Hindawi
Applied Computational Intelligence and Soft Computing
Volume 2022, Article ID 2014510, 10 pages
https://doi.org/10.1155/2022/2014510

Research Article

Parameter Estimation for Dynamical Systems Using a Deep Neural Network

Tamirat Temesgen Dufera,[1] **Yadeta Chimdessa Seboka**,[1] **and Carlos Fresneda Portillo**[2]

[1]*Applied Mathematics, Adama Science and Technology University, Adama 1888, Oromia, Ethiopia*
[2]*Departamento de Métodos Cuantitativos, Universidad Loyola Andalucía, Avenida de Las Universidades, Dos Hermanas 41704, Sevilla, Spain*

Correspondence should be addressed to Yadeta Chimdessa Seboka; yady.chimdessa@gmail.com

Received 16 December 2021; Revised 4 April 2022; Accepted 5 April 2022; Published 27 April 2022

Academic Editor: Jun He

The deep neural network (DNN) was applied for estimating a set of unknown parameters of a dynamical system whose measured data are given for a set of discrete time points. We developed a new vectorized algorithm that takes the number of unknowns (state variables) and number of parameters into consideration. The algorithm, first, trains the network to determine weights and biases. Next, the algorithm solves the systems of algebraic equations to estimate the parameters of the system. If the right hand side function of the system is smooth and the system have equal numbers of unknowns and parameters, the algorithm solves the algebraic equation at the discrete point where absolute error between the neural network solutions and the measured data is minimum. This improves the accuracy and reduces computational time. Several tests were carried out in linear and non-linear dynamical systems. Last, we showed that the DNN approach is more successful in terms of computational time as the number of hidden layers increases.

1. Introduction

Our environment is surrounded by phenomena that can often be modeled by dynamical systems. For instance, a researcher in mathematical biology [1], ecology [2, 3], or epidemiology [4] trying to understand and predict the interaction between different species may encounter a multidimensional dynamical systems with several parameters. Another example of applications of multidimensional dynamical systems are chemical reactions [5]. These also depend on certain parameters, such as the reaction rate or the equilibrium constant [6, 7].

Further applications can be found in economy. For example, when analyzing the dynamics of a certain economic systems for predicting an unfavorable situation, such as a recession or a depression [8]. Understanding the dynamics of such cases enables planning ahead to reduce the impact of possible negative effects.

The study of the dynamics of several continuous variables leads to the analysis of a system of ordinary differential equations [9, 10]. The coefficients of these systems usually depend on some parameters which need to be estimated so that the dynamical system is valid, i.e., it provides a reliable mathematical explanation for a certain phenomenon or predictions are reliable. We refer to the estimation of this set of parameters as calibrating the dynamical system.

In most cases, estimating the parameters involved in dynamical systems is, in fact, a challenging optimization problem which needs special consideration since it is an iterative method, and at times, there can be issues with the convergence [11, 12]. Some of the standard approaches: the Gauss–Newton method [13], multiple shooting, recursive estimation [14], collocation methods [15], modified multiple shooting algorithm [16], cross-entropy approach [17], a generalized smoothing approach [18], or principal differential analysis [19]; etc. However, most of these methods suffer from one or more of the following issues: small convergence region, convergence to a local minimum instead of the absolute minimum, high computational cost, convergence highly

dependent on the initial guesses, see, for example, [20, 21] and further references therein. Despite the variety of methods, this estimation problem remains still a well-known challenging problem.

An alternative approach for calibrating dynamical systems is artificial neural networks (ANN). The beginning of ANN is often attributed to the research article by McCulloch [22]. By then, it was less popular due to the capacity of computational machines. Nevertheless, the fast development of computer science and technology in the last decade, has led to an exponential increase of the capacity of machines, for both: storing and processing data.

An ANN is defined as "an information-processing system that has certain performance characteristics in common with biological neural networks" [23, 24]. A network is composed of several layers. The first layer is usually called the input layer, whereas the last one is referred as the output layer. Layers falling in between the input and output layers are called hidden layers. Furthermore, each layer have a set of neurons or units. Deep neural networks (DNN) are ANNs with more than one hidden layer [25, 26]. DNNs are widely applied in artificial intelligence. For instance: in computer vision, image processing, pattern recognition, and cybersecurity [27–29]. The successful performance of DNNs is owed to the fact that deep layers are able to capture more variances [30].

ANNs and, in particular, DNNs, could potentially address some of the challenges of the aforementioned standard methods. One of these drawbacks is the need of a large training dataset to obtain a sufficiently accurate estimation of the parameters, which entails a high computational cost [31–33]. The ANN approach has been implemented to minimize this limitation. In particular, Dua [34, 35] proposed ANN methods for parameter estimation in systems of differential equations. However, to the best of the knowledge of the authors, we have not encountered in the literature a vectorized DNN algorithm which approximates the solution of a dynamical system with unknown parameters given a set of values of the solution in a set of time points, and thus, it is the main purpose of the paper. Additionally, we provide the following original results.

(1) We extended the algorithm from [36] for systems of differential equations to systems of differential equations with unknown parameters.

(2) We enhance the efficiency and accuracy of the algorithm in the case when the number of unknowns of the dynamical system coincides with the number of unknown parameters.

(3) We show that the DNN algorithm for this problem becomes faster in terms of computational time as the number of layers increase.

For the calculation of gradients of the cost functions, we utilized the auto-differentiation technique supported in the code by the Autograd package [37] and for the optimization of the learning rule, we implemented the Adam method [38], which successfully addresses the local minimum problem present in the standard approaches.

The paper is structured as follows: (2) mathematical formulation of the problem; (3) DNN model; (4) vectorized algorithm for parameter estimation; (5) numerical experiments; and (6) conclusions and further work.

2. Problem Formulation

Let us consider a dynamical system described by n ordinary differential equations involving m unknown parameters,

$$\begin{aligned} \frac{\mathrm{d}\mathbf{u}(t)}{\mathrm{d}t} &= \mathbf{f}(t, \mathbf{u}(t), \mathbf{p}), \quad t \in (t_0, t_{\text{end}}) \subset \mathbb{R}, \\ \mathbf{u}(t_0) &= \mathbf{u}_0, \end{aligned} \tag{1}$$

where $\mathbf{u}(t) = [u_1(t), u_2(t), \ldots, u_n(t)]^T$ is a vector field with n components $u_i(t)$, $i = 1, \ldots, n$; $\mathbf{p} = [p_1, p_2, \ldots, p_m] \in \mathbb{R}^m$ is the vector containing the unknown parameters; $u_0 \in \mathbb{R}^n$ is the initial condition and

$$\mathbf{f}(t, \mathbf{u}, \mathbf{p}): = \begin{bmatrix} f_1(t, u_1, u_2, \ldots, u_n, p_1, p_2, \ldots, p_m) \\ f_2(t, u_1, u_2, \ldots, u_n, p_1, p_2, \ldots, p_m) \\ \vdots \\ f_n(t, u_1, u_2, \ldots, u_n, p_1, p_2, \ldots, p_m) \end{bmatrix}, \tag{2}$$

is a given vector-valued function, not necessarily linear, with n components $f_i(t, u_1, u_2, \ldots, u_n, p_1, p_2, \ldots, p_m)$, $i = 1, \ldots, n$. The measured data for the model equation (1) will be denoted by $\mathbf{u}^*(t)$.

Objective: Given measured data at N time points of (1), i.e., $(t_k, \mathbf{u}^*(t_k))$ with $k = 1, 2, \ldots, N$, we aim to develop a vectorized deep neural network algorithm that estimates the unknown parameters $\mathbf{p}$ and the solution $\mathbf{u}(t)$ for all t.

3. Deep Neural Network Model

We consider a deep neural network with a similar architecture as in [36] for non-parametric systems of ODEs depicted in Figure 1. In this network diagram, $t_i \in (t_0, t_{\text{end}})$ with $i \in \{1, 2, \ldots N\}$ represents N time points; $a_k{}^{(l)}$ is the state of the j-th neuron in layer l with $l \in \{1, 2, \ldots, L\}$, where L denotes the total number of layers; $b^{(l)}$ denotes the bias in layer l; and $W^{(l)}: = w_{jk}^{(l)}$ denotes the weight matrix of layer l. Lastly, $\widehat{u}_j^{(i)}$ denotes the estimated solution of the unknown function $u_j(t^{(i)})$, evaluated at the time point $t^{(i)}$.

The architecture of the DNN is based on the following:

(i) An input layer $(l = 0)$ consisting of a single neuron corresponding to the time point $t^{(i)}$;

(ii) An output layer $(l = L)$ with n output functions $\widehat{u}_j^{(i)}$;

(iii) $L - 1$ hidden layers with h_l neurons in each layer, $l = \{1, \ldots L - 1\}$.

Let us remark that the number of neurons in each hidden layer might not be the same. As a result, the weight matrix W has dimension $h_l \times h_{l-1}$, and thus it might not be a square matrix.

Moreover, the number of neurons in each hidden layer could be determined based on the performance of the model [36]. Here, by performance of the model, we mean the DNN architecture which gives the best approximation with a small number of iterations and reduced computational time.

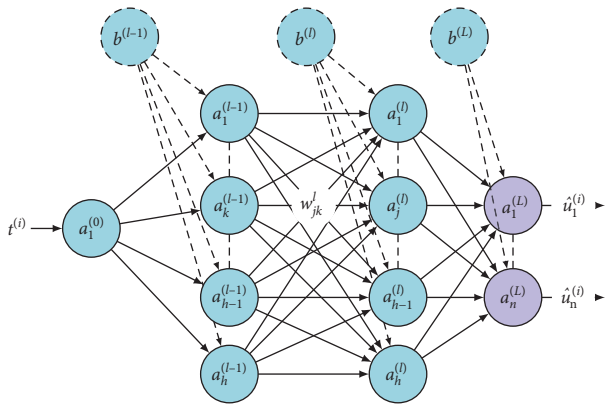

Figure 1: Schematic diagram of DNN.

3.1. Feed-Forward Propagation. The feed-forward network proceeds as follows. For each data point $t^{(i)}$, the learning process starts by assigning $a_1^{[0]} = t^{(i)}$.

Then, to obtain $\mathbf{a}^{(l)}$ given $\mathbf{a}^{(l-1)}$, we require two steps. The first step is working out a provisional vector $\mathbf{z}^l$ by multiplying $\mathbf{a}^{(l-1)}$ by the weight matrix W^l, and then we add the bias vector $\mathbf{b}^l$, i.e.,

$$z_j^l := \sum_{k=1}^{h_{l-1}} w_{jk}^l a_k^{l-1} + b_j^l. \tag{3}$$

Alternatively, one can write equation (3) in matrix form as follows:

$$\mathbf{z}^l = W^l \mathbf{a}^{l-1} + \mathbf{b}^l. \tag{4}$$

The second and last step requires applying an appropriate choice of activation function σ^l to $\mathbf{z}^l$ to obtain $\mathbf{a}^l$. Thus, the values for the next layer are expressed by the following:

$$\mathbf{a}^l = \sigma^l\left(\mathbf{z}^l\right) = \sigma^l\left(W^l \mathbf{a}^{l-1} + \mathbf{b}^l\right). \tag{5}$$

In short, for each given time point t_i, we will obtain a $\mathbf{z}^l$ and an output $\mathbf{a}^l$. To illustrate, the dependence from t_i, we will add the superscript (i), i.e., $\mathbf{a}^{l(i)} = \sigma^l(\mathbf{z}^{l(i)})$. Therefore, using vector and matrix notation, we can write $T = [t_1, \ldots, t_N]$, $Z^l = [\mathbf{z}^{l(1)}, \ldots, \mathbf{z}^{l(N)}]^t$ and $A^l = [\mathbf{a}^{l(1)}, \ldots, \mathbf{a}^{l(N)}]$ so that we can summarise (4) and (5) for all t_i

$$\begin{aligned} A^0 &= T, \\ Z^l &= W^l A^{l-1} + \mathbf{b}^l, \\ A^l &= \sigma^l\left(Z^l\right), \quad \text{for } l \in \{1, \ldots, L\}. \end{aligned} \tag{6}$$

We will refer to the values of the output layer A^L as the output of the network, i.e., $N(T, W^L, \mathbf{b}^L) := A^L$ or, in component form $N_j(t_i, W^L, \mathbf{b}^L) := \pi_j(A^L) := A_j^L(t_i, W, \mathbf{b})$, where π_j denotes the j-th projection.

3.2. Parameter Estimation. In this section, we describe the DNN algorithm for estimating the parameters in the dynamical system (1).

Let us suppose that we have measured data at N points of (1), i.e., $(t_k, \mathbf{u}^*(t_k))$ with $k = 1, 2, \ldots, N$. For each t_i, $i = 1, \ldots, N$, we compute the network output, denoted by $A_j^L(t_i, W, \mathbf{b})$. Then, the trial solution satisfying the initial conditions is given by the following:

$$\widehat{u}_j\left(t^{(i)}, W, \mathbf{b}\right) = u_{0j} + \left(t^{(i)} - t_0\right) A_j^L\left(t^{(i)}, W, \mathbf{b}\right), \quad j = 1, \ldots, n. \tag{7}$$

The expression (7) can be written in the matrix form as follows:

$$\widehat{\mathbf{u}}(T, W, \mathbf{b}) = \mathbf{u}_0 + (T - t_0 1) N(T, W, \mathbf{b}). \tag{8}$$

We train the network to find W and $\mathbf{b}$ by minimizing the following cost function,

$$J_1(T, W, \mathbf{b}) = \frac{1}{2} \sum_{i=1}^{m} \left\| \widehat{\mathbf{u}}\left(t^{(i)}, W, \mathbf{b}\right) - \mathbf{u}^{(i)} \right\|^2. \tag{9}$$

With the optimal learning parameters W^* and $\mathbf{b}^*$, the approximate solutions of the dynamical systems were obtained. In this paper, Adaptive Moment method (also called Adam method) was applied to update the learning parameters [39]. The updating rule is

$$\begin{aligned} M_w^k &= \beta_1 M_w^{k-1} + (1 - \beta_1) \nabla J_1\left(W^k\right), \\ M_b^k &= \beta_1 M_b^{k-1} + (1 - \beta_1) \nabla J_1\left(\mathbf{b}^k\right), \\ V_w^k &= \beta_2 V_w^{k-1} + (1 - \beta_2) \left(\nabla J_1\left(W^k\right)\right)^2, \\ V_b^k &= \beta_2 V_b^{k-1} + (1 - \beta_2) \left(\nabla J_1\left(\mathbf{b}^k\right)\right)^2, \\ \widehat{M}_w^k &= \frac{M_w^k}{1 - \beta_1^k}, \\ \widehat{V}_w^k &= \frac{V_w^k}{1 - \beta_2^k}, \\ \widehat{M}_b^k &= \frac{M_b^k}{1 - \beta_1^k}, \\ \widehat{V}_b^k &= \frac{V_b^k}{1 - \beta_2^k}, \\ W^{k+1} &= W^k - \frac{\eta}{\sqrt{\widehat{V}_w^k} + \epsilon} \widehat{M}_w^k, \\ \mathbf{b}^{k+1} &= \mathbf{b}^k - \frac{\eta}{\sqrt{\widehat{V}_b^k} + \epsilon} \widehat{M}_b^k, \end{aligned} \tag{10}$$

where $\beta_1, \beta_2 \in [0, 1)$ are decay rates for the moment estimates, η is the learning rate, and V_w, V_b and M_w, M_b are the first and the second moment vectors, respectively, that are initialized to be zero. The values $\eta = 0.001$, $\beta_1 = 0.9, \beta_2 = 0.999$ and $\epsilon = 10^{-8}$ are usually applied in the literature, see e.g., [39].

After obtaining the optimal values of the learning parameters W^* and $\mathbf{b}^*$, we seek the values of parameter $\mathbf{p}$ by solving the following non-linear system of algebraic equations,

$$\mathbf{F}(\mathbf{p}): = \frac{\mathrm{d}\hat{\mathbf{u}}}{\mathrm{d}t}\left(t^{(i)}, W^*, \mathbf{b}^*\right) - \mathbf{f}\left(t^{(i)}.\hat{\mathbf{u}}, \mathbf{p}\right) = 0. \tag{11}$$

When attempting to solve (11), we will distinguish two possible cases: (1) The number of equations equals to the number of parameters and (2) Otherwise.

Case 1. For the particular case in which the right hand side function of the dynamical system (1) has as many parameters as equations, i.e., $n = m$, the following proposition addresses the existence of a solution in this case. This proposition directly follows from the well-known inverse function theorem.

Proposition 1. *If number of unknowns and number of parameters are equal, i.e.,* $(n = m)$, $\mathbf{F}(\mathbf{p}) \in \mathbb{R}^n$ *is* C^1 *in a neighborhood of some initial parameters* $\mathbf{p}_0$, *and if the Jacobian* $(\partial\mathbf{F}/\partial\mathbf{p})(\mathbf{p}_0)$ *is non-singular, then equation (9) has unique solution in the neighborhood of* $\mathbf{p}_0$ *and at point* $t^* \in (t_0, t_{end})$.

Proof. Let $t^* \in (t_0, t_{\mathrm{end}})$, then $\mathbf{C}^* = (\mathrm{d}\hat{\mathbf{u}}/\mathrm{d}t)(t^*, W^*, \mathbf{b}^*)$ is constant vector, and

$$\mathbf{F}(\mathbf{p}) = \mathbf{C}^* - \mathbf{f}\left(t^*, \hat{\mathbf{u}}, \mathbf{p}\right) = 0, \tag{12}$$

is a system of non-linear algebraic equations having n equations in n unknowns. Hence, the proposition holds by virtue of the inverse function theorem. □

Remark 1. The properties of $\mathbf{F}$ in the proposition is determined by the right hand side function $\mathbf{f}$ of the dynamical systems (1). Hence, we require the $\mathscr{C}^1$ smoothness of $\mathbf{f}$ in order to apply Proposition 1.

Proposition 2. *The best estimation for the parameter* $\mathbf{p}$ *is obtained if* t^* *is chosen according to Proposition 1 such that the sum of the absolute error between solutions of the neural network and the measured data is minimum. That is, choose* t^* *such that*

$$\sum_{j=1}^{n} \left\| \hat{u}_j\left(t^*, W^*, \mathbf{b}^*\right) - u_j\left(t^*\right) \right\|, \tag{13}$$

is minimum.

Proof. Condition (13) ensures the best fit for the solution trajectories. □

Case 2. The number of parameters does not equal to the number of equations of the dynamical system. In this case, we can solve the non-linear system (11) by minimizing the following objective function

$$J_2(\mathbf{p}) = \frac{1}{2}\sum_{i=1}^{m} \left\| \frac{\mathrm{d}\hat{\mathbf{u}}}{\mathrm{d}t}\left(t^{(i)}, W^*, \mathbf{b}^*\right) - \mathbf{f}\left(t^{(i)}, \hat{\mathbf{u}}, \mathbf{p}\right) \right\|^2. \tag{14}$$

Remark 2. Let us note, that the approach suggested in Case 2 can also be applied to Case 1 but not the other way round.

4. Vectorized Algorithm for Parameter Estimation

In this section, we describe the DNN algorithm to estimate the parameters of the system of differential equations (1). It differs from the algorithm in [36]. The current algorithm follows supervised learning, since we have the target output. Additionally, it includes the case where the number of parameters equals the number of unknowns.

(1) Input data: Define the vector $T = [t^{(1)}, t^{(2)}, \ldots, t^{(m)}]$ of size $1 \times m$.

(2) Define the deep neural network structure: Determine the numbers of layers L, input layer (having one unit), $L - 1$ hidden layers (having n^l units), and the output layer (having n units).

(3) Initialize the network parameters: Choose the weights to have small random entries, whereas the biases and the moment matrices in the Adam algorithm to have zero entries.

- (i) W^1 has $n^1 \times 1$ dimension,
- (ii) W^l has $n^l \times n^{l-1}$ dimension, $\mathbf{b}^l$ has $n^l \times 1$ dimension,
- (iii) W^L has $n \times n^{L-1}$ dimension, $\mathbf{b}^L$ has $n \times 1$ dimension,
- (iv) M_w and V_w have the same size as the corresponding W, and
- (v) M_b and V_b have the same size as the corresponding $\mathbf{b}$.

(4) Forward propagation:

- (i) For the input layer start by assigning, $A^0 = T$.
- (ii) For the hidden layers, $1 \le l \le L - 1$,

$$Z^l = W^l A^{l-1} + \mathbf{b}^l, \quad A^l = \sigma^l\left(Z^l\right), \tag{15}$$

where σ^l is the activation function corresponding to the l^{th} hidden layer.

- (iii) For the output layer,

$$Z^L = W^L A^{L-1} + \mathbf{b}^L, \quad A^L = \sigma^L\left(Z^L\right). \tag{16}$$

- (iv) Assign the trial solution using equation (8).

(5) Compute the cost $J_1(T, W, \mathbf{b})$ and its gradient (9): Calculate the partial derivatives of the J_1 function, given in (9), gradients with respect to T, W and $\mathbf{b}$. To carry out these computations, we apply automatic differentiation techniques [37, 40].

(6) Update the learning parameters using the Adam method described in (10).

(7) Solve the non-linear system of algebraic equations (11) to obtain the parameter $\mathbf{p}$. Recall that there are two possible scenarios.

(i) If the number of unknowns equals that of parameters i.e., $m = n$, find t^* which minimize the absolute error given by (13) and solve the algebraic equation given by (12).

(ii) If $m \neq n$; solve the minimization problem (14).

5. Numerical Experiments

In this section, we apply the algorithm proposed in the previous Section 4, implemented in Python, to four different benchmark problems taken from the literature. The choice of numbers of neurons, numbers of hidden layers, activation functions, and other network parameters are specified within each example. Three out of four of these problems have an analytical exact solution which is then used for validating the model.

Example 1. In this example, taken from [34, 35] and ([6], problem 1), we consider the following system of differential equations related to the mathematical modeling of a chain of irreversible chemical reactions

$$\begin{aligned} \frac{du_1}{dt} &= -k_1 u_1, \quad u_1(0) = 1, \\ \frac{du_2}{dt} &= k_1 u_1 - k_2 u_2, \quad u_2(0) = 0. \end{aligned} \tag{17}$$

This system consists of two unknown functions, two parameters. In this case, given the parameters $[k_1, k_2] = [5, 1]$, the analytical expression of the solution is as follows:

$$\begin{aligned} u_1(t) &= e^{-5t}, \\ u_2(t) &= \frac{5}{4} e^{-5t}\left(-1 + e^{4t}\right). \end{aligned} \tag{18}$$

Let us take discrete points from the analytical solution (18), see Table 1.

To solve this problem, we considered a neural architecture with one hidden layer of $h = 40$ neurons. Furthermore, sigmoid activation functions were applied to each unit and trained with 90000 epochs.

The results of applying the algorithm are shown in Figure 2. On the one hand, Figure 2(a) proves the accuracy of the method as we cannot distinguish between the analytical and the approximated solution. On the other hand, Figure 2(b) shows that the absolute error is bounded by 0.00015.

The parameters k_1 and k_2 were worked out using both approaches, Case 1 and Case 2, (see Remark 2). For the approach described in Case 1, the corresponding system of algebraic equation (11) was solved at $t^* = 0.6$. Table 2 shows the estimated parameters obtained through the approaches described in Case 1 and Case 2, which can be applied due to Remark 2. In this particular example, the approach from Case 1 proved to be more accurate. Moreover, the computational time of Step 7 in the algorithm was 2.5 ms CPU time for the first approach, whereas for the second approach, it was 7.6 ms.

Table 1: Discrete points of the analytical solution (18).

t	u_1	u_2
0.0	1.000000	0.000000
0.1	0.606531	0.372883
0.2	0.367879	0.563564
0.3	0.223130	0.647110
0.4	0.135335	0.668731
0.5	0.082085	0.655557
0.6	0.049787	0.623781
0.7	0.030197	0.582985
0.8	0.018316	0.538767
0.9	0.011109	0.494326
1.0	0.006738	0.451427

Example 2. Dynamical system models a biomass transfer ([41], p. 531).

$$\begin{aligned} \frac{du_1}{dt} &= -c_1 u_1 + c_2 u_2, \\ \frac{du_2}{dt} &= -c_2 u_2 + c_3 u_3, \\ \frac{du_3}{dt} &= -c_3 u_3, \end{aligned} \tag{19}$$

with initial condition $\mathbf{u}(0) = (0, 0, 1)^T$. For $\mathbf{c} = [1, 3, 5]$, the analytical solution is given by the following:

$$\begin{aligned} u_1(t) &= \frac{15}{8}\left(e^{-5t} - 2e^{-3t} + e^{-t}\right), \\ u_2(t) &= \frac{5}{2}\left(-e^{-5t} + e^{-3t}\right), \\ u_3(t) &= e^{-5t}. \end{aligned} \tag{20}$$

Following a similar procedure as for the previous application example, we consider 11 points uniformly distributed in the interval from the analytical solution (20), see, Table 3.

In this example, we have three unknown functions and three system parameters. The neural architecture considered has one hidden layer with $h = 10$ neurons, tanh activation functions for each unit. The network is trained with 71800 epochs and the results are shown in Figure 3. On the one hand, Figure 3 shows the estimated solutions using the DNN approach and the analytical solutions. As we can see, Figure 3(a) proves again the accuracy of the method. On the other hand, Figure 3(b) shows that the absolute error is bounded by 0.0002. The corresponding system of algebraic equation (11) was solved at $t^* = 0.4$. Again, the parameter estimation was conducted using approaches from Case 1 and Case 2. The output is shown in Table 4. Last, the computational time of Step 7 in the algorithm was 2.9 ms CPU time for the first approach and 10.5 ms for the second approach.

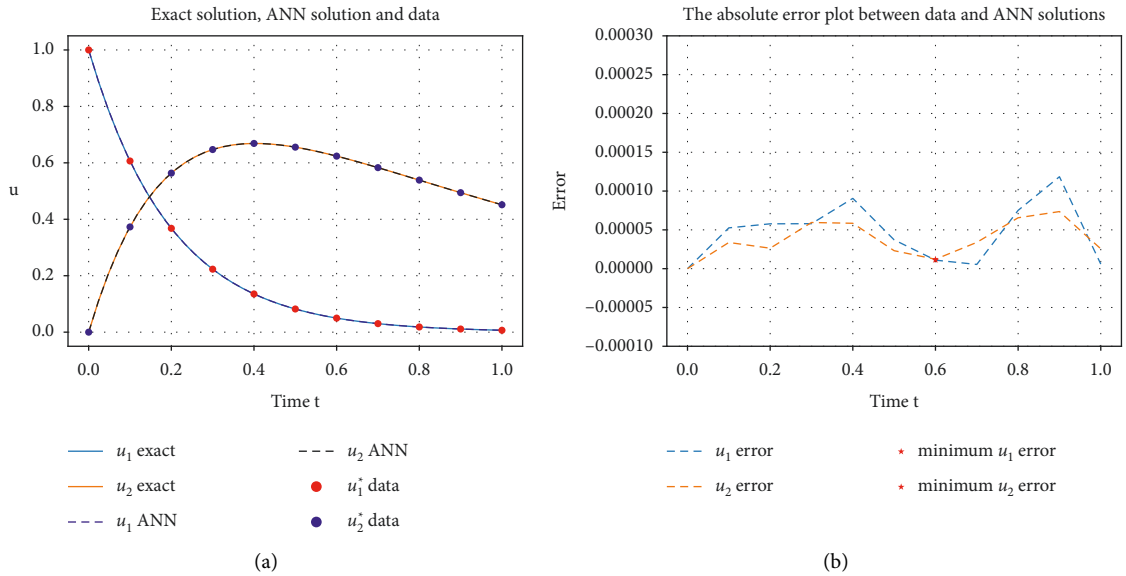

Figure 2: Results of Example 1. (a) Exact solution, ANN, and data. (b) Absolute error.

Table 2: Comparison of the two approaches for Example 1.

K	True values	Approach 1	Error of approach 1	Approach 2	Error of approach 2
k_1	5.0	5.0050	0.005	4.9890	0.011
k_2	1.0	1.0006	0.006	0.9974	0.003

Table 3: Data generated for Example 2.

t	u_1	u_2	u_3
0.0	0.000000	0.000000	1.000000
0.1	0.055747	0.335719	0.606531
0.2	0.166850	0.452330	0.367879
0.3	0.282767	0.458599	0.223130
0.4	0.381125	0.414647	0.135335
0.5	0.454416	0.352613	0.082085
0.6	0.502502	0.288780	0.049787
0.7	0.528506	0.230648	0.030197
0.8	0.536641	0.181006	0.018316
0.9	0.531127	0.140241	0.011109
1.0	0.515706	0.107623	0.006738

We conclude that in this example, the approach described in Case 1 was again more successful.

Example 3. The next problem is taken from [20] and consists of a dynamical system with initial value condition containing three parameters p_1, p_2, and p_3, see also, ([6], problem 7). The model occurs in several applications such as chemical kinetics, theoretical biology, ecology, etc.

$$\begin{aligned} \frac{du_1}{dt} &= p_1 u_1 - p_2 u_1 u_2, \quad u_1(0) = 1.0, \\ \frac{du_2}{dt} &= p_2 u_1 u_2 - p_3 u_2, \quad u_2(0) = 0.3. \end{aligned} \tag{21}$$

The data are taken from reference [20], as shown in Table 5.

It is to be noted that the number of unknowns and parameters do not coincide in this example. Hence, unique solvability for the corresponding algebraic system (11) is not guaranteed. For solving this ODE system, we considered a single layer neural network with 60 units in the hidden layers, we have chosen 100000 epochs and the activation functions in the hidden layer were tanh functions. Using the approach described in Case 2, we obtain $\mathbf{p} = [p_1, p_2, p_3] = [0.854, 2.201, 2.013]$. The proposed algorithm gives better fit compared to the results reported in [20]. This estimation of the parameters provides an accurate solution; see Figure 4.

Example 4. Consider a dynamical system with three state variables which models methanol-to-hydrocarbon process ([34], Example 5).

$$\begin{aligned} \frac{du_1}{dt} &= -\left(2k_1 - \frac{k_1 u_2}{(k_2 + k_5)u_1 + u_2} + k_3 + k_4\right)u_1, \\ \frac{du_2}{dt} &= \frac{k_1 u_1 (k_2 u_1 - u_2)}{(k_2 + k_5)u_1 + u_2} + k_3 u_1, \\ \frac{du_3}{dt} &= \frac{k_1 u_1 (u_2 + k_5 u_1)}{(k_2 + k_5)u_1 + u_2} + k_4 u_1, \end{aligned} \tag{22}$$

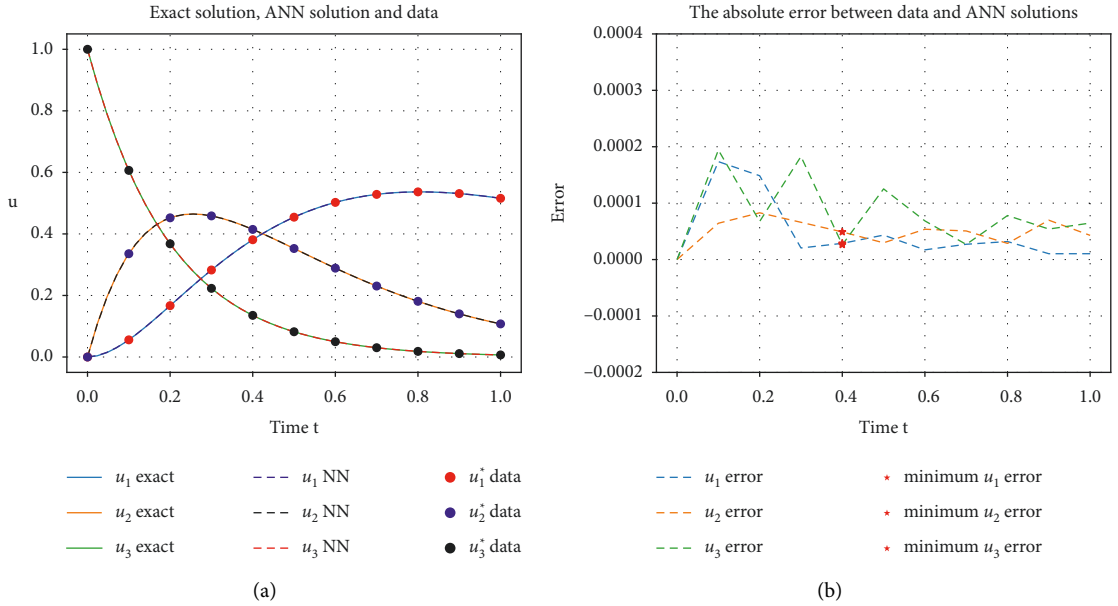

Figure 3: Plot for comparing the ANN solution and exact solution. (a) Exact solution, ANN and data. (b) Absolute error.

Table 4: Comparison of the two approaches for Example 2.

c	True values	Approach 1	Error of approach 1	Approach 2	Error of approach 2
c_1	1	1.0024	0.0024	0.9896	0.0103
c_2	3	3.0026	0.0026	2.9811	0.1890
c_3	5	5.0150	0.0150	4.9615	0.0385

Table 5: Measurement taken at discrete points Example 3.

t	u_1	u_2
0.0	1.0	0.3
0.5	1.1	0.35
1.0	1.3	0.4
1.5	1.1	0.5
2.0	0.9	0.5
2.5	0.7	0.4
3.0	0.5	0.3
3.5	0.6	0.25
4.0	0.7	0.25
4.5	0.8	0.3
5.0	1.0	0.35

with initial conditions, $\mathbf{u}_0 = [1, 0, 0]$, and $t \in [0, 1]$. We took the data from the reference ([34], Table 6).

Using the second approach, we estimated the parameters k_1, k_2, k_3, k_4, and k_5 for Example 4. Here, the sigmoid activation function with 90000 epochs and one hidden layer having 40 neurons was used. See Table 7 for comparison. Figure 5 shows how the solution trajectories nicely fitted to the data.

Table 8 shows that the results obtained in this paper are in agreement with those from the existing literature.

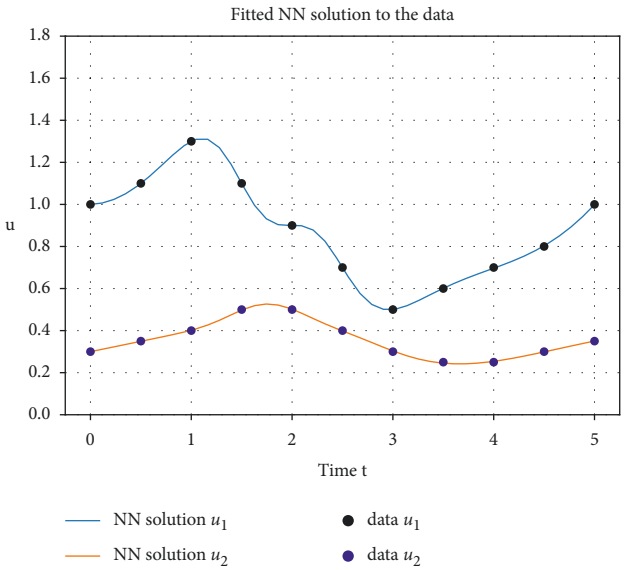

Figure 4: Neural network solution and the measured data for Example 3.

5.1. Shallow vs. Deep Layers. Here, we looked at the effect of adding more layers in the network. Specifically, we compared the network architecture with one, two, and three

Table 6: Comparison of shallow vs. deep layers for Example 2.

Layers and neurons	k_1	k_2	k_3	Time (sec)	Epochs
1, (40)	1.0044	2.9936	4.9451	20	20000
2, (40, 40)	1.0051	3.0125	5.0515	19	15107
3, (40, 40, 40)	1.0044	2.9936	4.9451	16	9800

Table 7: Comparison of the true and estimated parameters for Example 4.

k	True values	Approach 2	Error of approach 2
k_1	5.2	5.199	0.001
k_2	1.2	1.202	0.002
k_3	0.0	0.0045	0.0045
k_4	0.0	−0.0174	0.0174
k_5	0.0	0.0056	0.0056

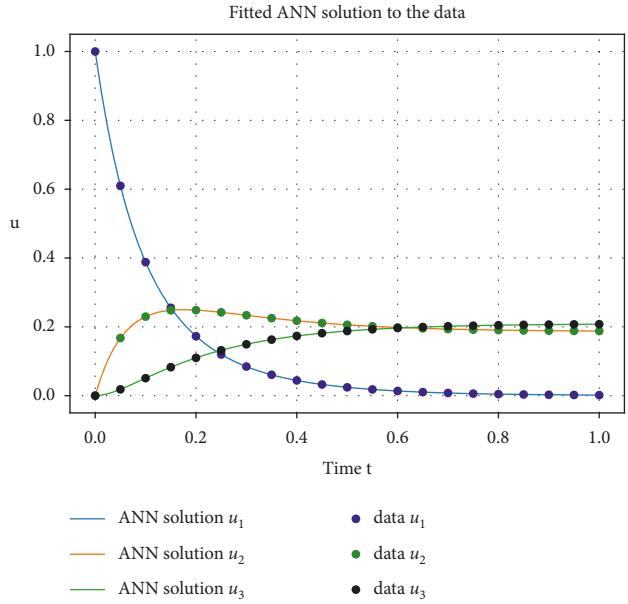

Figure 5: Neural network solution and the measured data for Example 4.

Table 8: Comparison of of results obtained vs. reported in literature.

Examples	True values	Reported in literature	This work	Epochs	CPU time (s)
Example 1	$k_1 = 5$ $k_2 = 1$	5.083 [34] 0.981 [34]	5.005 1.0006	90000	79
Example 4	$k_1 = 5.2$ $k_2 = 1.2$ $k_3 = 0$ $k_4 = 0$ $k_5 = 0$	5.187 [35] 1.199 [35] 0 2.028×10^{-4} 0	5.199 1.202 0.00145 −0.017 0.0056	90000	79

hidden layers. We considered Examples 1 and 2. In the three possible network structure, we take the same activation function, $1/(1 + e^{-2.5x})$ and we fix the error tolerance of the cost function to be 10^{-6}. Although further analysis is required, the experimental results from Tables 9 and 6 show that reasonable accuracy on the parameter estimation could be achieved by increasing the hidden layer with less numbers of epochs and computation time.

Table 9: Comparison of shallow vs. deep layers for Example 1.

Layers and neurons	k_1	k_2	Time (sec)	Epochs
1, (40)	5.0460	0.9979	19	21378
2, (40, 40)	5.0549	0.9997	30	24358
3, (40, 40, 40)	4.9103	1.0129	26	16021

6. Conclusions and Outlook

In this paper, we developed a vectorized deep neural network algorithm for estimating the unknown parameters in a dynamical system based on a system of ODEs as well as the solution functions of the system. This problem arises in different sciences as shown in the Numerical Experiments Section. We highlighted that there are two approaches applicable to problems where the number of parameters coincides with the number of ODEs in the system. In this case, the system of algebraic equation arising from the optimal learning parameters (weights and biases) is uniquely solved at a specific point provided that the right side function satisfies the conditions stated in Proposition 1. Choosing t^* significantly improves the accuracy of the parameter estimation.

Last, the experimental result shows that, for deep neural network, reasonable accuracy of parameter estimation could be achieved for less number of epochs and hence less running time.

For the future work, it is worth comparing the proposed method with the traditional methods in terms of accuracy, computational complexity, and robustness. Furthermore, it is worth also comparing with other neural network architectures. Moreover, the method is yet to be exploited, for instance, for solving delay differential equations, stochastic dynamical systems, or rather more complex problems such as integral equations.

Data Availability

The underlying data supporting this result are available from the literature cited in this article.

Conflicts of Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

The third author truly acknowledges the support received from the grant PID2020-117800GB-I00 of the Ministry of Science and Innovation of the Government of Spain for the research project METRICA.

References

[1] M. Ashyraliyev, Y. Fomekong-Nanfack, J. A. Kaandorp, and J. G. Blom, "Systems biology: parameter estimation for biochemical models," *FEBS Journal*, vol. 276, no. 4, pp. 886–902, 2009.

[2] D. Gonze, K. Z. Coyte, L. Lahti, and K. Faust, "Microbial communities as dynamical systems," *Current Opinion in Microbiology*, vol. 44, pp. 41–49, 2018.

[3] S. Qureshi and A. Yusuf, "Mathematical modeling for the impacts of deforestation on wildlife species using caputo differential operator," *Chaos, Solitons & Fractals*, vol. 126, pp. 32–40, 2019.

[4] F. Brauer, C. Castillo-Chavez, and Z. Feng, *Mathematical Models in Epidemiology*, Vol. 32, Springer, Berlin, Germany, 2019.

[5] D. J. Higham, "Modeling and simulating chemical reactions," *SIAM Review*, vol. 50, no. 2, pp. 347–368, 2008.

[6] I. B. Tjoa and L. T. Biegler, "Simultaneous solution and optimization strategies for parameter estimation of differential-algebraic equation systems," *Industrial & Engineering Chemistry Research*, vol. 30, no. 2, pp. 376–385, 1991.

[7] M. Peifer and J. Timmer, "Parameter estimation in ordinary differential equations for biochemical processes using the method of multiple shooting," *IET Systems Biology*, vol. 1, no. 2, pp. 78–88, 2007.

[8] J. Han, A. Jentzen, and W. E, "Solving high-dimensional partial differential equations using deep learning," *Proceedings of the National Academy of Sciences*, vol. 115, no. 34, pp. 8505–8510, 2018.

[9] H. Murakami and R. Zimka, "On dynamics in a two-sector Keynesian model of business cycles," *Chaos, Solitons & Fractals*, vol. 130, Article ID 109419, 2020.

[10] V. Novotná and V. Štěpánková, "Parameter estimation for dynamic model of the financial system," *Acta Universitatis Agriculturae et Silviculturae Mendelianae Brunensis*, vol. 63, no. 6, pp. 2051–2055, 2015.

[11] M. Nouiehed and M. Razaviyayn, "Learning deep models: critical points and local openness," 2018, https://arxiv.org/abs/1803.02968.

[12] C. Yun, S. Sra, and A. Jadbabaie, "A critical view of global optimality in deep learning," 2018, https://arxiv.org/abs/1802.03487.

[13] N. Kalogerakis and R. Luus, "Improvement of gauss-Newton method for parameter estimation through the use of information index," *Industrial & Engineering Chemistry Fundamentals*, vol. 22, no. 4, pp. 436–445, 1983.

[14] H. U. Voss, J. Timmer, and J. Kurths, "Nonlinear dynamical system identification from uncertain and indirect measurements," *International Journal of Bifurcation and Chaos in Applied Sciences and Engineering*, vol. 14, no. 6, pp. 1905–1933, 2004.

[15] N. Baden and J. Villadsen, "A family of collocation based methods for parameter estimation in differential equations," *The Chemical Engineering Journal*, vol. 23, no. 1, pp. 1–13, 1982.

[16] O. Aydogmus and A. H. Tor, "A modified multiple shooting algorithm for parameter estimation in ODEs using adjoint sensitivity analysis," *Applied Mathematics and Computation*, vol. 390, 2021.

[17] B. Wang and W. Enright, "Parameter estimation for odes using a cross-entropy approach," *SIAM Journal on Scientific Computing*, vol. 35, no. 6, 2013.

[18] J. O. Ramsay, G. Hooker, D. Campbell, and J. Cao, "Parameter estimation for differential equations: a generalized smoothing approach," *Journal of the Royal Statistical Society: Series B*, vol. 69, no. 5, pp. 741–796, 2007.

[19] A. A. Poyton, M. S. Varziri, K. B. McAuley, P. J. McLellan, and J. O. Ramsay, "Parameter estimation in continuous-time dynamic models using principal differential analysis,"

Computers & Chemical Engineering, vol. 30, no. 4, pp. 698–708, 2006.

[20] L. Edsberg and P.-A. Wedin, "Numerical tools for parameter estimation in ode-systems," *Optimization Methods and Software*, vol. 6, no. 3, pp. 193–217, 1995.

[21] A. Gábor and J. R. Banga, "Robust and efficient parameter estimation in dynamic models of biological systems," *BMC Systems Biology*, vol. 9, no. 1, pp. 74–25, 2015.

[22] W. S. McCulloch and W. Pitts, "A logical calculus of the ideas immanent in nervous activity," *Bulletin of Mathematical Biophysics*, vol. 5, no. 4, pp. 115–133, 1943.

[23] N. Yadav, A. Yadav, and M. Kumar, *An Introduction to Neural Network Methods for Differential Equations*, Springer, Berlin, Germany, 2015.

[24] I. A. Basheer and M. Hajmeer, "Artificial neural networks: fundamentals, computing, design, and application," *Journal of Microbiological Methods*, vol. 43, no. 1, pp. 3–31, 2000.

[25] J. Schmidhuber, "Deep learning in neural networks: an overview," *Neural Networks*, vol. 61, pp. 85–117, 2015.

[26] I. Goodfellow, Y. Bengio, A. Courville, and Y. Bengio, *Deep Learning*, Vol. 1, MIT Press Cambridge, Cambridge, UK, 2016.

[27] S. Dong, P. Wang, and K. Abbas, "A survey on deep learning and its applications," *Computer Science Review*, vol. 40, Article ID 100379, 2021.

[28] P. Dixit and S. Silakari, "Deep learning algorithms for cybersecurity applications: a technological and status review," *Computer Science Review*, vol. 39, Article ID 100317, 2021.

[29] S. Minaee, Y. Y. Boykov, F. Porikli, A. J. Plaza, N. Kehtarnavaz, and D. Terzopoulos, "Image segmentation using deep learning: a survey," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2021.

[30] J. Bruna and L. Dec, *Mathematics of Deep Learning*, Courant Institute of Mathematical Science, NYU, 2018.

[31] M. Paliwal and U. A. Kumar, "Neural networks and statistical techniques: a review of applications," *Expert Systems with Applications*, vol. 36, no. 1, pp. 2–17, 2009.

[32] C. G. Villegas-Mier, J. Rodriguez-Resendiz, J. M. Álvarez-Alvarado, H. Rodriguez-Resendiz, A. M. Herrera-Navarro, and O. Rodríguez-Abreo, "Artificial neural networks in mppt algorithms for optimization of photovoltaic power systems: a review," *Micromachines*, vol. 12, no. 10, 1260 pages, 2021.

[33] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," in *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, pp. 249–256, Chia Laguna Resort, Sardinia, Italy, May 2010.

[34] V. Dua, "An artificial neural network approximation based decomposition approach for parameter estimation of system of ordinary differential equations," *Computers & Chemical Engineering*, vol. 35, no. 3, pp. 545–553, 2011.

[35] V. Dua and P. Dua, "A simultaneous approach for parameter estimation of a system of ordinary differential equations, using artificial neural network approximation," *Industrial & Engineering Chemistry Research*, vol. 51, no. 4, pp. 1809–1814, 2012.

[36] T. T. Dufera, "Deep neural network for system of ordinary differential equations: vectorized algorithm and simulation," *Machine Learning with Applications*, vol. 5, Article ID 100058, 2021.

[37] J. Bradbury, R. Frostig, P. Hawkins et al., "JAX: Composable Transformations of Python+NumPy Programs," 2018, http://github.com/google/jax.

[38] L. Balles and P. Hennig, "Dissecting Adam: the sign, magnitude and variance of stochastic gradients," in *Proceedings of the International Conference on Machine Learning*, pp. 404–413, PMLR, 2018.

[39] D. P. Kingma and J. Ba, "Adam: a method for stochastic optimization," 2014, https://arxiv.org/abs/1412.6980.

[40] A. G. Baydin, B. A. Pearlmutter, A. A. Radul, and J. M. Siskind, "Automatic differentiation in machine learning: a survey," *Journal of Machine Learning Research*, vol. 18, 2018.

[41] B. Winkel and G. B. Gustafson, "Differential equations course materials," 2017, https://www.simiode.org/resources/3892.