuschkin, W. M. Czarnecki, M. Mathieu, A. Dudzik, J. Chung, D. H. Choi, R. Powell, T. Ewalds, P. Georgiev, J. Oh, D. Horgan, M. Kroiss, I. Danihelka, A. Huang, L. Sifre, T. Cai, J. P. Agapiou, M. Jaderberg, A. S. Vezhnevets, R. Leblond, T. Pohlen, V. Dalibard, D. Budden, Y. Sulsky, J. Molloy, T. L. Paine, C. Gulcehre, Z. Wang, T. Pfaff, Y. Wu, R. Ring, D. Yogatama, D. Wünsch, K. McKinney, O. Smith, T. Schaul, T. Lillicrap, K. Kavukcuoglu, D. Hassabis, C. Apps, and D. Silver, "Grandmaster level in StarCraft II using multi-agent reinforcement learning," *Nature*, vol. 575, pp. 350–354, October 2019.

[4] P. A. Beerel and M. Pedram, "Opportunities for machine learning in electronic design automation," in *Proc. IEEE Int. Symp. on Circuits and Systems (ISCAS)*, May 2018.

[5] W. L. Neto, M. Austin, S. Temple, L. Amaru, X. Tang, and P.-E. Gaillardon, "LSOracle: a logic synthesis framework driven by artificial intelligence," in *Proc. Int. Conf. on Computer Aided Design (ICCAD)*, November 2019.

[6] H. Wang, T. Xiao, D. Huang, L. Zhang, C. Zhang, H. Tang, and Y. Yuan, "Runtime stress estimation for three-dimensional IC reliability management using artificial neural network," *ACM Trans. on Design Automation of Electronic Systems*, vol. 24, no. 6, pp. 69:1–69:29, November 2019.

[7] H. Wang, X. Guo, S. X.-D. Tan, C. Zhang, H. Tang, and Y. Yuan, "Leakage-aware predictive thermal management for multi-core systems using echo state network," *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, vol. 39, no. 7, pp. 1400–1413, July 2020.

[8] W. Jin, S. Sadiqbatcha, J. Zhang, and S. X.-D. Tan, "Full-chip thermal map estimation for commercial multi-core CPUs with generative adversarial learning," in *Proc. Int. Conf. on Computer Aided Design (ICCAD)*, November 2020.

[9] H. Jaeger, "The "echo state" approach to analysing and training recurrent neural networks," GMD - German National Research Institute for Computer Science, Tech. Rep., 2001.

[10] W. Maass, T. Natschläger, and H. Markram, "Real-time computing without stable states: a new framework for neural computation based on perturbations," *Neural Computation*, vol. 14, no. 11, pp. 2531–2560, November 2002.

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

[12] D. Li, M. Han, and J. Wang, "Chaotic time series prediction based on a novel robust echo state network," *IEEE Trans. on Neural Networks and Learning Systems*, vol. 23, no. 5, pp. 787–799, May 2012.

[13] M. Xu and M. Han, "Adaptive elastic echo state network for multivariate time series prediction," *IEEE Trans. on Cybernetics*, vol. 46, no. 10, pp. 2173–2183, October 2016.

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

[15] S. Han, J. Pool, J. Tran, and W. Dally, "Learning both weights and connections for efficient neural network," in *Proc. Conf. on Neural Information Processing Systems (NeurIPS)*, December 2015, pp. 1135–1143.

[16] S. Chen and Q. Zhao, "Shallowing deep networks: Layer-wise pruning based on feature representations," *IEEE Trans. on Pattern Analysis and Machine Intelligence*, vol. 41, no. 12, pp. 3048–3056, December 2019.

[17] J.-H. Luo, H. Zhang, H.-Y. Zhou, C.-W. Xie, J. Wu, and W. Lin, "ThiNet: Pruning CNN filters for a thinner net," *IEEE Trans. on Pattern Analysis and Machine Intelligence*, vol. 41, no. 10, pp. 2525–2538, October 2019.

[18] M. Jaderberg, A. Vedaldi, and A. Zisserman, "Speeding up convolutional neural networks with low rank expansions," in *Proc. British Machine Vision Conf. (BMVC)*, September 2014.

[19] X. Zhang, J. Zou, K. He, and J. Sun, "Accelerating very deep convolutional networks for classification and detection," *IEEE Trans. on Pattern Analysis and Machine Intelligence*, vol. 38, no. 10, pp. 1943–1955, October 2016.

[20] F. M. Bianchi, E. De Santis, A. Rizzi, and A. Sadeghian, "Short-term electric load forecasting using echo state networks and PCA decomposition," *IEEE Access*, vol. 3, pp. 1931–1943, 2015.

[21] L. Boccato, A. Lopes, R. Attux, and F. J. Von Zuben, "An extended echo state network using volterra filtering and principal component analysis," *Neural Networks*, vol. 32, pp. 292–302, August 2012.

[22] G. Berkooz, P. Holmes, and J. L. Lumley, "The proper orthogonal decomposition in the analysis of turbulent flows," *Annual Review of Fluid Mechanics*, vol. 25, pp. 539–575, 1993.

[23] P. Benner, S. Gugercin, and K. Willcox, "A survey of projection-based model reduction methods for parametric dynamical systems," *SIAM Review*, vol. 57, no. 4, pp. 483–531, 2015.

[24] S. Chaturantabut, "Nonlinear model reduction via discrete empirical interpolation," Ph.D. dissertation, Rice University, May 2011.

[25] A. Hochman, B. N. Bond, and J. K. White, "A stabilized discrete empirical interpolation method for model reduction of electrical, thermal, and microelectromechanical systems," in *Proc. Design Automation Conf. (DAC)*, June 2011.

[26] W. Wen, C. Wu, Y. Wang, Y. Chen, and H. Li, "Learning structured sparsity in deep neural networks," in *Proc. Conf. on Neural Information Processing Systems (NeurIPS)*, December 2016, pp. 2082–2090.

[27] H. Li, A. Kadav, I. Durdanovic, H. Samet, and H. P. Graf, "Pruning filters for efficient ConvNets," in *Proc. Int. Conf. on Learning Representations (ICLR)*, 2017.

[28] H. Mao, S. Han, J. Pool, W. Li, X. Liu, Y. Wang, and W. J. Dally, "Exploring the granularity of sparsity in convolutional neural networks," in *Proc. IEEE/CVF Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2017, pp. 13–20.

[29] H. Jaeger, M. Lukoševičius, P. Dan, and U. Siewert, "Optimization and applications of echo state networks with leaky-integrator neurons," *Neural Networks*, vol. 20, no. 3, pp. 335–352, 2007.
[30] J. J. Steil, "Online reservoir adaptation by intrinsic plasticity for backpropagation-decorrelation and echo state learning," *Neural Networks*, vol. 20, no. 3, pp. 353–364, April 2007.
[31] B. Schrauwen, M. Wardermann, D. Verstraeten, J. J. Steil, and D. Stroobandt, "Improving reservoirs using intrinsic plasticity," *Neurocomputing*, vol. 71, no. 7–9, pp. 1159–1171, March 2008.
[32] C. Gallicchio, A. Micheli, and L. Pedrelli, "Deep reservoir computing: A critical experimental analysis," *Neural Computation*, vol. 268, pp. 87–99, December 2017.
[33] J. Qiao, F. Li, H. Han, and W. Li, "Growing echo-state network with multiple subreservoirs," *IEEE Trans. on Neural Networks and Learning Systems*, vol. 28, no. 2, pp. 391–404, February 2017.
[34] A. Rodan and P. Tiňo, "Simple deterministically constructed cycle reservoirs with regular jumps," *Neural Computation*, vol. 24, no. 7, pp. 1822–1852, July 2012.
[35] S. Løkse, F. M. Bianchi, and R. Jenssen, "Training echo state networks with regularization through dimensionality reduction," *Cognitive Computation*, vol. 9, pp. 364–378, 2017.
[36] M. Lukoševičius, "A practical guide to applying echo state networks," in *Neural Networks: Tricks of the Trade*, G. Montavon, G. B. Orr, and K.-R. Müller, Eds. Springer, Berlin, Heidelberg, 2012, ch. 27, pp. 659–686.
[37] R. Pascanu, T. Mikolov, and Y. Bengio, "On the difficulty of training recurrent neural networks," in *Proc. Int. Conf. on Machine Learning (ICML)*, 2013, pp. 1310–1318.
[38] H. Jaeger, "Adaptive nonlinear system identification with echo state networks," in *Proc. Conf. on Neural Information Processing Systems (NeurIPS)*, January 2002, pp. 609–616.
[39] A. Elafrou, G. Goumas, and N. Koziris, "Performance analysis and optimization of sparse matrix-vector multiplication on modern multi- and many-core processors," in *Proc. Int. Conf. on Parallel Processing (ICPP)*, August 2017, pp. 292–301.
[40] H. Wang, S. X.-D. Tan, and R. Rakib, "Compact modeling of interconnect circuits over wide frequency band by adaptive complex-valued sampling method," *ACM Trans. on Design Automation of Electronic Systems*, vol. 17, no. 1, pp. 5:1–5:22, January 2012.
[41] H. Wang, J. Wan, S. X.-D. Tan, C. Zhang, H. Tang, Y. Yuan, K. Huang, and Z. Zhang, "A fast leakage-aware full-chip transient thermal estimation method," *IEEE Trans. on Computers*, vol. 67, no. 5, pp. 617–630, May 2018.
[42] M. Barrault, Y. Maday, N. C. Nguyen, and A. T. Patera, "An 'empirical interpolation' method: application to efficient reduced-basis discretization of partial differential equations," *Comptes Rendus Mathematique*, vol. 339, no. 9, pp. 667–672, November 2004.
[43] Jena Climate Dataset. [Online]. Available: www.bgc-jena.mpg.de/wetter
[44] A. F. Atiya and A. G. Parlos, "New results on recurrent network training: Unifying the algorithms and accelerating convergence," *IEEE Trans. on Neural Networks*, vol. 11, no. 3, pp. 697–709, May 2000.
[45] Z. K. Malik, A. Hussain, and Q. J. Wu, "Multilayered echo state machine: A novel architecture and algorithm," *IEEE Trans. on Cybernetics*, vol. 47, no. 4, pp. 946–959, April 2017.
[46] Q. Ma, L. Shen, and G. W. Cottrell, "DeePr-ESN: A deep projection-encoding echo-state network," *Information Sciences*, vol. 511, pp. 152–171, February 2020.

Hai Wang received the B.S. degree from the Huazhong University of Science and Technology, Wuhan, China, in 2007, and the M.S. and Ph.D. degrees from the University of California at Riverside, Riverside, CA, USA, in 2008 and 2012, respectively.

He is currently an associate professor with the University of Electronic Science and Technology of China, Chengdu, China. His research interests include design automation of VLSI circuits and systems, software/hardware codesign of computer systems, and artificial intelligence algorithms and hardwares.

Dr. Wang was a recipient of the Best Paper Award nomination from Asia and South Pacific Design Automation Conference (ASP-DAC) in 2019 at Tokyo, Japan. He has served as a Organizing Committee Member of International Conference on Computer Design (ICCD), Technical Program Committee Member of Design, Automation and Test in Europe Conference (DATE), Asia and South Pacific Design Automation Conference (ASP-DAC), International Green and Sustainable Computing Conference (IGSC) and International Symposium on Quality Electronic Design (ISQED), and also served as a Reviewer of many journals including the IEEE Transactions on Computers, the IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, the IEEE Transactions on Parallel and Distributed Systems, and ACM Transactions on Design Automation of Electronic Systems.

Xingyi Long received the B.S. degree from the University of Electronic Science and Technology of China, Chengdu, China, in 2019. He is currently working toward the master's degree at the University of Electronic Science and Technology of China. His current research interests include power analysis and thermal management of integrated circuits, and acceleration techniques for recurrent neural networks.

Xue-Xin Liu received the B.S. and M.S. degrees from Fudan University, Shanghai, China, in 2005 and 2008, respectively and the Ph.D. degree from the University of California at Riverside, Riverside, CA, USA, in 2013.

He is currently with Pure Storage, Inc., Mountain View, CA, USA. His research interests include numerical analysis, circuit simulation, and parallel computing techniques.