Bulletin of Mathematical Biology (2021) 83:19
https://doi.org/10.1007/s11538-020-00851-7

Society for Mathematical Biology

ORIGINAL ARTICLE

Deep Learning of Biological Models from Data: Applications to ODE Models

Wei-Hung Su[1] · Ching-Shan Chou[1] · Dongbin Xiu[1]

Received: 6 July 2020 / Accepted: 21 December 2020 / Published online: 16 January 2021

Abstract

Mathematical equations are often used to model biological processes. However, for many systems, determining analytically the underlying equations is highly challenging due to the complexity and unknown factors involved in the biological processes. In this work, we present a numerical procedure to discover dynamical physical laws behind biological data. The method utilizes deep learning methods based on neural networks, particularly residual networks. It is also based on recently developed mathematical tools of flow-map learning for dynamical systems. We demonstrate that with the proposed method, one can accurately construct numerical biological models for unknown governing equations behind measurement data. Moreover, the deep learning model can also incorporate unknown parameters in the biological process. A successfully trained deep neural network model can then be used as a predictive tool to produce system predictions of different settings and allows one to conduct detailed analysis of the underlying biological process. In this paper, we use three biological models—SEIR model, Morris–Lecar model and the Hodgkin–Huxley model—to show the capability of our proposed method.

Keywords Deep neural network · Residual network · Mathematical biology · Governing equation discovery

✉ Dongbin Xiu
xiu.16@osu.edu

Wei-Hung Su
su.677@osu.edu

Ching-Shan Chou
chou.160@osu.edu

[1] Department of Mathematics, The Ohio State University, Columbus, OH 43221, USA

1 Introduction

One of the main goals in modeling biological systems is to reveal the mechanisms underlying complex behaviors of various biological processes. For example, one can often describe intracellular or intercellular networks using mathematical equations. In particular, ordinary differential equation (ODE) models have been used to describe such systems. However, finding the correct governing equations is often challenging for several reasons: The complete picture of biological network is only partly known, that is, some information of the network is missing; the parameters such as kinetic rates are not available or experimentally measurable; the parameters are often time dependent but assumed to be constant (for ease of modeling), which leads to biased models and questionable conclusion (Ye et al. 2015). Moreover, in biological systems, many interactions are highly nonlinear and complex. It is difficult to determine the correct nonlinear terms in the model (DeAngelis and Yurek 2015), and nonlinearity-induced sensitivity to parameters can greatly lower the predictive power of the model (Wood and Thomas 1999; Yodzis 1988; Goodfellow et al. 2016). Those factors make determination of high fidelity governing equations and models extremely difficult in many cases. And misspecified and low-fidelity mechanistic models often have poor performance on explanatory or predictive power (Wood and Thomas 1999; Perretti et al. 2013). Therefore, one may question whether it is possible to use only observational data to numerically determine the underlying mechanisms without resorting to explicit analytical derivation of equation-based models.

Recently, there has been a growing amount of research in numerical modeling of unknown governing equations using observational data. Some notable efforts include symbolic regression (Bongard and Lipson 2007; Schmidt and Lipson 2009), equation-free modeling (Kevrekidis et al. 2003), heterogeneous multiscale method (HMM) (Weinan et al. 2003), artificial neural networks (Gonzalez-Garcia et al. 1998), nonlinear regression (Voss et al. 1999), empirical dynamic modeling (Sugihara et al. 2012), nonlinear Laplacian spectral analysis (Giannakis and Majda 2012) and automated inference of dynamics (Daniels and Nemenman 2015a, b). Among those, artificial neural network (ANN), and particularly deep neural network (DNN), has seen tremendous successes in many different disciplines, particularly in recent few years. The number of publications is too large to mention. Here, we cite only a few relatively more recent review/summary-type publications (Montufar et al. 2014; Bianchini and Scarselli 2014; Eldan and Shamir 2016; Poggio et al. 2017; Du and Swamy 2014; Goodfellow et al. 2016; Schmidhuber 2015). Efforts have been devoted to the use of DNN for various aspects of scientific computing, including construction of reduced order model (Hesthaven and Ubbiali 2018), aiding solution of conservation laws (Ray and Hesthaven 2018), multiscale problems (Chan and Elsheikh 2018; Wang et al. 2018), solving and learning systems involving ODEs and PDEs (Mardt et al. 2018; Chen et al. 2018; Long et al. 2018; Khoo et al. 2018), uncertainty quantification (Tripathy and Bilionis 2018; Zhu and Zabaras 2018), and more.

Some efforts also have been made in biology areas to find governing equations by data-driven approaches, such as equation-free method through empirical dynamic modeling for marine ecosystem (Ye et al. 2015), automated inference of dynamics with symbolic regression for metabolic network (Schmidt et al. 2011), sparse identification

of nonlinear dynamic for a biological structure (Mangan et al. 2016), and Genmodel learning system for unknown qualitative simulation (QSIM) in physiological study (Hau and Coiera 1995). The use of DNN for model discovery has not, however, been investigated systematically in biological modeling.

The focus of this paper is on how to use DNN to learn underlying dynamical system of biological processes. First, we extend the method of Qin et al. (2019) to incorporate additional variables into the DNN model. Some of the variables can be controlled during experimentation such as temperature or electronic current. Their incorporation results in a more flexible neural network model for the underlying process and allows us to investigate the influence of the controlled variables on the dynamics. We then employ three classical biological models to demonstrate the deep learning model discovery procedure and examine its effectiveness. The first example is the SEIR model (Hethcote 2000), one of the common models in the epidemiology. It is used for modeling the dynamics of infectious disease such as chickenpox, rubella, mumps and malaria. The second example is the Morris–Lecar model (Morris and Lecar 1981), one of the simplest ion channel models. This model was developed to understand the relation between the applied current, which is a controlled variable, and the neuronal excitability in barnacle muscle fibers. This model is particularly interesting because it is known to have rich dynamical behaviors such as bifurcations. The last example is Hodgkin–Huxley model (Hodgkin and Huxley 1952), another ion channel model. The dynamical system features a spike structure, which makes numerical modeling particularly difficult. We use these three examples to demonstrate the capability of our numerical approach. In all examples, our method produces a neural network model for the underlying system using only measurement data. The constructed neural network model is then used to conduct system analysis and simulations, and its results are compared against the reference results generated by the known true models.

The structure of this paper is as follows: In Sect. 2, we set up the general framework and notations, along with an overview of data requirement and the basics of neural networks. In Sect. 3, we discuss the technical details of DNN learning for governing equations. Then in Sect. 4, we apply the learning method to the three biological models to examine the performance and properties of our method. Summary and discussions are in Sect. 5.

2 Setup

We assume that the biological process under consideration can be modeled by an autonomous dynamical system in the following general form:

$$\frac{d}{\mathrm{d}t}\mathbf{x}(t;\boldsymbol{\alpha}) = \mathbf{f}(\mathbf{x},\boldsymbol{\alpha}), \qquad \mathbf{x}(0) = \mathbf{x}_0, \tag{1}$$

where $\mathbf{x} = (x_1, \ldots, x_d)$, $d \geq 1$, represents the time-dependent state variables and $\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_\ell)$ are a set of parameters associated with the model. We assume that the model, namely the governing Eq. (1), is unknown. However, we assume that we have access to measurement data of the state variables at various time instants. Our goal

is to construct a numerical model to approximate the unknown governing equations. We will then use the numerical model to conduct system analysis. Note that the model (1) may also contain certain algebraic constraints. Here we use a general description of ODEs to emphasize that the model is a dynamical system.

2.1 Data

We assume measurement data of the state variables $\mathbf{x}$ are available. Since different initial conditions and parameters lead to different trajectories in the phase space, we assume that these data are measured along various trajectories with various lengths in time. The information about the corresponding initial conditions is not available, as is the case in practice, and is not needed for our learning method.

Let N_T be the total number of trajectories on which data are measured. Consider the ith trajectory, where $i = 1, \ldots, N_T$. Let $(K^{(i)} + 1)$ be the number of temporal data entries on the ith trajectory. We collect the data into the following set:

$$\mathbf{X}^{(i)} = \left\{ \mathbf{x}\left(t_k^{(i)}; \boldsymbol{\alpha}^{(i)}, \mathbf{x}_0^{(i)}\right) \right\}, \qquad k = 0, \ldots, K^{(i)}.$$

Here, $\{t_k^{(i)}, k = 0, \ldots, K^{(i)}\}$ are the time instants of the data entries, $\boldsymbol{\alpha}^{(i)}$ are system parameter values associated with this trajectory, and $\mathbf{x}_0^{(i)}$ is the initial condition of the trajectory. We emphasize here that both $\boldsymbol{\alpha}^{(i)}$ and $\mathbf{x}_0^{(i)}$ are treated as unknowns. We then reorganize the data into a set of pairs of adjacent time instants,

$$\left\{ \mathbf{x}\left(t_k^{(i)}; \boldsymbol{\alpha}^{(i)}, \mathbf{x}_0^{(i)}\right), \quad \mathbf{x}\left(t_{k+1}^{(i)}; \boldsymbol{\alpha}^{(i)}, \mathbf{x}_0^{(i)}\right) \right\}, \qquad k = 0, \ldots, K^{(i)} - 1.$$

Each of these pairs can be considered as a trajectory of length two, with the first entry serving as the "initial condition" and the second entry as the "end condition." Since we assume the true model (1) is autonomous, time variable can be arbitrarily shifted. We rewrite the data entry pairing as

$$\left\{ \mathbf{x}\left(0; \boldsymbol{\alpha}^{(i)}\right), \quad \mathbf{x}\left(\Delta_k^{(i)}; \boldsymbol{\alpha}^{(i)}\right) \right\}, \quad \Delta_k^{(i)} = t_{k+1}^{(i)} - t_k^{(i)}, \quad k = 0, \ldots, K^{(i)} - 1, \quad (2)$$

where $\Delta_k^{(i)}$ is the time difference between the two data entries. The dependence on the initial condition is suppressed, as the first data entry serves as the initial condition for this two-entry short trajectory.

Finally, we collect the data pairs from all trajectory measurements $i = 1, \ldots, N_T$, and re-label them using a single index to simplify notation. By also taking into account the possible measurement noises in the data, the entire dataset can be written as

$$S = \left\{ \mathbf{z}_j^{(1)}, \quad \mathbf{z}_j^{(2)} \right\}, \qquad j = 1, \ldots, J, \quad (3)$$

with the total number of the pairs $J = K^{(1)} + \cdots + K^{(N_T)}$ and

$$\mathbf{z}_j^{(1)} = \mathbf{x}\left(0; \boldsymbol{\alpha}^{(i_j)}\right) + \epsilon_j^{(1)}, \qquad \mathbf{z}_j^{(2)} = \mathbf{x}\left(\Delta_j; \boldsymbol{\alpha}^{(i_j)}\right) + \epsilon_j^{(2)}, \tag{4}$$

where $\epsilon_j^{(1)}$ and $\epsilon_j^{(2)}$ are noises/errors in the state variable data. That is, each jth pair, $j = 1, \ldots, J$, consists of data of the state variables $\mathbf{x}$ separated by the time difference Δ_j. Also, each jth pair corresponds uniquely to an underlying i_jth trajectory in the original data pairings (2). For notational convenience, throughout this paper we assume $\Delta_j = \Delta$ as a constant.

2.2 Deep Neural Networks

Following the recent work of Qin et al. (2019), we adopt deep neural network (DNN) as the primary modeling method for recovering unknown governing equations. In particular, we employ feedforward neural network (FNN) as the core building block. A standard FNN defines a nonlinear map as follows.

Let $\mathbf{N} : \mathbb{R}^m \to \mathbb{R}^n$ be the operator associated with a FNN with M hidden layers ($M \geq 1$). Let $\mathbf{y}^{\text{in}} \in \mathbb{R}^m$ be the input and $\mathbf{y}^{\text{out}} \in \mathbb{R}^n$ the corresponding output. The DNN mapping can be written as

$$\mathbf{y}^{\text{out}} = \mathbf{N}(\mathbf{y}^{\text{in}}; \Theta) = \mathbf{W}_{M+1} \circ (\sigma_{\text{M}} \circ \mathbf{W}_M) \circ \cdots \circ (\sigma_1 \circ \mathbf{W}_1)(\mathbf{y}^{\text{in}}), \tag{5}$$

where $\mathbf{W}_m$ is the weight matrix between the mth and the $(m+1)$th layers, $\sigma_m : \mathbb{R} \to \mathbb{R}$ is the activation function of the neurons in the mth layer, and $\circ$ stands for operator composition. Following the neural network nomenclature tradition, we have incorporated the network biases into the weight matrices, and applied activation functions in a component-wise manner. We shall use Θ to denote all the parameters in the network.

When the input and output dimensions are identical, i.e., $m = n$, "residual network" (ResNet) He et al. (2016) can be readily defined as

$$\mathbf{y}^{\text{out}} = [\mathbf{I}_m + \mathbf{N}(\cdot; \Theta)]\,(\mathbf{y}^{\text{in}}), \tag{6}$$

where $\mathbf{I}_m$ is the identity matrix of size $m \times m$. In this form, the neural network in fact models the difference between the input and output (thus the term "residual"). Although mathematically equivalent to the standard DNN, ResNet has been shown to be exceptionally useful in practice after its introduction in He et al. (2016). We will adopt the ResNet idea and modify it to our modeling work in the following section.

3 Deep Learning Method Description

In this section we describe the details of the deep learning method. The method is based on ResNet learning of general dynamical system (Qin et al. 2019). Our current

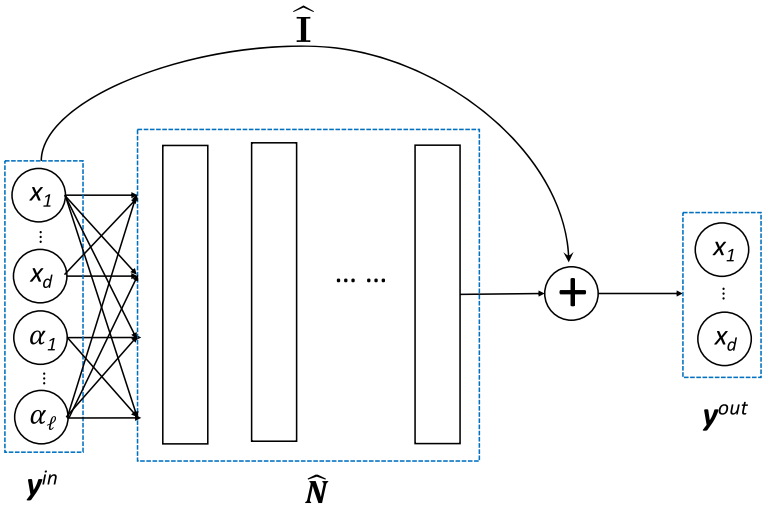


Fig. 1 (Color figure online) Structure of the neural network

approach extends the method from Qin et al. (2019) by incorporating unknown system parameters in the deep learning model.

3.1 Neural Network Structure

Straightforward application of ResNet, as proposed in Qin et al. (2019), is only applicable for learning equations with fixed parameters $\boldsymbol{\alpha}$. To model unknown system (1) with unknown parameters, we employ a modified ResNet structure.

The network structure is illustrated in Fig. 1. The input consists of the state variable $\mathbf{x}^{\text{in}}$, along with the system parameters $\boldsymbol{\alpha}$, i.e., $\mathbf{y}^{\text{in}} = [\mathbf{x}^{\text{in}}; \boldsymbol{\alpha}]$. Therefore, the input layer consists of $(d + \ell)$ neurons.

Multiple fully connected feedforward hidden layers follow the input layer. The width and depth of the hidden layers determine the complexity and modeling capability of the network. The output layer consists of only the state variables $\mathbf{x} \in \mathbb{R}^d$, i.e., $\mathbf{y}^{\text{out}} = \mathbf{x}^{\text{out}}$. An operator $\widehat{\mathbf{I}}$ is employed to re-introduce the input state variable $\mathbf{x}^{\text{in}}$ before the output layer. It is defined as a $d \times (d + \ell)$ matrix

$$\widehat{\mathbf{I}} = \begin{bmatrix} \mathbf{I}_d & \mathbf{0} \end{bmatrix}, \tag{7}$$

where $\mathbf{I}_d$ is the identity matrix of size $d \times d$ and $\mathbf{0}$ is a zero matrix of size $d \times \ell$. Let $\mathbf{N} : \mathbb{R}^{d+\ell} \to \mathbb{R}^d$ be the mapping operator defined by our network structure. The introduction of the operator $\widehat{\mathbf{I}}$ effectively produces the following mapping

$$\begin{aligned} \mathbf{y}^{\text{out}} &= \mathbf{N}(\mathbf{y}^{\text{in}}; \Theta) = \left[\widehat{\mathbf{I}} + \widehat{\mathbf{N}}\right]\left(\mathbf{y}^{\text{in}}\right) \\ &= \mathbf{x}^{\text{in}} + \widehat{\mathbf{N}}(\mathbf{x}^{\text{in}}, \boldsymbol{\alpha}; \Theta), \end{aligned} \tag{8}$$

where $\widehat{\mathbf{N}} : \mathbb{R}^{d+\ell} \to \mathbb{R}^d$ is the mapping operator defined by the hidden layers, and Θ is the parameter set associated with the network.

3.2 Network Training and Prediction

To train the network (8), we use the dataset (3), which contains pairs of state variables $\left\{\mathbf{z}^{(1)}, \mathbf{z}^{(2)}\right\}$ at two adjacent time instants separated by time step Δ. For each pair, we use the first data entry $\mathbf{z}^{(1)}$ as $\mathbf{x}^{\text{in}}$ in the network input $\mathbf{y}^{\text{in}}$. That is, $\mathbf{y}_j^{\text{in}} = [\mathbf{z}_j^{(1)}; \boldsymbol{\alpha}_j]$ is the jth input of the DNN model, for all $j = 1, \ldots, J$. The second data entry $\mathbf{z}_j^{(2)}$ is used to compute the cost of the network output. Our loss function is defined as the mean squared loss,

$$L(\Theta) = \frac{1}{J} \sum_{j=1}^{J} \left\| \mathbf{z}_j^{(2)} - \mathbf{N}(\mathbf{y}_j^{\text{in}}; \Theta) \right\|_2^2, \tag{9}$$

where $\|\cdot\|_2$ denotes vector-2 norm. Upon minimizing the loss function, we obtain a set of trained parameter Θ^*. This in turn defines our trained network model $\mathbf{N}(\mathbf{y}^{\text{in}}; \Theta^*)$. Upon iterative use of the trained network model (8), we can conduct system prediction. For any given new initial condition $\mathbf{x}_0$ at t_0 and system parameter $\boldsymbol{\alpha}$, we have

$$\begin{cases} \mathbf{x}(t_0) = \mathbf{x}_0, \\ \mathbf{x}(t_{n+1}) = \mathbf{x}(t_n) + \widehat{\mathbf{N}}(\mathbf{x}(t_n), \boldsymbol{\alpha}; \Theta^*), \quad n = 0, 1, 2, \ldots \\ t_{n+1} = t_n + \Delta. \end{cases} \tag{10}$$

This predictive model allows us to conduct system prediction for a new initial condition and with a new set of system parameters $\boldsymbol{\alpha}$.

4 Deep Learning of Biology Models

In this section we present three major test cases to illustrate the effectiveness of our deep learning method for biological models. Since the purpose is to validate the learning method, we use synthetic data generated from the known biological models. We first use the synthetic data to train neural network models. Once the models are trained satisfactorily, the DNN models are then used as a predictive tool to conduct numerical prediction of the underlying biological systems under new initial states and system parameter values that are not in the training data. The DNN model predictions are then compared against the reference solutions generated by the corresponding true mathematical models. In another word, the knowledge of the true models (which is not available in practical situation) is only used to generate the synthetic data for the DNN model training and to validate the prediction results produced by the DNN models.

4.1 Data Preparation and Network Training

Our training dataset follows the form of (3) and (4). In order to produce the synthetic training data, the following procedure is adopted:

- Choose domain-of-interest $I_{\mathbf{x}}$ and $I_{\boldsymbol{\alpha}}$ for the state variable $\mathbf{x}$ and system parameter $\boldsymbol{\alpha}$, respectively. These are the regions where we are interested in the dynamical behavior of the underlying system. This choice is obviously problem dependent. (Although ideally it is desirable to understand the system dynamics for all values of $\mathbf{x}$ and $\boldsymbol{\alpha}$, in practice one often needs to confine the study to a more localized region. This is common for numerical modeling and simulation.)
- Once the domain-of-interest $I_{\mathbf{x}}$ and $I_{\boldsymbol{\alpha}}$ are determined, we randomly generate samples in the domains. For bounded domain, we employ a uniform distribution; and for unbounded domain we employ a Gaussian distribution. This sampling procedure generates a set of samples $(\mathbf{x}^{(j)}, \boldsymbol{\alpha}^{(j)})$, $j = 1, \ldots, J$, where total J the number of samples. These samples serve as the "initial states" $\mathbf{x}(0; \boldsymbol{\alpha}^{(j)})$ of our trajectory data.
- For each initial state $\mathbf{x}(0; \boldsymbol{\alpha}^{(j)})$, $j = 1, \ldots, J$, we collect its trajectory data of the underlying system. In this paper, the trajectory data are synthetic data generated by solving the underlying true systems via the high-resolution solver, LSODE Radhakrishnan and Hindmarsh (1993). The solver was executed by a short time Δ to generate the solution $\mathbf{x}(\Delta; \boldsymbol{\alpha}^{(j)})$, which is the "end state" of the jth trajectory. The initial state and end state solutions then form the jth data pair in our training dataset (3), after performing the standard data normalization procedure. Note that our training data are essentially a large number of short trajectory data. This follows the mathematical analysis of Wu and Xiu (2019), where it was shown a large number of short trajectories can greatly facilitate the discovery of the underlying equations.

We remark that $\boldsymbol{\alpha}$ represents not only the system parameters of (1), it can also represent the external inputs or control variables. For example, in our ion channel example below, $\boldsymbol{\alpha}$ is the electric current which can be controlled by experimentalists.

In all the examples in this paper, our DNN models consist of 3 hidden layers, each of which with 45 neurons. The number of neurons in the input layer equals the number of the state variables $\mathbf{x}$ plus the number of system parameters and control variables $\boldsymbol{\alpha}$. The number of neurons in the output layer is the same as the number of the state variables $\mathbf{x}$. Networks with different depth and width have been investigated. Our current choice represents a good balance between model accuracy and computational cost. All neural network models were trained in mini-batch by Adam's algorithm (Kingma and Ba 2017) in the open-source Tensorflow (Abadi et al. 2015) and Keras (Chollet et al. 2015) library. Loss function values over the training datasets are monitored during training. The network training is considered satisfactory when the loss saturates or reaches a sufficiently small level (problem dependent).

4.2 SEIR Model

The SEIR model is one of the most common epidemiology models for infectious disease such as chickenpox, rubella, mumps, and malaria. The system describes the dynamics of health among susceptible (S), exposed (E), infected (I), and resistant (R) populations. While there are variations such as SEI, SIR and SEIRS models, we mainly focus on the SEIR model here. For more details about the model, see Hethcote (2000).

The SEIR model describes the dynamical behavior of the amount of population in the four states variables, S, E, I and R:

$$\begin{cases} \frac{\mathrm{d}S}{\mathrm{d}t} = \mu(N - S) - \beta\frac{SI}{N} - \nu S, \\ \frac{\mathrm{d}E}{\mathrm{d}t} = \beta\frac{SI}{N} - (\mu + \sigma)E, \\ \frac{\mathrm{d}I}{\mathrm{d}t} = \sigma E - (\mu + \gamma)I, \\ \frac{\mathrm{d}R}{\mathrm{d}t} = \gamma I - \mu R + \nu S, \end{cases} \tag{11}$$

where $N = S + E + I + R$ is the total population, and $\boldsymbol{\alpha} = (\beta, \gamma, \sigma, \mu, \nu)$ are the system parameters.

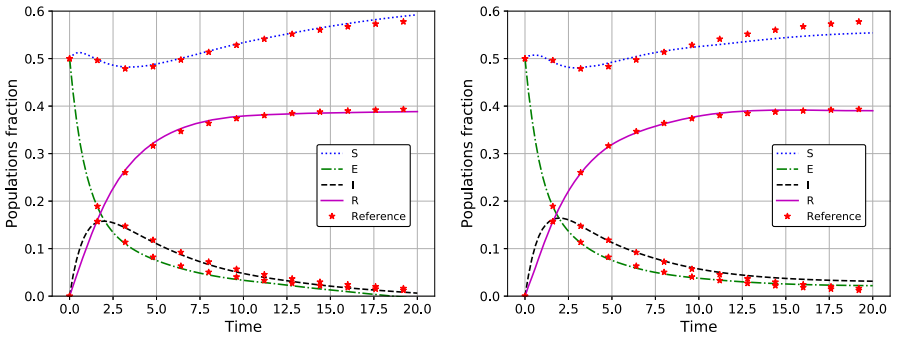
It is easy to observe that N is a constant. Therefore, we divide both sides of Eq. (11) by N and subsequently consider all the state variables as fractions of the total population. That is, $\mathbf{x} = (S, E, I, R) \in [0, 1]^4$ with $S + E + I + R = 1$.

To generate the synthetic data for our model learning, we chose, rather arbitrarily, $(\bar{\beta}, \bar{\gamma}, \bar{\sigma}, \bar{\mu}, \bar{\nu}) = (0.9, 0.2, 0.5, 0.3, 0.2)$ as the mean value of the system parameters and add $\pm 10\%$ perturbations. Therefore, the parameter domain is a hypercube and we use a uniform distribution to sample the domain of interest and utilize the true model (11) to generate the training data. Different levels of noise following Gaussian distribution were added to the data to simulate measurement noises. Two cases are presented here: one with 1% noise level, where our training dataset consists of 10,000 data pairs, and the other with 5% noise level, where the training dataset consists of 30,000 data pairs. The data pairs in the dataset are separated by $\Delta = 0.2$.

Once the neural network model is trained, we conduct system analysis using the DNN model and compare its results against the reference solution from the true model (11). In Fig. 2, we present time evolution of the state variables predicted by the neural network model, using an initial condition $\mathbf{x}_0 = (0.5, 0.5, 0, 0)$ and system parameter $(\beta, \gamma, \sigma, \mu, \nu) = (0.9, 0.2, 0.5, 0.3, 0.2)$. Very good agreement with the reference solution from the true model can be observed, indicating sufficient accuracy in the predictive capability of the DNN model. Results for other initial conditions and system parameter values are of similar quality and thus not shown. In Fig. 3, we show the training loss history during the network training, for both noisy data case and noiseless data case.

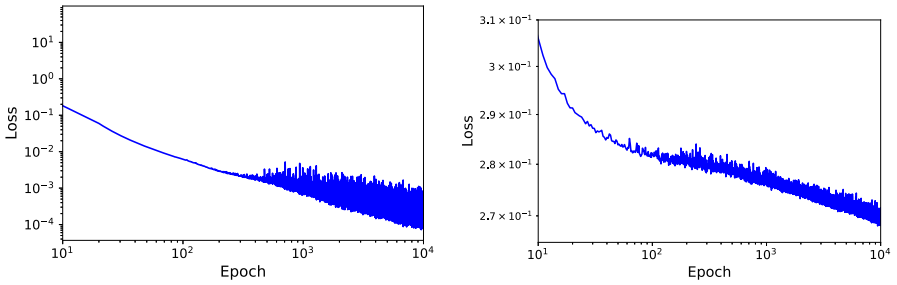
4.3 Morris–Lecar Model

We now consider the Morris–Lecar model (Morris and Lecar 1981), which is one of the simplest models for neuronal excitability. The conceptual idea of the model is that



(a) data noise level = 1% (b) data noise level = 5%

Fig. 2 (Color figure online) SEIR modeling: predicted solution from the neural network (lines) versus reference solutions (stars)



(a) Noiseless training data (b) Noisy training data with 5% noises.

Fig. 3 (Color figure online) SEIR modeling: the training loss history

cell membranes resemble electronic circuit. The phospholipid bilayer could be viewed as a capacitor because the bilayer can maintain a separation of charge. Each ion flows across the cell membrane via its own channel on the plasma membrane. This means each ion channel can been seen as a resistor. The injection of current can be regarded as a battery driving the ionic current. Because of the structure of the membrane and the channels, the capacitor and all resistors are arranged in a parallel circuit. By using Kirchhoff's law, the capacitive current should be equal to the sum of the injected current and the ionic current. That is, we can formulate the following equation:

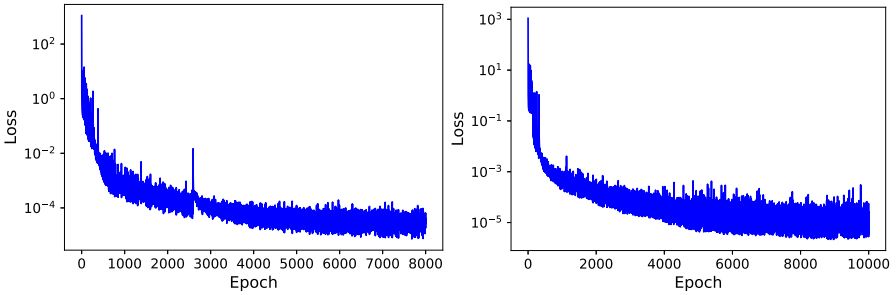
$$C_{\mathrm{M}}\frac{\mathrm{d}V}{\mathrm{d}t} = I_{\mathrm{cap}} = I_{\mathrm{ion}} + I_{\mathrm{app}}, \tag{12}$$

where I_{app} is the injected current and C_{M} is the capacitance of the membrane.

The Morris–Lecar model only involves three different channels, which are potassium channels, calcium channels and a leak channel. Thus, $I_{\mathrm{ion}} = I_{\mathrm{K}} + I_{\mathrm{Ca}} + I_{\mathrm{leak}}$. Furthermore, whether the channel is open depends on the voltage between the membrane sides, which is the difference in the concentrations of ions on opposite sides of the cellular membrane. In this model, Ca^{2+} is a fast electric current, which means that

Table 1 Parameters in the Morris–Lecar model

	C_{M}	g_L	V_L	g_{Ca}	V_{Ca}	g_k	V_k	V_1	V_2	V_3	V_4	ϕ
Type I	20	2	−60	4	120	8	−84	−1.2	18	12	17.4	0.066
Type II	20	2	−60	4.4	120	8	−84	−1.2	18	2	30	0.04



(a) Type I (b) Type II

Fig. 4 (Color figure online) The log-loss plots for the Morris–Lecar model

the behavior of the calcium channel instantaneously depends on the voltage. Therefore, we can assume the probability M of the calcium channel being open is M_∞ without time dependence. The potassium channel is a delayed rectifier, which leads to a nonconstant probability $N(t)$ for the potassium channel being opened. The true learning model is:

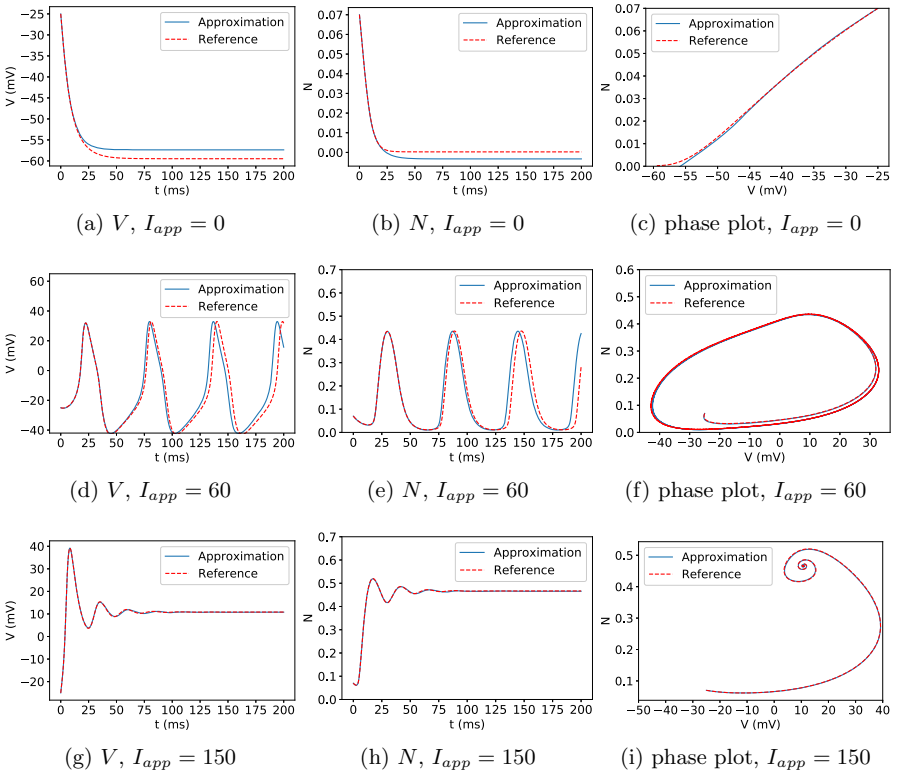
$$\begin{cases} C_{\mathrm{M}}\frac{\mathrm{d}V}{\mathrm{d}t} &= -g_L(V-V_L) - g_{\mathrm{Ca}}(V-V_{\mathrm{Ca}})M_\infty - g_k(V-V_K)N + I_{\mathrm{app}}, \\ \frac{\mathrm{d}N}{\mathrm{d}t} &= \lambda_N(N_\infty - N), \end{cases} \tag{13}$$

where

$$M_\infty = 0.5\left(1+\tanh\frac{V-V_1}{V_2}\right), N_\infty = 0.5\left(1+\tanh\frac{V-V_3}{V_4}\right), \lambda_N = \phi\cosh\frac{V-V_3}{2V_4}.$$

The Morris–Lecar model has rich bifurcation behavior depending on the parameters and the control variable I_{app}. The model can exhibit two different types of excitability under different parameters. The type I model has a continuous frequency–current curve, which means it can produce arbitrarily low frequency. On the other hand, the starting frequency of oscillation is different from zero. The generation of periodic behavior in the type I model results from a saddle node bifurcation, and the oscillations in the type II follow from the subcritical Holf bifurcation. More details can be found in Tsumoto et al. (2006). We use two sets of parameters to represent type I and II models (Rinzel and Ermentrout 1989), as listed in Table 1.

In this example, we let $\boldsymbol{\alpha}$ be the control variable, which is the applied current I_{app} varying from 0 to 300. The training domains $I_{\mathbf{x}}$ and $I_{\boldsymbol{\alpha}}$ are $[-75, 75] \times [0, 1]$ and $[0, 300]$, respectively; 100,000 samples are found to be sufficient for the DNN model



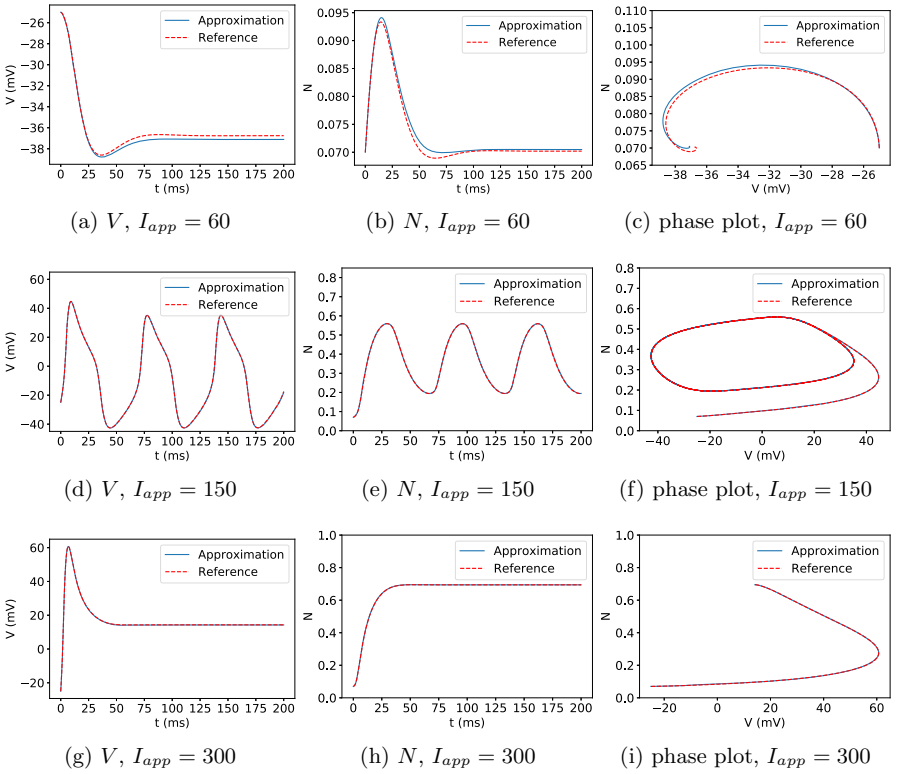
(a) V, $I_{app} = 0$ (b) N, $I_{app} = 0$ (c) phase plot, $I_{app} = 0$

(d) V, $I_{app} = 60$ (e) N, $I_{app} = 60$ (f) phase plot, $I_{app} = 60$

(g) V, $I_{app} = 150$ (h) N, $I_{app} = 150$ (i) phase plot, $I_{app} = 150$

Fig. 5 (Color figure online) Time evolution and phase plots for the type I Morris–Lecar model with $\mathbf{x}_0 = (-25, 0.07)$: **a–c** $I_{\text{app}} = 0$; **d–f** $I_{\text{app}} = 60$; **g–i** $I_{\text{app}} = 150$

training. The network training were performed for up to 10,000 epochs. The training loss history is shown in Fig. 4. Upon training the network satisfactorily, we conduct system analysis for time up to $t = 200$. We remark that the trained DNN model is valid for all system parameter I_{app}. In the following prediction and analysis, it is the same single DNN model executed at different values of I_{app}. In Figs. 5 and 6, we present system analysis using the network model for type I and type II, respectively. The system evolution of V and N are shown for different values of I_{app}. Also shown are the phase portrait of N versus V, which clearly display the different dynamical behaviors of the system under different conditions. In all these different scenarios, the ResNet model is able to produce highly accurate system predictions, as compared to the reference solutions. It is worth noting that these analyses represent rather long-time prediction for time up to 200, using training data pairs of very short length of $\Delta = 0.2$. The errors produced by the DNN network model are computed against the reference solution generated by the true model. Figure 7 contains the time evolution of the errors for the type II results from Fig. 6. The errors are ℓ_1 errors, which are combined errors in both V and N, at different level of I_{app}. We observe that the errors for steady state solutions remain stable and very small—less than 1% of relative amplitude. In the case of limit cycle solution, it is less than 2% during the $t = 200$ prediction. There is



(a) V, $I_{app} = 60$ (b) N, $I_{app} = 60$ (c) phase plot, $I_{app} = 60$

(d) V, $I_{app} = 150$ (e) N, $I_{app} = 150$ (f) phase plot, $I_{app} = 150$

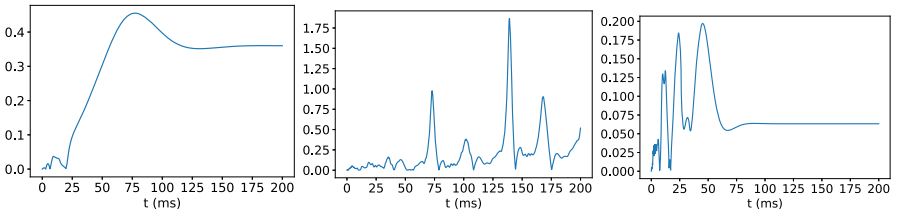
(g) V, $I_{app} = 300$ (h) N, $I_{app} = 300$ (i) phase plot, $I_{app} = 300$

Fig. 6 (Color figure online) Time evolution and phase plots for the type II Morris–Lecar model with $\mathbf{x}_0 = (-25, 0.07)$: **a–c** $I_{\text{app}} = 60$; **d–f** $I_{\text{app}} = 150$; **g–i** $I_{\text{app}} = 300$

a slow growth of errors over time. This growth is common for all numerical methods when predicting limit cycle solutions. Error behavior is very similar for all other cases and thus not shown.

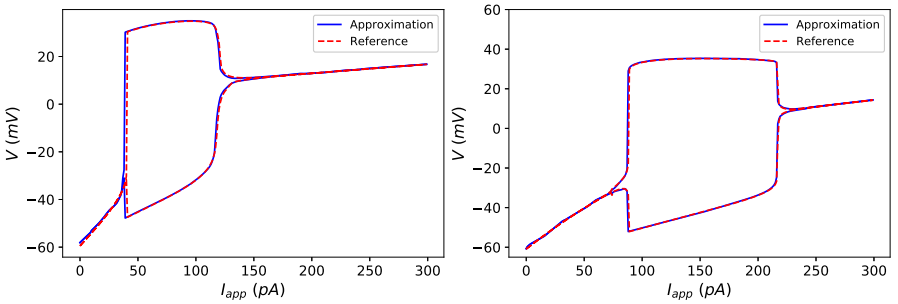
To further analyze the dynamical behavior of the system, we investigate the relation between the applied current I_{app} and the corresponding amplitude of voltage V. When the system exhibit oscillations, the maximum and minimum of the oscillations are recorded; when the system settles into a steady state, the maximum and minimum of the response converge to a single value. The results are shown in Fig. 8, for both type I and type II. We clearly observe the bifurcation behavior of the system, which transits between steady states and oscillatory states depending on the value of I_{app}. A bifurcation marks the change of the qualitative behavior of the solution. The DNN model analysis results are compared with the reference solutions from the true system. Excellent agreement can be seen in the bifurcation plot, indicating high fidelity in predictability of our trained network model.

To further examine the oscillatory behavior of the system, we employ Fourier frequency analysis to the oscillatory solutions and record the dominant frequency under different applied current I_{app}. The results are shown in Fig. 9, where we again



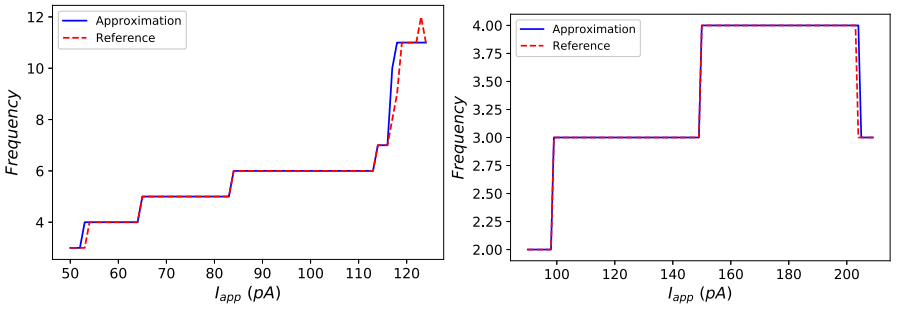
(a) $I_{app} = 60$ (b) $I_{app} = 150$ (c) $I_{app} = 300$

Fig. 7 (Color figure online) Time evolution of the DNN model prediction error of the type II Morris–Lecar model with $\mathbf{x}_0 = (-25, 0.07)$: **a** $I_{\text{app}} = 60$; **b** $I_{\text{app}} = 150$; **c** $I_{\text{app}} = 300$



(a) Type I (b) Type II

Fig. 8 (Color figure online) Bifurcation plots for the Morris–Lecar model



(a) Type I (b) Type II

Fig. 9 (Color figure online) Frequency plots for the Morris–Lecar model

observe very good agreement between the DNN model results and the reference results from the true model.

4.4 Hodgkin–Huxley Model

The Hodgkin–Huxley model describes how potential in neurons are initiated and propagated (Hodgkin and Huxley 1952). The model was developed based on abundant

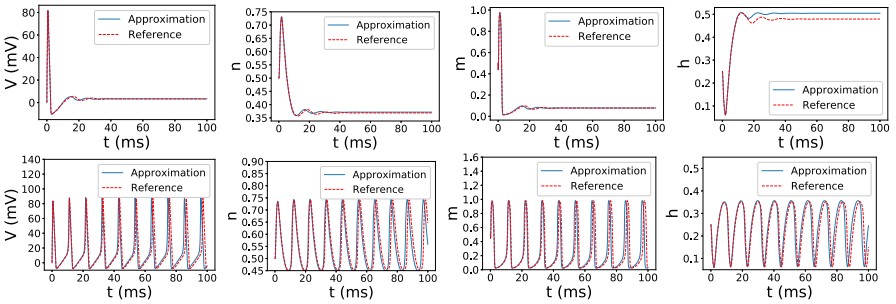


Fig. 10 (Color figure online) Time evolution of each state variable by the neural network modeling of the Hodgkin–Huxley model. Top: $I_{\text{app}} = 5$; bottom: $I_{\text{app}} = 25$

empirical data, a very much data-driven approach, and takes the following form:

$$\begin{cases} C_{\text{M}} \frac{\mathrm{d}V}{\mathrm{d}t} = -g_L(V - V_L) - g_{Na} m^3 h (V - V_{Na}) - g_K n^4 (V - V_K) + I_{\text{app}}, \\ \frac{\mathrm{d}m}{\mathrm{d}t} = -(m - m_\infty)/\tau_m, \\ \frac{\mathrm{d}h}{\mathrm{d}t} = -(h - h_\infty)/\tau_h, \\ \frac{\mathrm{d}n}{\mathrm{d}t} = -(n - n_\infty)/\tau_n, \end{cases} \tag{14}$$

where

$$\begin{cases} m_\infty = \frac{\alpha_m}{\alpha_m + \beta_m}, \ \tau_m = \frac{1}{\alpha_m + \beta_m}, \ \alpha_m = \frac{0.1(25 - V)}{\exp(\frac{25 - V}{10}) - 1}, \ \beta_m = 4\exp(\frac{-V}{18}), \\ h_\infty = \frac{\alpha_h}{\alpha_h + \beta_h}, \ \tau_h = \frac{1}{\alpha_h + \beta_h}, \ \alpha_h = 0.07\exp(\frac{-V}{20}), \ \beta_h = \frac{1}{\exp(\frac{30 - V}{10}) + 1}, \\ n_\infty = \frac{\alpha_n}{\alpha_n + \beta_n}, \ \tau_n = \frac{1}{\alpha_n + \beta_n}, \ \alpha_n = \frac{0.01(10 - V)}{\exp(\frac{10 - V}{10}) - 1}, \ \beta_n = 0.125\exp(\frac{-V}{80}). \end{cases} \tag{15}$$

Solutions of the Hodgkin–Huxley model exhibit continuous or periodical spiking behavior under different applied current I_{app}. This is one of the difficulties to approximate the solution, because this rapidly changing behavior is difficult to capture. Here, the training domain is $[-10, 120] \times [0.3, 0.8] \times [0, 1] \times [0, 0.6]$ for V, n, m, h. The sampling domain of the parameter I_{app} is $[0, 50]$. The length of all trajectory data is fixed at $\Delta = 0.05$.

The DNN model training was conducted with 100,000 data samples and 32,000 epochs. (Further increasing epochs and dataset size does not improve the performance of the model.) The trained DNN model prediction by the neural network model is presented in Fig. 10, along with the reference solution from the true model. Two cases are considered: $I_{\text{app}} = 5$ and $I_{\text{app}} = 25$. We observe that the network model predictions match well with the reference solutions. Note also that the qualitative behaviors are different: The system reaches steady state with $I_{\text{app}} = 5$ and becomes oscillatory with $I_{\text{app}} = 25$, indicating bifurcation.

To examine the bifurcation behavior of the system, we again measure the maximum and minimum of the solution evolution over time. These extrema are recorded for the different input I_{app} and plotted in Fig. 11. The excellent agreement between the DNN model analysis and the true reference solution can be clearly seen.

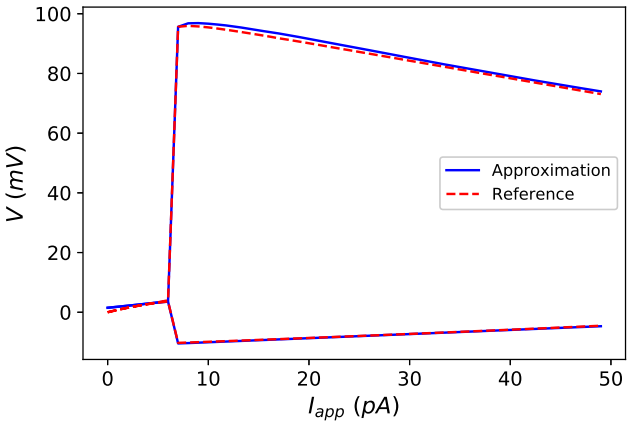


Fig. 11 (Color figure online) Bifurcation diagram for the Hodgkin–Huxley model

5 Discussion

In this paper, we have presented a method of using deep neural network (DNN) to discover unknown governing equations for biology problems. Our method extends a recent work on using residual network (ResNet) to learn dynamical system by incorporating system parameters as part of the DNN model. We applied the method to several well-known biology models, including the SEIR model, the Morris–Lecar model and the Hodgkin–Huxley model. Analysis of the learned models demonstrate that they offer good accuracy and are able to produce accurate system analysis results. In particular, the DNN models are capable of capturing the fairly complicated bifurcation behavior of the underlying dynamics.

One of the most notable features of the method lies in its use of training data. As illustrated in the (3) and (4), the training data are collections of data pairs of the state variables. Each pair consists of an *initial* state, which is generated randomly; and an *end* state, which is the initial state evolved for a short time interval Δ. In other words, the training data contain solution trajectories with only two entries. Therefore, *no physics* is present in the training data—any two states separated by the short time Δ are almost identical. However, the trained neural network models are able to learn the mechanism behind these data pairs. During system prediction, the network models execute the learned mechanism to arbitrarily given initial conditions and produce long-term system behaviors that contain *physics*, e.g., limit cycle, bifurcation, etc. In a way, the neural network models do not need to "see" the physics in order to produce the physics. This feature is different from most other existing learning methods, which require data containing relevant physics in order to predict the similar physics. This new feature of our method provides a more flexible way for data collection, as one can use a large amount of easy-to-collect trajectory data of very short length to train the models.

Although the proposed deep learning method holds much promise, there are several important open issues to consider.

- The current deep learning methods, not only our method but also other related methods, are "data hungry" and require big data. In many biological problems, access to large amount of data may not be feasible. Also, the quality of data varies greatly in practical situations. Therefore, methods on how to deal with insufficient and low-fidelity data are in need. Related to this issue, and perhaps more importantly from a practical point of view, is the issue of missing variables. That is, in many cases, one can only observe a subset of the state variables, and no data are available for the rest of the state variables. Learning governing equations for a subset of variables is therefore required. A very recent study addressed this by using Mori–Zwanzig formulation and neural networks with memory (Eldan and Shamir 2016). We will investigate the feasibility of this approach to biology problems in a future work.
- The DNN modeling method discussed in this paper is applicable when the underlying dynamical system is autonomous, that is, without explicit dependence on time. This may not be true for many complex biology problems, which may be subject to time-dependent controls or excitations. Deep learning of non-autonomous systems represents a rather non-trivial extension of the current method. This has recently been developed (Qin et al. 2020), and its applicability to biology problems needs to be examined.
- Even though our method is able to identify and predict bifurcation behaviors accurately, it is unclear how it will perform on more complex dynamical systems with chaotic behavior. This direction requires further study.

References

Abadi M, Agarwal A, Barham P, Brevdo E, Chen Z, Citro C, Corrado GS, Davis A, Dean J, Devin M, Ghemawat S, Goodfellow I, Harp A, Irving G, Isard M, Jia Y, Jozefowicz R, Kaiser L, Kudlur M, Levenberg J, Mané D, Monga R, Moore S, Murray D, Olah C, Schuster M, Shlens J, Steiner B, Sutskever I, Talwar K, Tucker P, Vanhoucke V, Vasudevan V, Viégas F, Vinyals O, Warden P, Wattenberg M, Wicke M, Yu Y, Zheng X (2015) TensorFlow: large-scale machine learning on heterogeneous systems. https://www.tensorflow.org/. Software available from tensorflow.org

Bianchini M, Scarselli F (2014) On the complexity of neural network classifiers: a comparison between shallow and deep architectures. IEEE Trans Neural Netw Learn Syst 25:1553–1565

Bongard J, Lipson H (2007) Automated reverse engineering of nonlinear dynamical systems. Proc Natl Acad Sci 104:9943–9948

Chan S, Elsheikh A (2018) A machine learning approach for efficient uncertainty quantification using multiscale methods. J Comput Phys 354:494–511

Chen RTQ, Rubanova Y, Bettencourt J, Duvenaud D (2018) Neural ordinary differential equations. Preprint arXiv:1806.07366

Chollet F et al (2015) Keras. https://github.com/fchollet/keras

Daniels BC, Nemenman I (2015a) Automated adaptive inference of phenomenological dynamical models. Nat Commun 6:8133

Daniels BC, Nemenman I (2015b) Efficient inference of parsimonious phenomenological models of cellular dynamics using S-systems and alternating regression. PLoS ONE 10:e0119821

DeAngelis DL, Yurek S (2015) Equation-free modeling unravels the behavior of complex ecological systems. https://doi.org/10.1073/pnas.1503154112

Du KL, Swamy M (2014) Neural networks and statistical learning. Springer, London

Eldan R, Shamir O (2016) The power of depth for feedforward neural networks. Conf Learn Theory 2016:907–940

Fu X, Chang LB, Xiu D (2020) Learning reduced systems via deep neural networks with memory. J Mach Learn Model Comput 2020:1
Giannakis D, Majda AJ (2012) Nonlinear Laplacian spectral analysis for time series with intermittency and low-frequency variability. Proc Natl Acad Sci 109:2222–2227
Gonzalez-Garcia R, Rico-Martinez R, Kevrekidis IG (1998) Identification of distributed parameter systems: a neural net based approach. Comput Chem Eng 22:S965–S968
Goodfellow I, Bengio Y, Courville A (2016) Deep learning. MIT Press, London
Grimm V, Railsback SF (2005) Individual-based modeling and ecology. Princeton University Press, Princeton
Hau DT, Coiera EW (1995) Learning qualitative models from physiological signals internal accession date only. Technical report
Hesthaven J, Ubbiali S (2018) Non-intrusive reduced order modeling of nonlinear problems using neural networks. J Comput Phys 363:55–78
Hethcote HW (2000) The mathematics of infectious diseases. SIAM Rev 42:599–653
He K, Zhang X, Ren S, Sun J (2016) Deep residual learning for image recognition. In: Proceedings of the IEEE conference on computer vision and pattern recognition, pp 770–778
Hodgkin AL, Huxley AF (Aug 1952) A quantitative description of membrane current and its application to conduction and excitation in nerve. https://www.ncbi.nlm.nih.gov/pmc/articles/PMC1392413/
Kevrekidis IG, Gear CW, Hyman JM, Kevrekidid PG, Runborg O, Theodoropoulos C et al (2003) Equation-free, coarse-grained multiscale computation: enabling mocroscopic simulators to perform system-level analysis. Commun Math Sci 1:715–762
Khoo Y, Lu J, Ying L (2018) Solving parametric PDE problems with artificial neural networks. Preprint arXiv:1707.03351
Kingma DP, Ba J (2017) Adam: a method for stochastic optimization. arXiv:1412.6980
Long Z, Lu Y, Ma X, Dong B (2018) PDE-net: learning PDEs from data. In: Proceedings of the 35th international conference on machine learning, 10–15 edn. vol 80 of proceedings of machine learning research, Stockholmsmässan, Stockholm, Sweden, pp 3208–3216
Mangan NM, Brunton SL, Proctor JL, Kutz JN (2016) Inferring biological networks by sparse identification of nonlinear dynamics. IEEE Trans Mol Biol Multiscale Commun 2:52–63
Mardt A, Pasquali L, Wu H, Noe F (2018) VAMPnets for deep learning of molecular kinetics. Nat Commun 9:5
Montufar GF, Pascanu R, Pascanu K, Bengio Y (2014) On the number of linear regions of deep neural networks. Adv Neural Inf Process Syst 2014:2924–2932
Morris C, Lecar H (1981) Voltage oscillations in the barnacle giant muscle fiber. https://www.ncbi.nlm.nih.gov/pmc/articles/PMC1327511/
Perretti CT, Munch SB, Sugihara G (2013) Model-free forecasting outperforms the correct mechanistic model for simulated and experimental data. Proc Natl Acad Sci USA 110:5253–5257. https://doi.org/10.1073/pnas.1216076110
Poggio T, Mhaskar H, Rosasco L, Miranda B, Liao Q (2017) Why and when can deep-but not shallow-networks avoid the curse of dimensionality: a review. Int J Autom Comput 14:503–519
Qin T, Wu K, Xiu D (2019) Data driven governing equations approximation using deep neural networks. J Comput Phys 395:620–635
Qin T, Chen Z, Jakeman J, Xiu D (2020) Data-driven learning of non-autonomous systems. SIAM J Sci Comput 2020:1
Radhakrishnan K, Hindmarsh AC (1993) Description and use of LSODE, the Livemore solver for ordinary differential equations. Technical report, Lawrence Livermore National Laboratory (LLNL), Livermore. https://doi.org/10.2172/15013302. http://www.osti.gov/servlets/purl/15013302-7Xcq5X/native/
Ray D, Hesthaven J (2018) An artificial neural network as a troubled-cell indicator. J Comput Phys 367:166–191
Rinzel J, Ermentrout G (1998) Analysis of neural excitability and oscillations. In: Koch C, Segev I (eds) Methods in neuronal modeling, 2nd edn. MIT Press, London, pp 251–291
Schmidhuber J (2015) Deep learning in neural networks: an overview. Neural Netw 61:85–117
Schmidt M, Lipson H (2009) Distilling free-form natural laws from experimental data. Science 324:81–85
Schmidt MD, Vallabhajosyula RR, Jenkins JW, Hood JE, Soni AS, Wikswo JP, Lipson H (2011) Automated refinement and inference of analytical models for metabolic networks. Phys Biol 8:055011
Sugihara G, May R, Ye H, Hsieh C, Deyle E, Fogarty M, Munch S (2012) Detecting causality in complex ecosystems. Science 338:496–500

Tripathy R, Bilionis I (2018) Deep UQ: learning deep neural network surrogate model for high dimensional uncertainty quantification. J Comput Phys 375:565–588
Tsumoto K, Kitajima H, Yoshinaga T, Aihara K, Kawakami H (2006) Bifurcations in Morris–Lecar neuron model. Neurocomputing 69:293–316. https://doi.org/10.1016/j.neucom.2005.03.006
Voss HU, Kolodner P, Abel M, Kurths J (1999) Amplitude equations from spatiotemporal binary-fluid convection data. Phys Rev Lett 83:3422
Wang Y, Cheung SW, Chung ET, Efendiev Y, Wang M (2018) Deep multiscale model learning. Preprint arXiv:1806.04830
Weinan E, Engquist B, Huang Z (2003) Heterogeneous multiscale method: a general methodology for multiscale modeling. Phys. Rev. B 67:092101
Wood SN, Thomas MB (1999) Super-sensitivity to structure in biological models. Proc R Soc B Biol Sci 266:565–570. https://doi.org/10.1098/rspb.1999.0673
Wu K, Xiu D (2019) Numerical aspects for approximating governing equations using data. J Comput Phys 384:200–221
Ye H, Beamish RJ, Glaser SM, Grant SCH, Hsieh C, Richards LJ, Schnute JT, Sugihara G (2015) Equation-free mechanistic ecosystem forecasting using empirical dynamic modeling. Proc Natl Acad Sci 112:1569–E1576
Yodzis P (1988) The indeterminacy of ecological interactions as perceived through perturbation. Technical report, p 2
Zhu Y, Zabaras N (2018) Bayesian deep convolutional encoder–decoder networks for surrogate modeling and uncertainty quantification. J Comput Phys 366:415–447