Predicting output responses of nonlinear dynamical systems with parametrized inputs using LSTM

Lihong Feng

***Abstract*—Long Short-Term Memory (LSTM) has been more and more used to predict time evolution of dynamics for many problems, especially the fluid dynamics. Usually, it is applied to the latent space after dimension reduction of the full dynamical system by proper orthogonal decomposition (POD), autoencoder (AE) or convolutional autoencoder (CAE). In this work, we propose to directly apply LSTM to the data of the output without dimension reduction for output response prediction. The dimension of the output is usually small, and no dimension reduction is necessary, thus no accuracy loss is caused by dimension reduction. Based on the standard LSTM structure, we propose an LSTM network with modified activation functions which is shown to be much more robust for predicting periodic waveforms. We are especially interested in showing the efficiency of LSTM for predicting the output responses corresponding to time-vary input signals, which is rarely considered in the literature. However, such systems are of great interests in electrical engineering, mechanical engineering, and control engineering, etc. Numerical results for models from circuit simulation, neuron science and a electrochemical reaction have shown the efficiency of LSTM in predicting the dynamics of output responses.**

***Index Terms*—Scientific machine learning, long short-term memory (LSTM), dynamical systems with inputs and outputs, output prediction.**

I. Introduction

In recent years, machine learning has attracted much attention in computational science [1]–[9]. Many of those works try to either combine the idea of model order reduction (MOR) with machine learning or aim at achieving the goal of MOR via neural networks. In contrast to traditional MOR based on projection [10], [11], those works take use of the power of machine learning to fast predict the dynamics without applying projection to the original systems, or even without knowing the governing equations. Scientific machine learning is known to be able to predict dynamics of many fluid problems. In particular, deep learning techniques AE, CAE, LSTM, and deep residual recurrent neural network (DR-RNN) have been applied to fluid problems for dynamics prediction [3], [4], [7], [9], [12]–[15]. Among those works, LSTM combined with CAE or with POD is shown to be able to predict the dynamics in the time domain [7], [9], [15]. A similar strategy of these methods is that dimension reduction is first implemented on the snapshots of the unknown state vector, then evolution prediction is done in the latent space by LSTM. Afterwards, the solution in the original space is recovered by a decoder mapping. In this work, we try to avoid the step of dimension reduction and the step of returning to the original space using a decoder, as these steps will unavoidably induce errors. For output response evolution, we propose to apply LSTM directly to the output data and the trained LSTM network directly outputs the evolution of the output response in the time domain. This is based on the observation that the dimension of the output is often much smaller than that of the state vector in the spatially discretized space, so that LSTM can accurately learn it without data compression. To the best of our knowledge, most of the existing work on LSTM for dynamics prediction only considers systems with no time-varying inputs, except for the work in [16]. LSTM is used in [16] for output prediction of nonlinear circuit models, where only standard LSTM structure is used. The training data are obtained by simulating the circuit model over the whole time interval $[0, T]$ with the training inputs. LSTM is used to predict the output response corresponding to the testing inputs. Compared to the work in [16], there are two main new contributions in this work. Firstly, we test the performance of LSTM on more general input-output systems from different application areas. For certain systems, the standard LSTM structure is not accurate, we further propose a modified LSTM with different activation functions that lead to accurate output prediction. Secondly, the training data for LSTM in this work correspond only to a *part of* the whole time interval. That means, we only simulate the original system with training inputs in a limited time interval $[0, T_0], T_0 < T$ to get the training data. LSTM is then used to predict the output response corresponding to the *testing* inputs in the whole time interval $[0, T]$.

Detailed discussions on machine learning implementation are often neglected in the current literature. To make our results reproducible, we present implementation details of LSTM in Tensorflow in a separate subsection. Finally, sensitivity of LSTM w.r.t. some parameters that may influence the accuracy of the final results is also discussed.

The paper is organized as follows. The next section addresses the systems and the problems we are interested in. This is followed by a detailed introduction of LSTM and its implementation in Tensorflow in Section III, where a variant of LSTM network is also proposed. Section IV presents numerical results of LSTM for three different models. Section V concludes the paper.

II. Systems with Inputs and Outputs

In many engineering applications, systems with inputs and outputs describing certain reactions, signal transmission or

Max Planck Institute for Dynamics of Complex Technical Systems, Sandtorstrasse 1, D-39106 Magdeburg, Germany. feng@mpi-magdeburg.mpg.de

control, are often presented in the following form,

$$\begin{aligned} dg(x(t))/dt &= f(x(t)) + b(u(\mu,t)), \\ y(t) &= c(x(t)), \end{aligned} \tag{1}$$

where $x(t) \in \mathbb{R}^n$ is the state vector, or the solution vector. $g(\cdot) \in \mathbb{R}^n, f(\cdot) \in \mathbb{R}^n$ are system operators and $b(\cdot) \in \mathbb{R}^p, c(\cdot) \in \mathbb{R}^q$ are the input, output operators. When $g(x(t))$ is linear, it is often written as $Ex(t)$. The input operator is usually also a linear matrix, so that $b(u(\mu,t))$ can be written as $Bu(\mu,t)$. The input signal $u(\mu,t)$ may be parametrized with the parameter μ, for example, the frequency. $f(x(t))$ is a nonlinear function of $x(t)$. In the end, we consider a general nonlinear system with inputs and outputs. The proposed method for dynamics prediction is purely based on the output snapshot data, therefore it has no restrictions on the nonlinearity of the system. For systems whose inputs are independent of the parameter μ, the aim of this work is to use the output data in a limited training time interval $[0, T_0]$ to train an LSTM network, and use the trained LSTM network to predict the output evolution outside the training time interval. For systems with parametrized inputs, we aim at using LSTM to predict the output dynamics at out-of-training parameter samples and in the out-of-training time intervals. We also propose a modified LSTM structure using different activation functions, which is found to be especially accurate for learning (nearly) periodic waveforms.

III. LSTM Structure and its Implementations

In this section, we review the structure of standard LSTM and propose a variant of it using different activation functions. Afterwards, we present some implementation details in Tensorflow.

A. LSTM Structure

Recurrent neural networks (RNN) try to learn the future status of a phenomenon based on its understanding of the phenomenon in the past. They are networks with loops, allowing information to be reused. Standard RNNs have the limitation of learning information with long-term dependencies [17], [18]. LSTM [18]–[21] was proposed to overcome this difficulty. Although all recurrent neural networks have the form of a chain of repeating modules (memory blocks) of neural network [7], [22], the main difference between the structure of standard RNNs and that of LSTM is that standard RNN has only a single layer in each repeating block, but each repeating block of LSTM contains four interacting layers. A schematic illustration in Fig. 1 explains the workflow of LSTM in one module at time t_k which is connected with two copies at times t_{k-1} and t_{k+1}. It should be pointed out that in [16], the LSTM block in Fig. 1 is called a LSTM cell, and the combination of all the cells from t_0 to the final time T is called a LSTM block. However, according to the machine learning tool Tensorflow, the LSTM block in [16] refers to a LSTM layer. Therefore, to be consistent with the terminology used in Tensorflow, we call those in Fig. 1 LSTM blocks and the combination of them from t_0 to the final time, an LSTM layer. The k-th module of the standard LSTM structure at time t_k

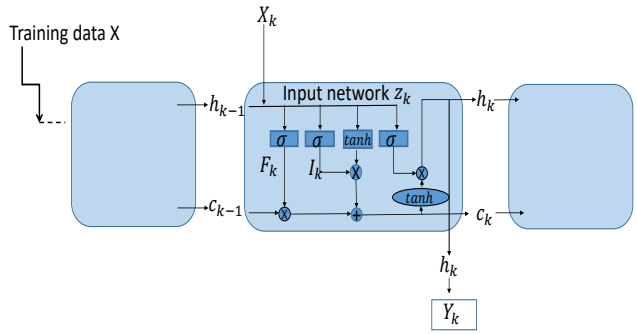

Fig. 1. Structure of LSTM.

can be given in the form of equations as below [7], [22],

$$\begin{aligned} z_k &= \lambda(W_h h_{k-1} + W_X X_k), \\ \mathcal{I}_k &= \sigma(W_{\mathcal{I}} z_k + b_{\mathcal{I}}), \\ \mathcal{F}_k &= \sigma(W_{\mathcal{F}} z_k + b_{\mathcal{F}}), \\ \mathcal{O}_k &= \sigma(W_{\mathcal{O}} z_k + b_{\mathcal{O}}), \\ c_k &= \mathcal{F}_k \odot c_{k-1} + \mathcal{I}_k \odot \xi, \\ \xi &= tanh(W_c z_k + b_c), \\ h_k &= \mathcal{O}_k \odot tanh(c_k), \\ Y_k &= \lambda(W_Y h_k + b_Y). \end{aligned} \tag{2}$$

In (2), z_k is the input layer of LSTM, where X_k is the input data, h_k is the hidden state vector. The weighted (weights W_h, W_X) sum of X_k at t_k and h_{k-1} at the previous time t_{k-1} activated by the activation function λ, gives rise to the input vector z_k. W_s and $b_s, s \in \{\mathcal{I}, \mathcal{F}, \mathcal{O}\}$ are respectively the weight matrices and the bias vectors for the gates and $\mathcal{I}, \mathcal{F}, \mathcal{O}$ are the gate functions, representing the input gate, the forget gate and the output gate. The structures of the three gates are the same, except for the different weights and bias. The input gate decides which values are going to be updated. The forget gate decides what information from the input vector should be removed. The output gate is going to be combined with the cell state c_k, so that the final output Y_k is a filtered version of c_k. The cell state vector c_{k-1} at t_{k-1} is filtered by the forget gate $\mathcal{F}$. The vector ξ is filtered by the input gate $\mathcal{I}$. The two filtered vectors added together, is the cell state vector c_k at the current time t_k. The auxiliary vector ξ is obtained from the input vector z_k activated by $tanh$. At this stage, partial information from the previous time and partial information from the current input vector, are transferred to the cell state vector at the current time. The updated hidden state vector h_k is the $\mathcal{O}$-filtered version of $tanh(c_k)$. It is then used to produce the output Y_k. λ is also the activation function for the output layer.

In our applications, we don't apply activation functions to the input and output layers, that is $\lambda = 1$. Although the standard LSTM uses $\sigma = sigmoid$ function and the $tanh$ function as activation functions, these functions are not accurate for some problems, for example, problems whose outputs are periodic and close to sinusoidal waveforms. For such problems, we propose to use $\sigma = sin$, the sinusoidal

function, and $tanh$ is also replaced by sin. The modified LSTM then has the following structure,

$$\begin{aligned} z_k &= W_h h_{k-1} + W_X X_k, \\ \mathcal{I}_k &= sin(W_{\mathcal{I}} z_k + b_{\mathcal{I}}), \\ \mathcal{F}_k &= sin(W_{\mathcal{F}} z_k + b_{\mathcal{F}}), \\ \mathcal{O}_k &= sin(W_{\mathcal{O}} z_k + b_{\mathcal{O}}), \\ c_k &= \mathcal{F}_k \odot c_{k-1} + \mathcal{I}_k \odot \xi, \\ \xi &= sin(W_c z_k + b_c), \\ h_k &= \mathcal{O}_k \odot sin(c_k), \\ Y_k &= W_Y h_k + b_Y. \end{aligned} \tag{3}$$

The modified LSTM will be used to learn the output waveforms of two examples in the numerical tests.

B. Implementing LSTM in Tensorflow

We implement LSTM in Tensorflow. In this section, we present some implementation details for better understanding its application. Our implementation of LSTM is mainly based on the work in [9], where LSTM is implemented in the latent space after the state snapshot data are compressed by the encoder part of a CAE. In this work we avoid compressing the snapshot data of the state vectors and apply LSTM directly to the output snapshots, thus the CAE is unnecessary. The most important step for correct implementation is the data. The training data are used to train LSTM, and the testing data are used to test the performance of LSTM at the online stage. Since formulation of the data and implementation of LSTM for problems with non-parametric but time-varying inputs and problems with parametrized and time-varying inputs are quite different, we discuss data preparation and LSTM implementation for the two kinds of problems separately.

1) LSTM implementation for problems with non-parametric but time-varying inputs: For problems with non-parametric but time-varying inputs, the training data include snapshots of the output at time instances in the training time interval $[0, T_0]$. In Tensorflow, the shape of the training data is (n_{T_0}, q), where n_{T_0} is the total number of time instances in $[0, T_0]$. Recall that q is the total number of outputs. The input sequence of LSTM is formed by the training data when a time window moves. Assume the width of the time window is w, this means there are w output snapshots at w time instances. Every time, the time window moves one time step forward and LSTM predicts the outputs at a new time instance next to the time window. When the time window sweeps all the time instances in the training time interval, LSTM predicts the output responses at all the time instances ahead of the initial time window. Finally, LSTM is trained. This process can be mathematically described as below,

$$\begin{aligned} \left[y^1, \ldots, y^w\right] &\rightarrow y_{NN}^{w+1}, \\ \left[y^2, \ldots, y^{w+1}\right] &\rightarrow y_{NN}^{w+2}, \\ &\vdots, \\ \left[y^{n_{T_0}-w}, \ldots, y^{n_{T_0}-1}\right] &\rightarrow y_{NN}^{n_{T_0}}, \end{aligned} \tag{4}$$

where $y^i \in \mathbb{R}^q, i = 1, \ldots, n_{T_0} - 1$ are the original outputs and $y_{NN}^i \in \mathbb{R}^q, i = w+1, \ldots, n_{T_0}$ are the LSTM-predicted outputs at $t_{w+1}, \ldots, t_{n_{T_0}}$. Rows on the left-hand side of (4) constitute the whole input sequence, which is a 3D tensor of shape $(n_{T_0} - w, w, q)$ in Tensorflow, where $n_{T_0} - w$ are the number of the time instances $t_{w+1}, \ldots, t_{n_{T_0}}$ outside of the time window, but inside the training time interval $[0, T_0]$. The predicted output at those time instances are the output sequence on the right-hand side of (4), which is of shape $(n_{T_0} - w, q)$. The loss function is defined as the mean squared error (MSE) between the predicted output and the original output at $t_{w+1}, \ldots, t_{n_{T_0}}$, i.e.,

$$\frac{1}{n_{T_0} - w} \sum_{i=w+1}^{n_{T_0}} \|y_{NN}^i - y^i\|_2^2.$$

At the online stage, the output data in the time window are fed into the trained LSTM that then predicts the outputs outside of the time window in a larger time interval $[0, T], T > T_0$. In other words, assuming that the time window size w corresponds to the time interval $[0, T_w]$, then LSTM predicts the output in the time interval $[T_w, T]$ at the online stage. For simplicity of explanation, we call $[T_w, T]$ the testing time interval in the following text.

2) LSTM implementation for problems with parametrized and time-varying inputs: For problems with parametrized and time-varying inputs, the training data for LSTM are the output snapshots at a set of training samples of the parameter μ in a training time interval $[0, T_0]$. Each training sample of μ corresponds to a vector (matrix for systems with multiple outputs) of outputs in the training time interval. The training data is a 3-D tensor including the training parameter samples, the outputs at different time instances and the output dimension. In Tensorflow, the training data have the shape (n_μ, n_{T_0}, q), where n_μ is the total number of training parameter samples. The input sequence and output sequence for training LSTM are then formulated similarly as in (4), by also taking the parameter samples into consideration. Mathematically, the input and output sequences can be described as below,

$$\begin{aligned} \left[y^{\mu_1,1}, \ldots, y^{\mu_1,w}\right] &\rightarrow y_{NN}^{\mu_1,w+1}, \\ \left[y^{\mu_1,2}, \ldots, y^{\mu_1,w+1}\right] &\rightarrow y_{NN}^{\mu_1,w+2}, \\ &\vdots, \\ \left[y^{\mu_1,n_{T_0}-w}, \ldots, y^{\mu_1,n_{T_0}-1}\right] &\rightarrow y_{NN}^{\mu_1,n_{T_0}}, \\ &\vdots, \\ \left[y^{\mu_{n_\mu},1}, \ldots, y^{\mu_{n_\mu},w}\right] &\rightarrow y_{NN}^{\mu_{n_\mu},w+1}, \\ \left[y^{\mu_{n_\mu},2}, \ldots, y^{\mu_{n_\mu},w+1}\right] &\rightarrow y_{NN}^{\mu_{n_\mu},w+2}, \\ &\vdots, \\ \left[y^{\mu_{n_\mu},n_{T_0}-w}, \ldots, y^{\mu_{n_\mu},n_{T_0}-1}\right] &\rightarrow y_{NN}^{\mu_{n_\mu},n_{T_0}}. \end{aligned} \tag{5}$$

The left-hand side of (5) is the input sequence, which is a 3D tensor of shape $(n_\mu(n_{T_0} - w), w, q)$ in Tensorflow. The output sequence of LSTM is the right-hand side of (5), which is a 2D tensor of shape $(n_\mu(n_{T_0} - w), q)$ in Tensorflow. They are the output snapshots at $n_{T_0} - w$ time instances $t_{w+1}, \ldots t_{n_{T_0}}$ outside of the time window corresponding to n_μ training samples.

The loss function for this kind of problems is defined as the mean squared error (MSE) between the predicted output and

the original output at the training parameter samples and the training time instances $t_{w+1}, \dots t_{n_{T_0}}$, i.e.,

$$\frac{1}{n_\mu}\frac{1}{n_{T_0}-w}\sum_{j=1}^{n_\mu}\sum_{i=w+1}^{n_{T_0}}\|y_{NN}^{j,i}-y^{j,i}\|_2^2.$$

At the online stage, the output data in the time window $[0, T_w]$ corresponding to the testing (out-of-training) parameter samples are first computed, then they are fed into the trained LSTM network for it to proceed and to predict the outputs in the testing time interval $[T_w, T]$.

In the next section, we present the performance of LSTM in Tensorflow on three systems with time-varying inputs from different applications. For all the three models, the LSTM network contains two LSTM layers. The outputs of the first LSTM layer are h_k, they are then used as input data for the next LSTM layer. The final output of the LSTM network is given by a separate output layer. The dimension of the output layer is the output dimension of the model, i.e., q in (1). The number of neurons is the number of units on each layer (input layer, input gate, forget gate, output gate, hidden state) in (2) or (3). It is also the length of the vectors z_k, c_k, ξ, h_k, respectively. The LSTM structure for the nonlinear circuit model is the one in (2) with $\lambda = 1$. To compare the performance of LSTM in (3) with sinusoidal activation function and that of the standard structure (2), we apply both LSTM structures to the FitzHugh-Nagumo model and the electrochemistry model. The details of the LSTM structures are listed in Table I. Table I lists the number of

TABLE I
LSTM CONFIGURATION FOR DIFFERENT MODELS

Model	unit #	q	activ.fun.	recur.activ.fun
Circuit	40	1	$tanh$	$sigmoid$
FitzHugh	60	2	tf.math.sin	tf.math.sin
FitzHugh	60	2	$tanh$	$sigmoid$
Electrochem.	40 (90)	3	tf.math.sin	tf.math.sin
Electrochem.	40	3	$tanh$	$sigmoid$

hidden units (unit #) on each layer in (2) or (3), the output dimension q, the activation functions used in the LSTM layers for each model. Activ.fun is the activation function applied to ξ and c_k, recur.active.fun is the activation function applied to the gates. They are called recurrent activation when setting LSTM in Tensorflow. Note that for the electrochemistry model, we use 40 hidden units for the non-parametric case, but 90 for the parametric case. We find that using only 40 hidden units, LSTM (3) cannot produce more accurate results than using 90. Since LSTM (2) with standard structure is inaccurate for this model, it is sufficient to present the results of it only for the non-parametric case, for which 40 hidden units are used.

3) Data normalization and solution recovery: Data normalization is a necessary step for successful training of almost all neural networks, including LSTM. Data normalization means the input data fed into the neural network must first be normalized so that the values of all the data after normalization are in the range $[-1, 1]$. Let $\tilde{Y}$ represent the original data in tensor form, there are different ways of normalization. One common way is:

$$Y = \frac{\tilde{Y}-\tilde{Y}_{\min}}{\tilde{Y}_{\max}-\tilde{Y}_{\min}}, \tag{6}$$

where $\tilde{Y}_{\max}$ is the maximal entry (element) in $\tilde{Y}$, and $\tilde{Y}_{\min}$ is the minimal entry in $\tilde{Y}$. Another way of normalization is simply dividing $\tilde{Y}$ by the maximal absolute value of the entries in $\tilde{Y}$, i.e.,

$$Y = \frac{\tilde{Y}}{|\tilde{Y}|_{\max}}, \tag{7}$$

where $|\cdot|$ takes the absolute value of $\tilde{Y}$ element-wise. $|\tilde{Y}|_{\max}$ is the maximal element in $|\tilde{Y}|$. In our numerical tests, we use both ways of normalization, since we find that the first way of normalization does not give better results than the second one for some problems. In particular, we apply the second way of normalization to the first two examples and the first way of normalization to the third model. For the first two models, data of all outputs are collected into $\tilde{Y}$ to do normalization. For the third example, the data corresponding to each output are normalized separately, otherwise, no accurate results can be obtained. This might be because the output magnitudes of the third example are quite different, which can be seen from Fig. 7 in Subsection IV-C of the numerical results.

When training LSTM, the data to be normalized are the training data. At the online stage, the data in the time window must be first generated for LSTM to do prediction. Those window data must also be normalized. For problems with non-parametrized and time-varying inputs, the normalized window data is already available when training LSTM. They can be directly fed into LSTM for prediction. The LSTM predicted output are also in the normalized range, and need to be returned to the original data range by doing inverse computations of the normalization in (6) or (7). For problems with parametrized and time-varying inputs, the window data corresponding to any new testing parameter are normalized by $\tilde{Y}_{\max}, \tilde{Y}_{\min}$ or $|\tilde{Y}|_{\max}$ of the training data, where $\tilde{Y}$ represents the training data. The LSTM predicted output then return to the original data range also using those values for the inverse of normalization.

IV. NUMERICAL RESULTS

We test the performance of LSTM with the structures listed in Table I on three models. The detailed information of each model and the results of LSTM are discussed separately in the following subsections. For each example, we present the performance of LSTM on two cases:

- case 1. The system with a non-parametrized input.
- case 2. The system with a parametrized input.

For case 1, the data of the output in a training time interval $[0, T_0], T_0 < T$ are used to train LSTM. The trained LSTM is then used to predict the output in the testing time interval $[T_w, T], T_w < T_0$. For case 2, the data of the output corresponding to inputs with training parameter samples in the training time interval are used to train LSTM. The trained LSTM is used to predict the output corresponding to inputs at new *testing* parameter samples in the *testing* time interval. For both cases, we also compute the relative error of the LSTM

This article has been accepted for publication in IEEE Journal on Multiscale and Multiphysics Computational Techniques. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/JMMCT.2023.3242044

prediction for each output over the time interval, which is defined as

$$\epsilon = |y - y_{NN}|/\|y\|_{\max} \in \mathbb{R}^{n_T},$$

where y is a vector of the output response at n_T time instances in the testing time interval $[T_w, T]$, which is computed by directly simulating the original model. y_{NN} is a vector of the output response at the same time instances, but is computed by the LSTM network. $\|\cdot\|_{\max}$ is the matrix maximum norm, defined as the maximal absolute values among all entries of y. $|\cdot|$ takes the absolute value of $y - y_{NN}$ element wise, such that $\epsilon \in \mathbb{R}^{n_T}$ is still a vector. Instead of plotting the relative error changing with time, we list the maximal relative errors and the mean relative errors [1] in a table for each example. They are defined as,

$$\epsilon_{\max} = \max_i \epsilon_i, \quad \epsilon_{\text{meam}} = \frac{1}{n_T}\sum_i \epsilon_i,$$

where ϵ_i is the i-th entry of ϵ. For simplicity of expression, we use y to refer to the output computed from direct numerical simulation (DNS), y_{NN} refers to the prediction by LSTM in all legends of the figures. The specific name of y is indicated beside the y-axis of the figures.

A. Nonlinear Circuit

The first model we test is a nonlinear circuit [23] in Fig. 2. It is a circuit with diodes which makes the system nonlinear. The total number of nodes in the circuit is $\bar{n} = 500,000$, resulting in a system of 500,000 nonlinear ordinary differential equations (ODEs). The system of ODEs with input is given as below,

$$\begin{array}{rcl} \frac{dx}{dt} & = & f(x(t)) + bu(\mu, t), \\ y(t) & = & cx(t), \quad t \in [0, 10], \end{array} \tag{8}$$

where the nonlinear function $f(x(t))$ is an exponential function of $x(t)$. The output $y(t)$ is the voltage v_1 at node 1 in Fig. 2, where $g(v) = e^{40v} + v - 1$. For this example, we test LSTM using two different input signals, one is a step function with step time $t = \mu/10s$, i.e., $u(\mu, t) = 0, t < \mu/10s$ and $u(\mu, t) = 1$, otherwise. Here, μ is taken as a parameter, meaning the step time of the input signal may change. The other input signal is an exponential function $u(\mu, t) = e^{-\mu t/10}$ with μ being the parameter. Recall that in the original description of the model in [23], μ has fixed values $\mu = 30$ and $\mu = 10$ for the step input and exponential input, respectively. In this work, we are also interested in LSTM prediction for output response variations corresponding to parametrized input signals in the time interval $t \in [0, 10]s$ and in the parameter domain $\mu \in [1, 10]$. Note that only the data of the output are used to train LSTM, the training time of LSTM is actually independent of the number of ordinary differential equations.

[1]From our observation, using the same number of epochs, the maximal and mean relative errors may vary by running the same code with different versions of Tensorflow and Python. Therefore, we need to specifically point out that we used Tensorflow version 2.3.0, Python 3.8.11 and Keras 2.4.3 to get the results in this work.

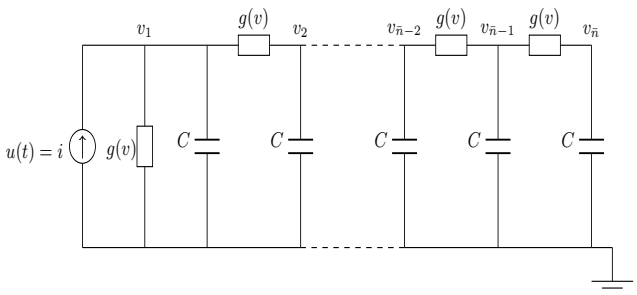

Fig. 2. A nonlinear circuit model.

1) Results of LSTM on case 1: We show the results of LSTM for the input with fixed $\mu = 1$. We get the output snapshots by simulating the system in (8) using backward Euler method with time step $\delta t = 0.02$. The training data corresponding to the step input are the output snapshots in the time interval $[0, 4]s$, those corresponding to the exponential input are the output snapshots in the time interval $[0, 6]s$. After LSTM is trained, a time window of size 100 is used to activate LSTM for the step input. This means output data at the first 100 time instances in the time interval $[0, 2]s$ are used as the time window data for LSTM to proceed, the outputs in the testing time interval $[2, 10]s$ are then predicted. In contrast, the time window size 200 is used for the exponential input. For this case, LSTM needs more time window data to predict the output with acceptable accuracy. Fig. (3a) and Fig. (4a) show the results corresponding to the step input and the exponential input, respectively. The red dashed line shows the predicted output and the blue solid line is the result from direct numerical simulation. We see that the predicted part fits the original one well.

2) Results of LSTM on case 2: The input now is parametrized with μ. For the step input, we use 5 samples of μ, i.e., $\mu = 1, 3, 5, 7, 9$ to generate the training data in the time interval $[0, 4]s$. The same window size of 100 is used for network training and prediction. LSTM is then used to predict the outputs corresponding to inputs with testing samples of μ, i.e., $\mu = 2, 4, 6, 8$ in the testing time interval $[2, 10]s$. In Fig. (3b), the predicted outputs at the testing inputs are compared with the original ones from direct numerical simulation. For the exponential input, the same samples of μ for the step input are used here for generating the training and testing data, respectively. The training time interval and the window size are the same as for case 1, i.e., [0, 6]s and 200, respectively. LSTM predicts the outputs at the testing parameter samples in the testing time interval $[4, 10]s$. The predicted results for different testing samples of μ are now presented in Fig. (4b). The prediction is acceptable in general, though a bit larger error can be observed at $\mu = 2$.

We list the errors of LSTM prediction in Tables II-III for the step input and the exponential input, respectively. For the non-parametric case 1, ϵ_{mean} and $\epsilon_{\max}$ are the mean relative error and the maximal relative error in the predicted time interval. For the parametric case 2, they are the corresponding errors at the testing samples of the parameter μ. The overall error is acceptable. The errors corresponding to the exponential input

This article has been accepted for publication in IEEE Journal on Multiscale and Multiphysics Computational Techniques. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/JMMCT.2023.3242044

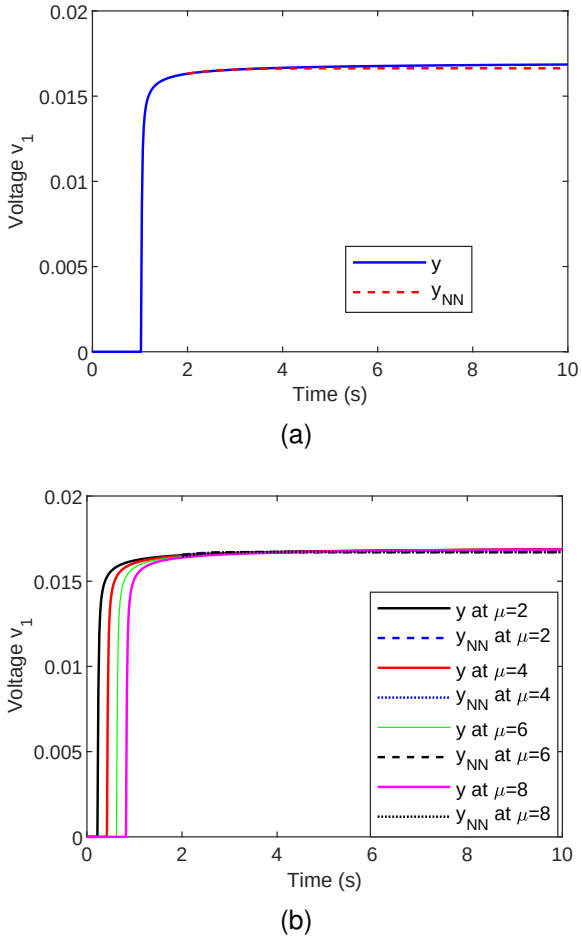

Fig. 3. Circuit model with step input $u(\mu,t)=0, t<\mu/10s$ and $u(\mu,t)=1$, otherwise. (a) Output y from DNS and LSTM prediction y_{NN} for v_1 with input $u(1,t), t\in[0,10]s$. (b) Output y from DNS and LSTM prediction y_{NN} for v_1 corresponding to the testing step inputs $u(\mu,t), t\in[0,10]s$, $\mu=2,4,6,8$.

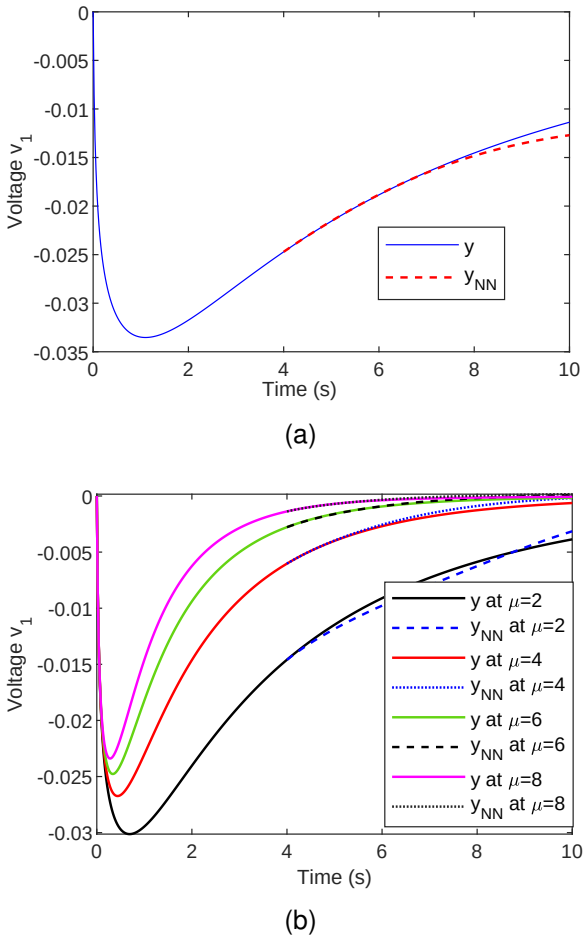

Fig. 4. Circuit model with exponential input $u(\mu,t)=e^{-\mu t/10}$. (a) Output y from DNS and LSTM prediction y_{NN} for v_1 with input $u(1,t), t\in[0,10]s$. (b) Output y from DNS and LSTM prediction y_{NN} for v_1 corresponding to the testing exponential inputs $u(\mu,t), t\in[0,10]s$, $\mu=2,4,6,8$.

are larger than those corresponding to the step input. This is because that the step output goes to almost constant after some time, while the exponential output continues changing till the final time. This shows that LSTM becomes less accurate for outputs with more complex waveforms.

TABLE II
ERROR OF LSTM FOR THE CIRCUIT MODEL, STEP INPUT

Cases	ϵ_{mean}	ϵ_{max}	training μ	testing μ
Case 1	0.0070	0.0131	1	—
Case 2	0.0049	0.0091	1,3,5,7,9	2
Case 2	0.0048	0.0090	1,3,5,7,9	4
Case 2	0.0048	0.0088	1,3,5,7,9	6
Case 2	0.0049	0.0086	1,3,5,7,9	8

B. FitzHugh-Nagumo Model

The second model is a prototype of an excitable system, for example, a neuron [24]. If the external stimulus exceeds a certain threshold value, then the system will exhibit a characteristic excursion in phase space, representing activation and deactivation of the neuron. It is a model with cubic nonlinearity. The system in the form of partial differential equations (PDEs) is described as below [25],

$$\begin{aligned}\varepsilon\frac{\partial v(x,t)}{\partial t} &= \varepsilon\frac{\partial^2 v(x,t)}{\partial x^2}+f(v(x,t))-w(x,t)+c,\\ \frac{\partial w(x,t)}{\partial t} &= bv(x,t)-\gamma w(x,t)+c,\ x\in[0,L], t\in[0,5]s,\end{aligned} \tag{9}$$

where the nonlinear function $f(v)=v(v-0.1)(1-v)$, $L=1, \varepsilon=0.015, b=0.5, \gamma=2, c=0.05$. The initial and boundary conditions are given as

$$\begin{aligned} &v(x,0)=0, && w(x,0)=0, && x\in[0,L],\\ &v_x(0,t)=-i_0(\mu,t), && v_x(L,t)=0, && t\in[0,5]s.\end{aligned} \tag{10}$$

The input signal is a stimulus $i_0(\mu,t)=5\times10^4 t^3 e^{(-\mu t)}$. Although in the original model [25], $\mu=15$, here we consider the parametric form of $i_0(\mu,t)$ with $\mu\in[13,19]$ being the

TABLE III
ERROR OF LSTM FOR THE CIRCUIT MODEL, EXPONENTIAL INPUT

Cases	ϵ_{mean}	ϵ_{max}	training μ	testing μ
Case 1	0.0116	0.0539	1	—
Case 2	0.0294	0.0493	1,3,5,7,9	2
Case 2	0.0410	0.0741	1,3,5,7,9	4
Case 2	0.0392	0.0882	1,3,5,7,9	6
Case 2	0.0816	0.01728	1,3,5,7,9	8

This article has been accepted for publication in IEEE Journal on Multiscale and Multiphysics Computational Techniques. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/JMMCT.2023.3242044

parameter. We aim at observing the response of the model with a parametrized input whose magnitude and decay rate change with μ. The output responses are the voltage $v(0,t)$ and the recovery of the voltage $w(0,t)$ on the left boundary. The training data of this model is obtained via numerical simulation of the spatially discretized PDEs. We use finite difference to discretize the PDEs. The resulting discrete system has 28,672 degrees of freedom. Likewise, only the output data are used to train LSTM, the training time of LSTM is independent of the degrees of freedom after numerical discretization in space. Due to space limitation, we only present the results of LSTM with modified structure in (3) in figures. The errors of LSTM with both structures are listed in Table IV and Table V, respectively for comparison.

1) Results of LSTM on case 1: We first show the performance of LSTM on the system with non-parametric input $i_0(t) = 5 \times 10^4 t^3 e^{(-15t)}$, where $\mu = 15$. The training output data are obtained by simulating the discretized system using the function ode15s in MATLAB till half of the whole time interval: $[0, 2.5]s$, where 250 time steps are taken. Compared to the first example, we need a relatively longer time interval to train LSTM for this example. Whereas, the time window size is only 25, i.e., only the output data at the first 25 time instances in the time interval $[0, 0.25]s$ are used for LSTM to proceed. The LSTM then is able to predict the output in the testing time interval $[0.25, 5]s$. Fig. 5 shows the results of LSTM for the two outputs: the voltage and recovery of the voltage. The predictions in the time interval $(0.25, 5]s$ for both outputs are good in general, but there is a visible mismatch for the recovery of the output around time 3s.

2) Results of LSTM on case 2: When the input is parametrized with μ, we take four samples of $\mu : 13, 15, 17, 19$ to generate the training input signal and the training output data. The training outputs are obtained by simulating the original model also till half of the final time: $[0, 2.5]s$. The testing samples of $\mu : 14, 16, 18$ are used to generate the testing inputs. For the parametric case, a larger time window size of 40 is used for LSTM in both the training stage and the online prediction stage, meaning output data in the time interval $[0, 0.4]s$ are first generated by direct numerical simulation. At the online stage, LSTM predicts the output dynamics in the testing time interval $[0.4, 5]s$ corresponding to those testing parameters. Fig. 6 plots the outputs predicted by LSTM at the testing parameter samples of $\mu : 14, 16, 18$, respectively. For this model, it is seen that LSTM prediction is very accurate in the training time interval $[0, 2.5]s$, but less accurate outside of the training time interval. Table IV lists the errors of LSTM (3) for both cases. LSTM has larger maximal error for predicting the voltage. Table V lists the errors of LSTM (2). Results in the two tables show that after proper training, LSTM with both structures predict the output responses of this model with similar accuracy.

C. An Electrochemistry Model

The last example is a model describing Ferrocyanide oxidation reaction. The governing equations of this model are two PDEs for the mass transport of oxidant and reductant

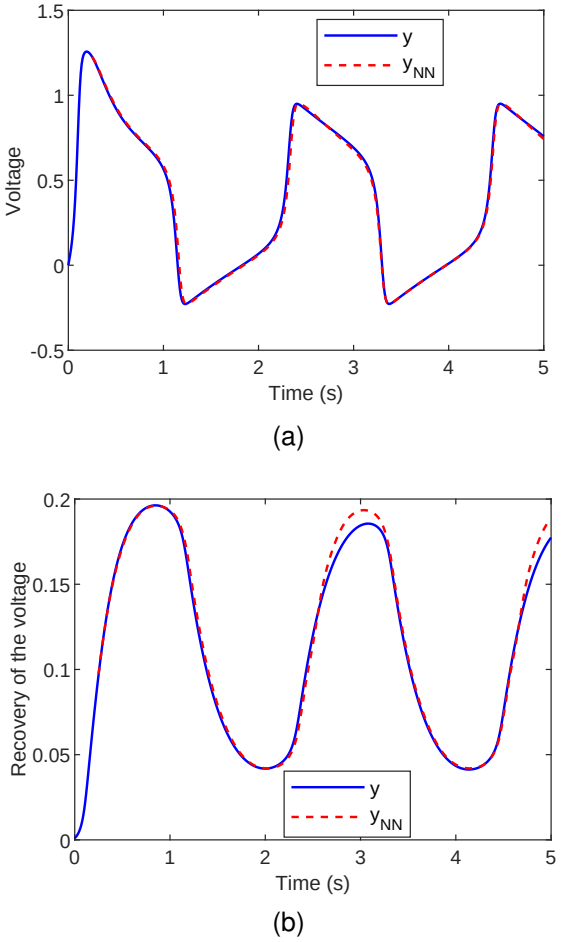

Fig. 5. FitzHugh-Nagumo model with input $i_0(t) = 5 \times 10^4 t^3 e^{(-15t)}$. (a) Output y from DNS and LSTM (3) prediction y_{NN} for the voltage on the left boundary. (b) Output y from DNS and LSTM (3) prediction y_{NN} for the recovery of the voltage on the left boundary.

TABLE IV
ERROR OF LSTM (3) FOR THE FITZHUGH-NAGUMO MODEL: V: VOLTAGE, V_{Re}: RECOVERY OF THE VOLTAGE. TRAINING μ: 13,15,17,19

Error	Case 1	Case 2 at μ: 14, 16, 18		
ϵ_{mean}	0.0107	0.0281	0.0188	0.0191
$\epsilon_{\text{max}}(V)$	0.1084	0.3321	0.2906	0.2979
$\epsilon_{\text{mean}}(V_{Re})$	0.0163	0.0199	0.0126	0.0123
$\epsilon_{\text{max}}(V_{Re})$	0.0596	0.0945	0.0752	0.0730

TABLE V
ERROR OF LSTM (2) FOR THE FITZHUGH-NAGUMO MODEL: V: VOLTAGE, V_{Re}: RECOVERY OF THE VOLTAGE. TRAINING μ: 13,15,17,19

Error	Case 1	Case 2 at μ: 14, 16, 18		
ϵ_{mean}	0.0216	0.0194	0.0290	0.0248
$\epsilon_{\text{max}}(V)$	0.2176	0.2656	0.3554	0.3473
$\epsilon_{\text{mean}}(V_{Re})$	0.0173	0.0173	0.0230	0.0199
$\epsilon_{\text{max}}(V_{Re})$	0.0624	0.0720	0.0929	0.0851

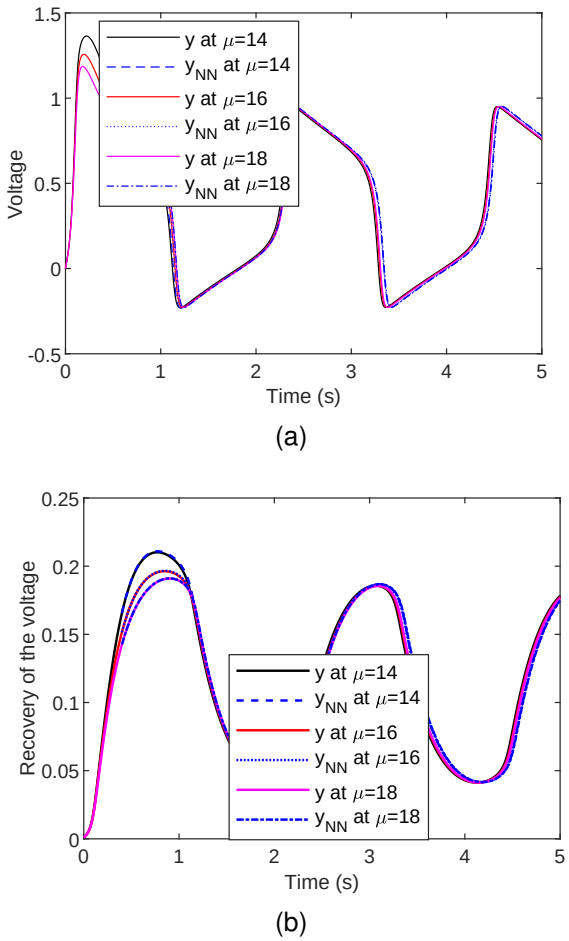

Fig. 6. FitzHugh-Nagumo model with parametrized input $i_0(t) = 5 \times 10^4 t^3 e^{(-\mu t)}$, $\mu = 14, 16, 18$. (a) Output y from DNS and LSTM (3) prediction y_{NN} for the voltage on the left boundary. (b) Output y from DNS and LSTM (3) prediction y_{NN} for the recovery of the voltage on the left boundary.

described by Fick's law in (11) and one ODE for the charge balance in (12). The PDE also includes electrochemical reaction kinetics as a boundary condition.

$$\frac{\partial C_i(z,t)}{\partial t} = D_i \frac{\partial^2 C_i(z,t)}{\partial z^2}, \tag{11}$$

where the subscript i stands for oxidant (ox) or reductant (red) in the reaction, and both reduction and oxidation can occur in the system. C_i, D_i are the corresponding concentration and diffusion coefficient, respectively.

$$C_{dl}\frac{dE(t)}{dt} = J(E, u(\omega,t)) - Fr(t), \tag{12}$$

where C_{dl} is the double-layer capacitance and $J(E, u(\omega,t))$ is the cell current density depending on the electrode potential E and the input signal $u(\omega,t)$. The main source of nonlinearity is the reaction rate $r(t)$ in (13) computed by Butler-Volmer kinetics, where E_r is the equilibrium electrode potential and $k = 1.15 \times 10^{-4}$ is the reaction rate constant.

$$r(t) = k\left\{c_{\rm red} e^{\beta f(E(t)-E_{\rm r})} - c_{\rm ox} e^{-(1-\beta) f(E(t)-E_{\rm r})}\right\}, \tag{13}$$

where $c_z = \frac{C_z(0,t)}{C_{z,\infty}}$, $z = \text{red}, \text{ox}$, The input signal of the model is $u(\omega,t) = E_0 + A cos\omega t$, where $E_0 = 0.107$. The amplitude $A = 0.0536$ and the angular frequency is $\omega \in [2\pi, 10\pi] rad/s$ which is considered as the parameter. The total number of degrees of freedom is 16,003 after discretization in space. There are three outputs: the current density $J(E, u(\omega,t))$, the concentration of the oxidant $C_{\rm ox}$ and the concentration of reductant $C_{\rm red}$. We only present the results of LSTM with modified structure (3) in figures. The errors of LSTM with standard structure (2) are listed in Table VII, which are sufficient to show its performance.

1) Results of LSTM on case 1: For the non-parametric case, the input signal is taken as $E(\omega,t) = E_0 + A cos 2\pi t$, with a fixed $\omega = 2\pi$. The training data are 400 snapshots of outputs in the time interval $[0,4]s$. The time window size is 200, meaning at the online prediction stage, 200 snapshots in the time interval $[0,2]s$ are used to make LSTM proceed. Fig. 7 presents outputs obtained from DNS and that computed by LSTM in the testing time interval $[2,20]s$. The LSTM predictions match the reference solutions for all the outputs very well. The relative errors for all outputs listed in Table VI are also very small.

2) Results of LSTM on case 2: For the parametric case, we take five samples of $\omega : \pi, 3\pi, 5\pi, 7\pi, 9\pi$ as the training parameter samples. The corresponding training data are the output snapshots at 400 time instances at those samples in the time intervals $[0,4]s$, $[0,1.33]s$, $[0,0.8]s$, $[0,0.57]s$, $[0.0.44]s$, respectively. Each training sample of ω corresponds to 400 training snapshots at 400 time instances in the corresponding training time intervals. Each training time interval includes the output waveform with 4 periods. Note that the period length of the outputs changes with the frequency ω, so that the corresponding training time interval also changes with ω if the same number of output snapshots are taken for different samples of ω. As a result, the output waveforms corresponding to different samples of ω have different frequencies and different magnitudes. The whole time interval corresponding to each parameter sample includes 20 periods of output waveforms. For low frequency ω, the corresponding output also evolves with low frequency, so that with the same 20 periods, its final simulation time interval is the longest. In contrast, the output corresponding to the highest input frequency ω changes with highest frequency so that its whole time interval including 20 periods, is the shortest.

For this problem, we aim at using LSTM to learn the future 16 periods of such output waveforms with *changing* frequencies and *changing* magnitudes in the corresponding *testing* time intervals. The time window size $w = 200$ is used in both the training stage and the online prediction stage, including 200 output snapshots in 2 periods of the output waveform. At the online stage, the trained LSTM is used to predict the output in the testing time intervals corresponding to 4 testing samples of $\omega : 2\pi, 4\pi, 6\pi, 8\pi$. The testing time intervals for the testing parameter samples are $[1,10]s$, $[0.5,5]s$, $[0.33,3.33]s$, $[0.25,2.5]s$, respectively.

The relative errors for each output and at each testing input of LSTM with modified structure (3) are listed in Table VI, where the maximal errors are all below or very close to 0.1 and the mean errors are even smaller. In contrast, the relative errors of LSTM with standard structure (2) in Table VII are

This article has been accepted for publication in IEEE Journal on Multiscale and Multiphysics Computational Techniques. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/JMMCT.2023.3242044

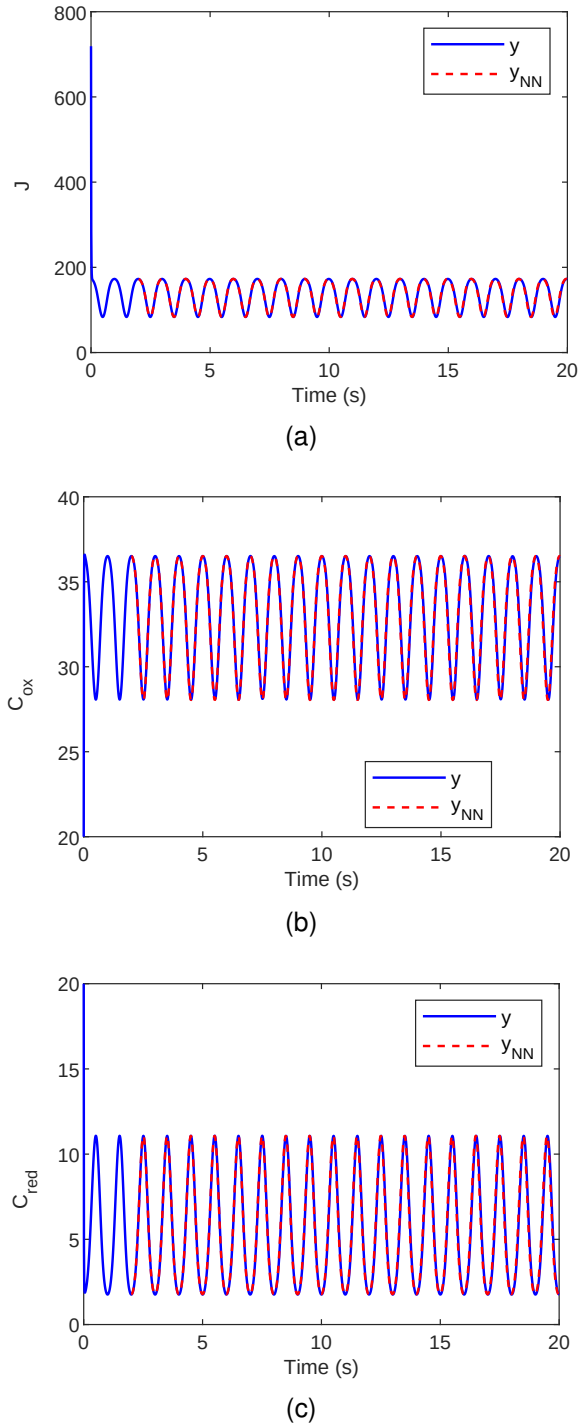

Fig. 7. Ferrocyanide oxidation reaction model with non-parametrized input $E(\omega, t) = Acos(2\pi t)$. (a) Output y from DNS and LSTM (3) prediction y_{NN} for the current density J. (b) Output y from DNS and LSTM prediction y_{NN} for the concentration $C_{\rm ox}$. (c) Output y from DNS and LSTM (3) prediction y_{NN} for the concentration $C_{\rm red}$.

unacceptable and are up to two orders of magnitude larger than those errors in Table VI. We need to point out that the results in this table are the best ones we can get, which indicate that the standard LSTM can hardly produce results with acceptable accuracy for this model.

To avoid redundancy, we plot in Fig. 8 only the results for the current density at two testing inputs. Here we omitted the initial transient behavior of the output, since the magnitude of the transient part is much larger than that of the periodic part, making the difference between y and y_{NN} hardly observable. The LSTM prediction is sufficiently accurate, though there is a bit larger error at the testing $\omega = 8\pi$ in the later time period. This example shows the capability of LSTM in learning multiple outputs at multiple parameter samples with different magnitudes and frequencies. The key of the success is the activation function tf.math.sin in the modified LSTM (3).

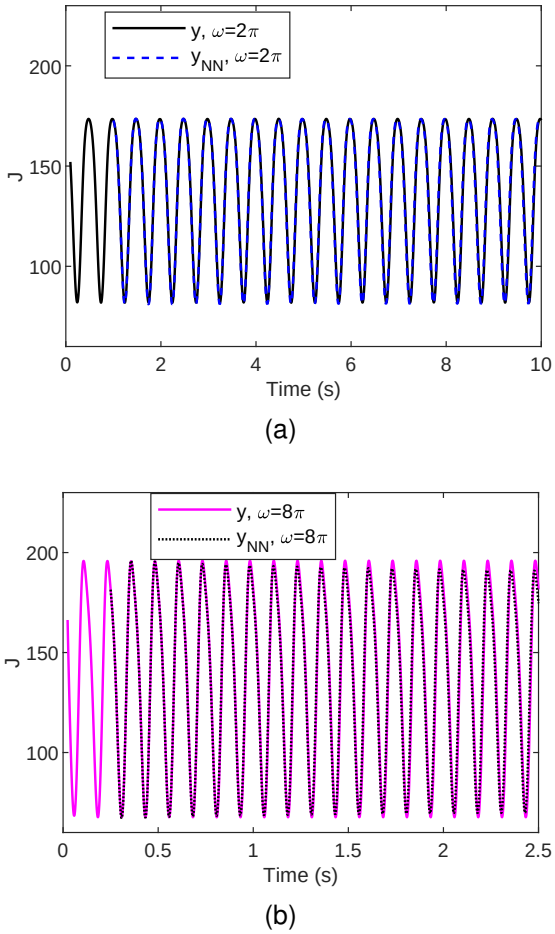

Fig. 8. Ferrocyanide oxidation reaction model with parametrized input $E(\omega, t) = Acos(\omega t)$. (a) Output y from DNS and LSTM (3) prediction y_{NN} for the current density J corresponding to the testing input $E(\omega, t)$, with $\omega = 2\pi$. (b) Output y from DNS and LSTM (3) prediction y_{NN} for J corresponding to the testing input with $\omega = 8\pi$.

D. Runtime Comparison

This section considers the runtime of training LSTM and the runtime of online LSTM prediction. We also compare the computational costs of LSTM to those of DNS. As

TABLE VI
ERROR OF LSTM (3) FOR THE FERROCYANIDE OXIDATION REACTION MODEL, TRAINING $\omega : \pi, 3\pi, 5\pi, 7\pi, 9\pi$.

Error	Case 1	Case 2 at $\omega : 2\pi, 4\pi, 6\pi, 8\pi$			
$\epsilon_{\texttt{mean}}$(J)	0.0116	0.0103	0.0146	0.0198	0.0236
$\epsilon_{\texttt{max}}$(J)	0.0416	0.0425	0.0613	0.0717	0.0983
$\epsilon_{\texttt{mean}}(C_{ox})$	0.0051	0.0175	0.0203	0.0138	0.0267
$\epsilon_{\texttt{max}}(C_{ox})$	0.0175	0.0658	0.0761	0.0559	0.1038
$\epsilon_{\texttt{mean}}(C_{red})$	0.0051	0.0048	0.0055	0.0036	0.0068
$\epsilon_{\texttt{max}}(C_{red})$	0.0661	0.0180	0.0206	0.0143	0.0267

mentioned before, using LSTM to predict the output directly is independent of the dimension of the original model. Therefore, the larger the original model is, the more benefit we have by using LSTM. Suppose we aim at computing the solution of the parametric nonlinear system at m samples of parameters. Without LSTM, we need to simulate the original system for m times in the whole time interval $[0, T]$. Using LSTM, we first divide the m samples into m_1 training and m_2 testing samples, where DNS needs to be done at those m_1 training samples in the training time interval $[0, T_0], T_0 < T$ to train LSTM. After LSTM is trained, DNS is only implemented at each testing parameter sample in the time window $[0, T_w], T_w < T_0 < T$, then LSTM is used to predict the solution at the testing time interval $[T_w, T]$. The total runtime of LSTM includes the offline time of training data generation ($t_{\texttt{offline}}$), the network training time ($t_{\texttt{train}}$), the online time of window data generation ($t_{\texttt{window}}$) corresponding to the testing parameters and the online time of prediction ($t_{\texttt{pred}}$). The LSTM network will outperform the DNS when $t_{\texttt{offline}} + t_{\texttt{train}} + t_{\texttt{window}} + t_{\texttt{pred}} \ll t_{\texttt{DNS}}$, where $t_{\texttt{DNS}}$ is the total time of DNS for m samples.

In Table VIII, we list the above times for all the three models for both the non-parametric case and the parametric case. Note that for the non-parametric case, the runtime of window data generation is actually zero, since the data in the time window are already included in the training data, so that they don't have to be generated again. However, for the parametric case, the data in the time window corresponding to the testing parameters are not available from the training data, so that they need to be computed online. We also present the time saved $t_{\texttt{saved}}$ by LSTM compared to DNS, i.e., $t_{\texttt{DNS}} - (t_{\texttt{offline}} + t_{\texttt{train}} + t_{\texttt{window}} + t_{\texttt{pred}})$. If we only compare the online time ($t_{\texttt{window}} + t_{\texttt{pred}}$) of LSTM to the DNS time $t_{\texttt{DNS}}$, then much more speedup can be achieved. The online speedup factor speedup$_{\texttt{online}}$ is shown in the last column of the table. To the best of the author's knowledge, complete runtime comparison is not yet available in the existing literature. Only partial information is presented, for example, only the training time of LSTM for a single epoch [7], or only the training time $t_{\texttt{train}}$ and predicting time of LSTM $t_{\texttt{pred}}$ [16].

Training LSTM and LSTM prediction are done on a Laptop with Intel (R) Core (TM) i7-5500U CPU @ 2.4GHz 2.39 GHz, 8 GB RAM. However, due to the very long time of running DNS on the Laptop, we move all computations involving DNS to a computer server, which is around 5 times faster than the Laptop. The configuration of the server includes two AMD EPYC 7763 64-core processors (each core has 32kB L1-cache, 512kB L2-cache, eight core share 32MB L3-cache) with hyper-threading; 1TB main memory split into two 512 GB. Each CPU socket controls one part (NUMA architecture); SSD RAID 1 with 1TB. We could also compare the runtime by running LSTM script on the same server. However, due to the small dimension of the output data, the training time on the server is not shorter than the training time on the Laptop. The training time of LSTM can be further reduced when we run the script on a cluster equipped with GPU. However, Tensorflow with GPU only supports LSTM with standard structure by using the $sigmoid$ and $tanh$ activation functions. Tensorflow with GPU will slow down to the CPU speed or even slower when the modified LSTM structure is trained. Since in this work, the modified LSTM produces much more accurate results than the standard LSTM for the third model, we stay with the Laptop for training LSTM and LSTM prediction.

E. Pros and Cons of LSTM

The results for the three examples show that LSTM predicts the outputs in the testing time interval and at the testing parameters sufficiently well. For non-parametric case, the training time of LSTM is short. For parametric case, the training time of LSTM depends on the number of training parameter samples which decides the dimension of the training data. When a large number of training parameter samples are needed, there is a large amount of training data, leading to longer training time. However, since the proposed LSTM learns only the output(s) of the system, the training time of LSTM is independent of the dimension (degrees of freedom) of the original model. The training time of LSTM is usually long, as can be seen from Table VIII. Therefore, LSTM has no advantage if it is used for predicting outputs of small models. However, for very large-scale systems taking long time of DNS, LSTM will outperform. This can also be seen from Table VIII. For problems with no mathematical models, but only with measured data, LSTM should be a choice for long-term prediction in time, especially for outputs with periodic waveforms or outputs that do not vary much over the time.

It is also noticed that when the output waveforms corresponding to different parameters are very different, LSTM will fail. For example, when the outputs are non-periodic at some parameter samples, but periodic at other parameter samples, LSTM cannot learn the parametric behavior of such outputs.

Similar to other machine learning techniques, some parameters for training the LSTM network need to be adjusted for more than one time and for different examples. For example, the number of hidden units, the time window size w, the number of epochs are the main factors that influence the accuracy of LSTM. Whereas, proper values of these parameters are not obtained purely with try-and-error. Some prior analysis or knowledge of the original model can help to decide the time window size. For the nonlinear circuit model, the output snapshots corresponding to the rise time of the output should be included into the time window and the training data. However, without simulating the original model, we never know the rise time of the output. One way of deciding the

TABLE VII
ERROR OF LSTM (2) FOR CASE 1 OF THE FERROCYANIDE OXIDATION REACTION MODEL.

$\epsilon_{\texttt{mean}}(\text{J})$	$\epsilon_{\texttt{max}}(\text{J})$	$\epsilon_{\texttt{mean}}(C_{ox})$	$\epsilon_{\texttt{max}}(C_{ox})$	$\epsilon_{\texttt{mean}}(C_{red})$	$\epsilon_{\texttt{max}}(C_{red})$
0.1482	0.4931	0.0665	0.2227	0.2414	0.8088

TABLE VIII
RUNTIME (SECONDS) COMPARISON BETWEEN LSTM AND DNS.

Model	Cases	$t_{\texttt{offline}}$	$t_{\texttt{train}}$	$t_{\texttt{window}}$	$t_{\texttt{pred}}$	$t_{\texttt{DNS}}$	$t_{\texttt{saved}}$	$\text{speedup}_{\texttt{online}}$
Circuit (step)	Case 1	1,131	318	0	27	2,596	1,120	96
Circuit (step)	Case 2	5,672	1,422	1,993	106	22,424	13,231	11
Circuit (exponential)	Case 1	1,640	665	0	23	2,652	324	115
Circuit (exponential)	Case 2	8,183	6,951	4,523	87	25,442	5,698	6
FitzHugh, LSTM(3)	Case 1	1,262	405	0	23	2,524	834	110
FitzHugh, LSTM (3)	Case 2	5,990	3994	630	70	18,378	7,694	26
FitzHugh, LSTM (2)	Case 1	1,262	405	0	26	2,524	831	97
FitzHugh, LSTM (2)	Case 2	5,990	2,644	630	74	18,378	9,040	26
Electrochem.	Case 1	396	1,131	0	122	4,252	2,603	35
Electrochem.	Case 2	2,468	7,784	845	537	19,021	7,387	14

time window size of problems with available mathematical models is to simulate the numerically discretized model on a rough space grid with much fewer degrees of freedom. Then use the output response of the small model to estimate the time window size. While the time window size could be determined using some techniques, the number of epochs is taken more or less with try-and-error. Usually, the number of epochs is initially taken conversely large. Then a more proper number could be observed from the values of the loss function (training error) after each epoch run. If the training error is already very small at an early stage, then the number of epochs can be further reduced; otherwise, it should be further increased. Moreover, the training error may oscillate, a more reliable number of epochs should be the one at which the loss function value is smaller and the loss function values before it have already exhibited a decreasing trend for quite a few epochs. In summary, how to decide proper values of epochs and number of hidden units is still an open issue not only for LSTM but also for other neural networks in machine learning.

V. CONCLUSIONS

In this work, we present the performance of the LSTM network on predicting output dynamics of systems with time-varying inputs and systems with parametrized time-varying inputs. For both cases, the LSTM network produces predictions with acceptable accuracy. All the results show the potential of LSTM in learning output dynamics purely based on output training data. Furthermore, it also shows that LSTM is promising for directly learning low dimensional dynamics, e.g., dynamics of the outputs, based on measured data. Certainly, many mathematical analysis are still missing for optimal performance of LSTM w.r.t. the time window size and other hyper-parameters of the network, e.g., the hidden units, the number of epochs, etc.

ACKNOWLEDGMENTS

We thank Prof. Tanja Vidaković-Koc and Ms. Tamara Milicic from Max Planck Institute for Dynamics of Complex Technical Systems, Germany for providing us with the Ferrocyanide oxidation reaction model.

REFERENCES

[1] A. Bērziņš, J. Helmig, F. Key, and S. Elgeti, "Standardized non-intrusive reduced order modeling using different regression models with application to complex flow problems," arXiv, e-prints:2006.13706v1, 2020.

[2] K. Bhattacharya, B. Hosseini, N. B. Kovachki, and A. M. Stuart, "Model reduction and neural networks for parametric PDEs," arXiv, e-prints:2005.03180v1, 2020.

[3] S. Fresca, L. Dedè, and A. Manzoni, "A comprehensive deep learning-based approach to reduced order modeling of nonlinear time-dependent parametrized PDEs," *J Sci. Comput.*, vol. 87, p. 61, 2021.

[4] F. J. Gonzalez and M. Balajewicz, "Deep convolutional recurrent autoencoders for learning low-dimensional feature dynamics of fluid systems," arXiv, e-prints:1808.01346v2, 2018.

[5] M. Guo and J. S. Hesthaven, "Data-driven reduced order modeling for time-dependent problems," *Computer Methods in Applied Mechanics and Engineering*, vol. 345, pp. 75–99, 2019.

[6] A. Mohan and D. V. Gaitonde, "A deep learning based approach to reduced order modeling for turbulent flow control using LSTM neural networks," arXiv, e-prints:1804.0926, 2018.

[7] S. M. Rahman, S. Pawar, O. San, A. Rasheed, and T. Iliescu, "Nonintrusive reduced order modeling framework for quasigeostrophic turbulence," *Phys. Rev. E*, vol. 100, p. 053306, 2019.

[8] M. Raissi, P. Perdikaris, and G. E. Karniadakis, "hysics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations," *J. Comput. Phys.*, vol. 378, no. 1, pp. 686–707, 2019.

[9] R.Maulika, B. Luscha, and P. Balaprakashb, "Reduced-order modeling of advection-dominated systems with recurrent neural networks and convolutional autoencoders," *Physics of Fluids*, vol. 33, no. 037106, 2021.

[10] P. Benner, S. Grivet-Talocia, A. Quarteroni, G. Rozza, W. Schilders, and L. M. Silveira, Eds., *Model Order Reduction, Volume 1: System- and Data-Driven Methods and Algorithms*. De Gruyter, 2021. [Online]. Available: https://www.degruyter.com/document/isbn/9783110498967/html

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

[12] S. Otto and C. Rowley, "Linearly-recurrent autoencoder networks for learning dynamics," arXiv, e-prints:1712.01378v2, 2017.

[13] J. N. Kani and A. H. Elsheikh, "Reduced-order modeling of subsurface multi-phase flow models using deep residual recurrent neural networks," *Transport in Porous Media*, vol. 126, pp. 713–741, 2018.

[14] Z. Li, N. Kovachki, K. Azizzadenesheli, B. Liu, K. Bhattacharya, A. Stuart, and A. Anandkuma, "Fourier neural operator for parametric partial differential equations," in *Proc. ICLR 2021*, 2021.

[15] S. B. Reddy, A. R. Magee, R. K. Jaiman, J. Liu, W. Xu, A. Choudhary, and A. A. Hussain, "Reduced order model for unsteady fluid flows via recurrent neural networks," in *Proc. ASME 2019 38th International Conference on Ocean, Offshore and Arctic Engineering*, 2019.

[16] M. M. A., S. A. Sadrossadat, and V. Derhami, "Long short-term memory neural networks for modeling nonlinear electronic components," *IEEE Trans. Compon. Packag. Technol.*, vol. 11, no. 5, pp. 840–847, 2021.
[17] J. Hochreiter, "Untersuchungen zu dynamischen neuronalen netzen," Master's thesis, Institut für Informatik Technische Universität München, Munich, Germany, 1991.
[18] F. Gers, "Long Short-Term Memory in recurrent neural networks," Master's thesis, Universität Hannover, Hannover, Germany, 2001.
[19] S. Hochreiter and J. Schmidhuber, "Long Short-Term Memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 1997.
[20] K. Greff, R. K. Srivastava, J. Koutník, B. R. Steunebrink, and J. Schmidhuber, "Lstm: A search space odyssey," *Ttransactions on Neural Networks and Learning Systems*, vol. 28, no. 10, pp. 2222–2232, 2017.
[21] A.Summerville, S. Snodgrass, M. Guzdial, C. Holmgard, A. K. Hoover, A. Isaksen, A. Nealen, and J. Togelius, "Procedural content generation via machine learning (pcgml)," *IEEE Transactions on Games*, vol. 10, no. 3, pp. 257–270, 2018.
[22] C. Olah, "Understanding LSTM networks," Tech. Rep., 2015. [Online]. Available: https://colah.github.io/posts/2015-08-Understanding-LSTMs/
[23] Y. Chen, "Model order reduction for nonlinear systems," Master's thesis, Massachusetts Institute of Technology, Cambrigde, Massachusetts, 1999.
[24] R. FitzHugh, "Impulses and physiological states in theoretical models of nerve membrane," *Biophysical Journal*, vol. 1, no. 6, pp. 445–466, 1961.
[25] The MORwiki Community, "Fitzhugh-nagumo system," MORwiki – Model Order Reduction Wiki, 2018. [Online]. Available: http://modelreduction.org/index.php/FitzHugh-Nagumo_System

Lihong Feng received the Ph.D. degree in computational mathematics from Fudan University, China, in 2002. She did post-doctoral research at Fudan University in 2003-2004 and became a Lecturer at Fudan University in 2005-2006. She worked as a Humboldt Fellow at TU Chemnitz, Germany in 2007-2008. She was a research fellow at Freiburg University, Germany in 2009-2010. Since 2010, she has been a Senior Scientist with the Max Planck Institute for Dynamics of Complex Technical Systems, Germany. Her research interests include model-order reduction and fast simulation of complex models arising from engineering applications, scientific machine learning, numerical analysis, scientific computing, etc.