Neural Network Representation for Ordinary Differential Equations

Anna Golovkina[0000-0002-8906-5227] and Vladimir Kozynchenko[0000-0003-1011-2455]

Saint Petersburg State University, Saint Petersburg 199034, Russia
a.golovkina@spbu.ru

Abstract. Ordinary differential equations (ODEs) are a well-known universal tool to describe dynamic processes in physics, chemistry, biology, etc. Nowadays, extensive development of sensing techniques and IoT devices resulted in the accumulation of a large amount of data that brought to the front edge the pure data-driven methods to describe and predict real-world processes. However, pure data-driven methods are unreliable in applications with higher requirements for generalization and the ability to interpret the results. To overcome the resulting problems, prior domain knowledge expressed in the form of ODEs should be incorporated into the data learning tool. For this reason, the paper suggests a polynomial neural network that, by design, corresponds to a system of ODEs. The proper initialization of the PNN transfers the knowledge of the specific dependencies and coefficient values from ODEs to the PNN. The paper introduces an iterative procedure for constructing a general solution for a nonstationary system of ODEs in the polynomial form. Matrix coefficients of this decomposition initialize PNN. This method is illustrated for generalized Lotka-Volterra ordinary differential equations demonstrating chaotic behavior.

Keywords: Physics informed neural network, polynomial learning, system identification, Taylor mapping, neural ODE

1 Introduction

Differential equations are the fundamental language of all physical laws expressing the intrinsic nature of biological, physical, and chemical processes. Aside from these, differential equations (ODEs) are an essential tool in describing the behavior of complex systems. In other words, they build the foundation for modeling process dynamics, design control systems, and short-term and mid-term forecasting. These explain a recent interest in the governing equations reconstruction from the experimental data.

Despite the universality of a system described by ordinary differential equations, it could be hard to build a full-scale model in real systems scenarios considering the interactions of many parameters. So, generally ODE model is a simplification that captures only fundamental features of the complex physical process.

On the other hand, due to the intense development and usage of machine learning (ML) algorithms, they tend to be a framework for building predicting models in different areas. Although ML approaches possess a practical perspective and have prior

success, most cannot extract interpretable information from the data and do not work well on examples different from training data. This ambiguity has made it problematic for machine learning systems to be adopted in sensitive domains, where their value could be tremendous, such as industrial production, pharmaceutics, and so on [12].

These challenges led to the development of hybrid models, such as physics-inspired neural networks (PINN), aimed to blend black-box predicting models and domain information expressed as ordinary/partial differential equations or conservation laws [12]. Using differential equations in the neural network basis allows combining ODEs with neural network approaches such as learning and automatic feature extraction and using them to create new forecasting tools with improved extrapolation and interpretability.

PINN often follows one of two ways to incorporate the governing physics equations: automatic differentiation of the loss function or classic numerical methods for solving a differential equation in the network design. For instance, [14] suggests the energy conservation law rather than the conventional L_1-L_2 regularization, [7, 8] use Hamilton's equations to maintain the underlying system's symplectic structure, [13–15] demand that the NN output meets the Euler-Lagrange equation. Followers of the second technique utilize finite-difference or Runge-Kutta methods in neural networks [4, 18]. For instance, the basic Euler discretization of autonomous ODEs is similar to residual NNs [3, 16, 19]. Runge-Kutta schemes are closely related to recurrent NNs [10], while convolutional NNs are comparable with the multigrid method for PDEs [9].

To sum up, PINN is an analog to ODE (PDE) capable of solving forward and inverse problems with extensive identification power through online adaptation from new incoming data. In contrast to the existing methods, the strategy presented in this research offers a way to incorporate past knowledge about dynamical systems into NN. It is based on the Taylor mapping approach for solving ODEs and allows direct transfer of ODE to neural polynomial architecture (PNN) [1, 2, 6, 11]. The algorithm for initializing the nonstationary PNN weights is presented in the study, along with the results of using it to solve forward and inverse problems for the specified ODEs.

The remainder of this article is arranged as follows: the problem statement is presented in Section 2, and the algorithm description for PNN initialization using ODE is provided in Section 3. The outcomes of numerically solving the forward and inverse problems for generalized Lotka-Volterra equations are shown in Section 4.

2 Problem statement

Consider the following set of ordinary differential equations:

$$\frac{d\mathbf{X}}{dt} = F(t, \mathbf{X}). \tag{1}$$

Here t is an independent variable (time), $\mathbf{X} \in \mathbb{R}^n$ is a n-dimensional state of the system, F is a known vector field that maps $F: \mathbb{R} \times \mathbb{R}^n \to \mathbb{R}^n$ and implies the system's past knowledge.

We aim to design a neural network's building block incorporating ODEs (1). This block is going to specify the underlying ODEs' dynamics and propagate the system state in time, i.e., solve (1) with initial condition $\mathbf{X}(t_0) = \mathbf{X}_0$:

$$\mathbf{X}(t, \mathbf{X}_0) = \mathcal{M}_F^t(\mathbf{X}_0) = \mathbf{X}_0 + \int_{t_0}^{t} F\left(\tau, \mathbf{X}(\tau)\right) d\tau. \tag{2}$$

Mapping $\mathcal{M}_F^t$ defines system's state propagator from time moment t_0 to t which is the core of ODE layer in neural network. Approximating $\mathcal{M}_F^t$ with traditional solvers is based on dividing the interval $[t_0, t]$ into several steps and numerical computing integral in (2). For example, the simplest iterative procedure is explicit Euler method where a single step takes the following form

$$\mathbf{X}(t_{k+1}) = \mathbf{X}(t_k) + hF\big(t_k, \mathbf{X}(t_k)\big). \tag{3}$$

The iterative formula (3) can also be used to represent advanced numerical integration techniques, where the state vector values at the previous time step are typically not multiplicatively separable in the nonlinear function F. This indicates that numerical computation should be repeated for each new initial condition when solving a forward problem for (1). On the other hand, it also complicates the identification of F in the inverse problem for (1).

The paper aims to build an expression of the flow $\mathcal{M}_F^t(\mathbf{X}_0)$ in the polynomial form

$$\mathbf{X}(t) = \sum_{i=1}^{m} R^{1i}(t, t_0)\mathbf{X}_0^{[i]}, \tag{4}$$

show the procedure to construct unknown $R^{1i}(t, t_0)$, $i = \overline{1..m}$ matrices based on the system (1) and a corresponding neural network's layer. Here $\mathbf{X}^{[k]}$ is k-th Kronecker power of vector $\mathbf{X}$ (means $\mathbf{X} \otimes \mathbf{X}^{[k-1]}$ after removing duplicate terms). The advantage of representation (4) is that it is linear with respect to the matrices $R^{1i}(t, t_0)$.

3 PNN design

3.1 General architecture

The length of the time interval $T = [t_0, t]$ and the number of nonlinear terms m affect how accurate the approximation (4) is. A more precise numerical ODE solution is obtained by increasing m and subdividing interval T into smaller increments. The flow $\mathcal{M}_F^t(\mathbf{X}_0)$ is identical to a network made up of a series of polynomial neurons displayed in Figure 1. If (1) is autonomous, PNN can be thought of as a recurrent neural network with shared weights because its weight matrices, $R^{1i}\big(t_j, t_{j-1}\big) \equiv W_i$, depend only on the size of a time step $t_j - t_{j-1}$. In contrast, for non stationary system, weights $R^{1i}\big(t_j, t_{j-1}\big) \equiv W_i(t)$ are time dependent.

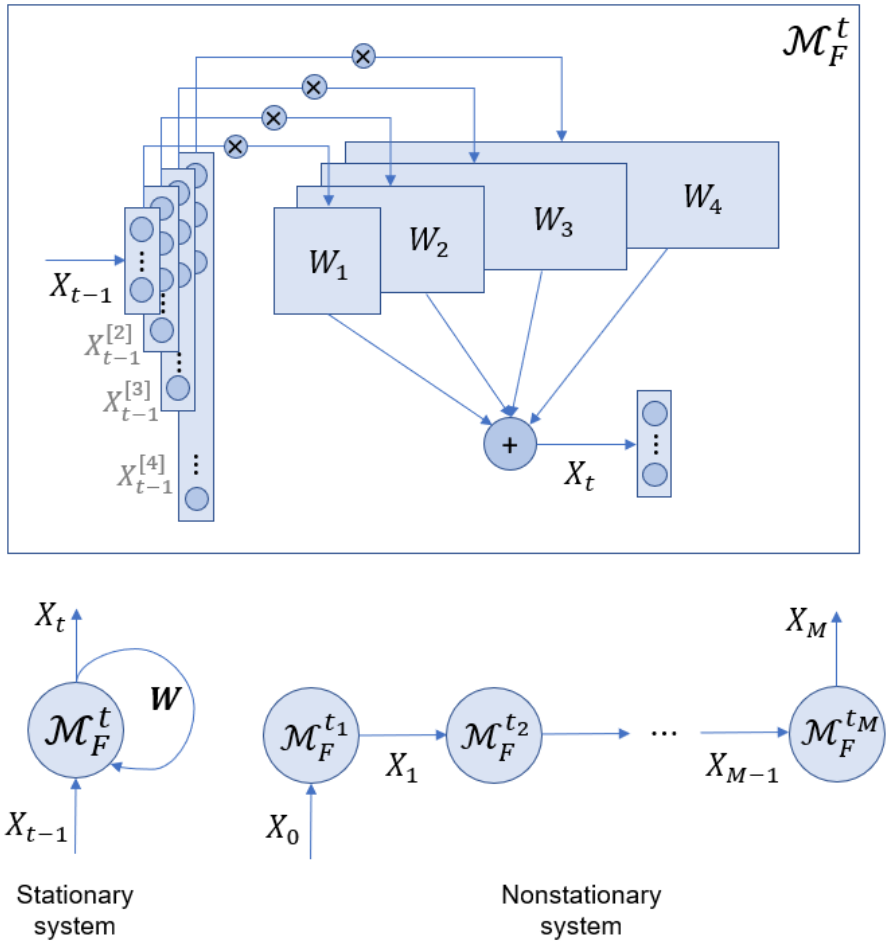

Fig. 1. Internal structure of polynomial neuron (4th order of nonlinearity) and construction of PNN for stationary and nonstationary systems

By design, PNN represents a general solution of ordinary differential equations up to the required degree of accuracy (4). To include the a priory knowledge about ODE into PNN, this is necessary to calculate weight matrices $R^{1i}(t_j, t_{j-1}),\ i = \overline{1..m},\ j = \overline{1..M}$ directly from (1).

3.2 PNN initialization

We clarify the proposed algorithm for the case, when $F(t, \mathbf{X}) \in C^{\infty}(\mathbb{R} \times \mathbb{R}^n)$, so it can be expanded in Taylor series in the neighborhood of $(t_0, \mathbf{X}_0)$. We consider the finite

number of terms in the expansion and for the following inferences consider a non-stationary nonlinear system of ODEs with a polynomial r.h.s.

$$\frac{d\mathbf{X}}{dt} = \sum_{k=0}^{N} \mathrm{P}^{1k}(t)\mathbf{X}^{[k]} \tag{5}$$

Let us introduce the notation for the vector of phase moments up to N-th order $\mathbf{X}^N = \left[\mathbf{X}, \mathbf{X}^{[2]}, \ldots \mathbf{X}^{[N]}\right]^T$, where $\mathbf{X}^{[m]}$ is m-th Kronecker's degree of vector $\mathbf{X}$. With the help of the axillary vector $\mathbf{X}^N$ it may be shown [2, 6] that the system (5) can be transformed to a linear system

$$\frac{d\mathbf{X}^N}{dt} = \mathrm{P}^N(t)\mathbf{X}^N, \tag{6}$$

by discarding terms of order higher than N, where

$$\mathrm{P}^N(t) = \begin{pmatrix} P^{11} & P^{12} & \ldots & P^{1N} \\ 0 & P^{22} & \ldots & P^{2N} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \ldots & P^{NN} \end{pmatrix}. \tag{7}$$

Now, the solution of the linear system (6) can be found according to the formula

$$\mathbf{X}^N = \mathrm{R}^N(t, t_0)\mathbf{X}_0^N, \tag{8}$$

where $\mathbf{X}_0^N = \mathbf{X}^N(t_0)$, $\mathrm{R}^N(t_0, t_0) = E$, E is an identity matrix of corresponding dimension.

Matrix R^N has a similar-to-(7) block structure

$$\mathrm{R}^N(t) = \begin{pmatrix} R^{11} & R^{12} & \ldots & R^{1N} \\ 0 & R^{22} & \ldots & R^{2N} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \ldots & R^{NN} \end{pmatrix}.$$

To find the block elements of the matrix R^N, we represent the linear system (8) as a system consisting of N subsystems of linear differential equations

$$\begin{cases} \frac{d\mathbf{X}}{dt} = \mathrm{P}^{11}(t)\mathbf{X} + \mathrm{P}^{12}(t)\mathbf{X}^{[2]} + \cdots + \mathrm{P}^{1N}(t)\mathbf{X}^{[N]}, \\ \frac{d\mathbf{X}^{[2]}}{dt} = \mathrm{P}^{22}(t)\mathbf{X}^{[2]} + \cdots + \mathrm{P}^{2N}(t)\mathbf{X}^{[N]}, \\ \cdots \\ \frac{d\mathbf{X}^{[N]}}{dt} = \mathrm{P}^{NN}(t)\mathbf{X}^{[N]}. \end{cases} \tag{9}$$

We will solve this system starting from the last equation as it is linear

$$\frac{d\mathbf{X}^{[N]}}{dt} = \mathrm{P}^{NN}(t)\mathbf{X}^{[N]}$$

and the solution to this linear system can be represented as

$$\mathbf{X}^{[N]}(t) = \mathrm{R}^{NN}(t, t_0)\mathbf{X_0}^{[N]}.$$

To find the matrix R^{NN}, as well as all other diagonal matrices R^{ii}, let us consider a linearized first equation of the system (9)

$$\frac{d\mathbf{X}}{dt} = \mathrm{P}^{11}(t)\mathbf{X}.$$

The solution can be found analytically or numerically in the form

$$\mathbf{X}(t) = \mathrm{R}^{11}(t, t_0)\mathbf{X_0}, \qquad \mathbf{X_0} = \mathbf{X}(t_0)$$

For example, if $\mathrm{P}^{11}(t) \equiv \mathrm{P}^{11} = \text{const}$, $\mathrm{R}^{11}(t, t_0) = \exp\big((t - t_0)\mathrm{P}^{11}\big)$.

This is easy to see here, that matrix R^{ii} is represented as

$$\mathrm{R}^{ii}(t, t_0) = \big(\mathrm{R}^{11}(t, t_0)\big)^{[i]}. \tag{10}$$

Continuing successive solving system (9), let us consider the penultimate subsystem

$$\frac{d\mathbf{X}^{[N-1]}}{dt} = \mathrm{P}^{N-1N-1}(t)\mathbf{X}^{[N-1]} + \mathrm{P}^{N-1N}(t)\mathbf{X}^{[N]}.$$

Substituting into this system the representation for $X^{[N]}$, we get again a linear but non-homogeneous system with respect to $X^{[N-1]}$

$$\frac{d\mathbf{X}^{[N-1]}}{dt} = \mathrm{P}^{N-1N-1}(t)\mathbf{X}^{[N-1]} + \mathrm{P}^{N-1N}(t)\mathrm{R}^{NN}(t, t_0)\mathbf{X_0}^{[N]}.$$

The solution of this system can be written using the Cauchy formula

$$\mathbf{X}^{[N-1]}(t) = \mathrm{R}^{N-1N-1}(t, t_0)\mathbf{X_0}^{[N-1]} + \mathrm{R}^{NN}(t, t_0)\mathbf{X_0}^{[N]},$$

where

$$\mathrm{R}^{N-1N-1}(t, t_0) = \int_{t_0}^{t} \mathrm{R}^{N-1N-1}(t, \tau)\mathrm{P}^{N-1N}(t)\mathrm{R}^{NN}(\tau, t_0)d\tau.$$

We continue the indicating iterative procedure and assume that the subsystem

$$\frac{d\mathbf{X}^{[i]}}{dt} = \mathrm{P}^{ii}(t)\mathbf{X}^{[i]} + \cdots + \mathrm{P}^{iN}(t)\mathbf{X}^{[N]} \tag{11}$$

has a solution

$$\mathbf{X}^{ii}(t) = \sum_{k=i}^{N} \mathrm{R}^{ik}(t, t_0)\mathbf{X_0}^{[k]}, \tag{12}$$

where

$$\mathrm{R}^{ij}(t, t_0) = \sum_{l=i+1}^{j} \int_{t_0}^{t} \mathrm{R}^{ii}(t, \tau)\mathrm{P}^{il}(t)\mathrm{R}^{lj}(\tau, t_0)d\tau. \tag{13}$$

Let us show by induction the validity of formulas (11)-(13) for arbitrary i. For this, we consider the subsystem

$$\frac{d\mathbf{X}^{[i-1]}}{dt} = \mathrm{P}^{i-1i-1}(t)\mathbf{X}^{[i-1]} + \cdots + \mathrm{P}^{i-1N}(t)\mathbf{X}^{[N]}$$

and substitute there the representations for $X^{[i]}, \dots, X^{[N]}$ from the inductive assumption (12)-(13). By this, we get a linear non-homogeneous system with respect to $X^{[i-1]}$

$$\begin{aligned}&\frac{d\mathbf{X}^{[i-1]}}{dt} = \mathrm{P}^{i-1i-1}(t)\mathbf{X}^{[i-1]} + \mathrm{P}^{i-1i}(t)\sum_{k=i}^{N}\mathrm{R}^{i-1k}(t,t_0)\mathbf{X_0}^{[k]} + \\ &\mathrm{P}^{i-1i+1}(t)\sum_{k=i+1}^{N}\mathrm{R}^{i-1k}(t,t_0)\mathbf{X_0}^{[k]} + \cdots + \mathrm{P}^{i-1N-1}(t)\sum_{k=N-1}^{N}\mathrm{R}^{i-1k}(t,t_0)\mathbf{X_0}^{[k]} \\ &+\mathrm{P}^{i-1N}(t)\mathrm{R}^{i-1N}(t,t_0)\mathbf{X_0}^{[N]} = \mathrm{P}^{i-1i-1}(t)\mathbf{X}^{[i-1]} + \\ &\sum_{l=i}^{i+1}\mathrm{P}^{i-1l}(t)\mathrm{R}^{lj}(t,t_0)\mathbf{X_0}^{[i+1]} + \cdots + \sum_{l=i}^{N-1}\mathrm{P}^{i-1l}(t)\mathrm{R}^{lj}(t,t_0)\mathbf{X_0}^{[N-1]} + \\ &\sum_{l=i}^{N}\mathrm{P}^{i-1l}(t)\mathrm{R}^{lj}(t,t_0)\mathbf{X_0}^{[N]}.\end{aligned} \tag{14}$$

Introducing the notation

$$\mathrm{Q}_{\mathrm{i-1j}}(t,t_0) = \sum_{L=i}^{j}\mathrm{P}^{i-1l}(t)\mathrm{R}^{lj}(t,t_0), \tag{15}$$

we can rewrite a linear non-homogeneous system (14) in a shorter way

$$\frac{d\mathbf{X}^{[i-1]}}{dt} = \mathrm{P}^{i-1i-1}(t)\mathbf{X}^{[i-1]} + \sum_{j=i}^{N}\mathrm{Q}_{\mathrm{i-1j}}(t,t_0)\mathbf{X_0}^{[j]}.$$

This system has a solution

$$\begin{aligned}&\mathbf{X}^{[i-1]}(t,t_0) = \mathrm{R}^{i-1i-1}(t,t_0)\big(\mathbf{X_0}^{[i-1]} + \\ &+\int_{t_0}^{t}\left(\mathrm{R}^{i-1i-1}(\tau,t_0)\right)^{-1}\sum_{j=i}^{N}\mathrm{Q}_{\mathrm{i-1j}}(t,t_0)\mathbf{X_0}^{[j]}d\tau\Bigg) = \\ &\mathrm{R}^{i-1i-1}(t,t_0)\mathbf{X_0}^{[i-1]} + \sum_{j=i}^{N}\left(\int_{t_0}^{t}\mathrm{R}^{i-1i-1}(t,\tau)\mathrm{Q}_{i-1j}(\tau,t_0)d\tau\right)\mathbf{X_0}^{[j]}.\end{aligned}$$

Denoting

$$\mathrm{R}^{i-1j}(t,t_0) = \int_{t_0}^{t}\mathrm{R}^{i-1i-1}(t,\tau)\mathrm{Q}_{\mathrm{i-1j}}(\tau,t_0)d\tau$$

and using the introduced in (15) representation for $\mathrm{Q}_{\mathrm{i-1j}}(t,t_0)$, we get

$$\mathrm{R}^{i-1j}(t,t_0) = \sum_{l=i-1+1}^{j}\int_{t_0}^{t}\mathrm{R}^{i-1i-1}(t,\tau)\mathrm{P}_{\mathrm{i-1l}}(\tau)\mathrm{R}^{lj}(\tau,t_0)d\tau.$$

These prove the validness of representations (11)-(13) and give an iterative procedure to compute the matrices R^{1k} and find solution approximation in the form (4).

As a result, this section offers an iterative method for obtaining the equation (5) solution in the polynomial form (4) and creating the associated PNN that may be used to solve both forward and inverse ODE problems.

To solve both forward and inverse ODE problems, this section suggests an iterative method for finding the solution to (5) in polynomial form (4) and constructing the corresponding PNN.

4 Numerical experiments

Let us consider generalized Lotka-Volterra equations representing two-predator one-prey population dynamics in an ecosystem to illustrate the proposed method of translating ODE to a neural network model. This system exhibits chaotic behavior for a specific set of parameters a, b, c. Thus, it is highly sensitive to initial conditions [17]. The following ODEs represent the considered Lotka-Volterra model:

$$\begin{aligned} \dot{x_1} &= x_1 - x_1 x_2 + c x_1^2 - a x_1^2 x_3, \\ \dot{x_2} &= -x_2 + x_1 x_2 \\ \dot{x_3} &= -b x_3 + a x_1^2 x_3, \end{aligned} \tag{16}$$

where a, b, c are constant positive parameters of the system and $x_i, i = 1,2,3$ are the state variables.

The existence of a stationary solution and the asymptotic behavior of the solution has been investigated in [5]. The parameters a, b, c are chosen in such a way that species densities in the model converge to an extinction steady state, although this fact has no bearing on the demonstration of the efficacy of the method described in the study for solving both forward and inverse problems. It is explained by basing the analysis on the time interval where the solutions noticeably stand above zero.

4.1 Forward problem

The forward problem formulated for ODEs means finding its numerical solution when the full knowledge of the system is available (structure, parameters, initial conditions). The system (16) demonstrates a chaotic behavior when the parameter value is taken as $a = 2.9851$, $b = 3$, $c = 2$. The initial values of generalized Lotka-Volterra chaotic system (16) are taken as $x_1(0) = 16$, $x_2(0) = 24$, $x_3(0) = 18$.

Let us show the ability of the method presented in this paper to approximate (16) solution. We will compare numerical solutions obtained with traditional solver *lsoda* and TM-solver at the time interval $T = [0; 2]$. We should construct a polynomial solution containing at least three matrices R^{1i}, $i = \overline{1..3}$, since the right-hand side of (16) has a maximum order of nonlinearity equal to three. Moreover, the equation (16) is autonomous, so the matrices R^{1i} don't depend on time and can be computed once for the fixed time step length.

According to numerical simulations, the accuracy of the solution improves marginally with increasing order of non-linearity for this system. Thus, we now concentrate on illustrating accuracy dependence on the time step.

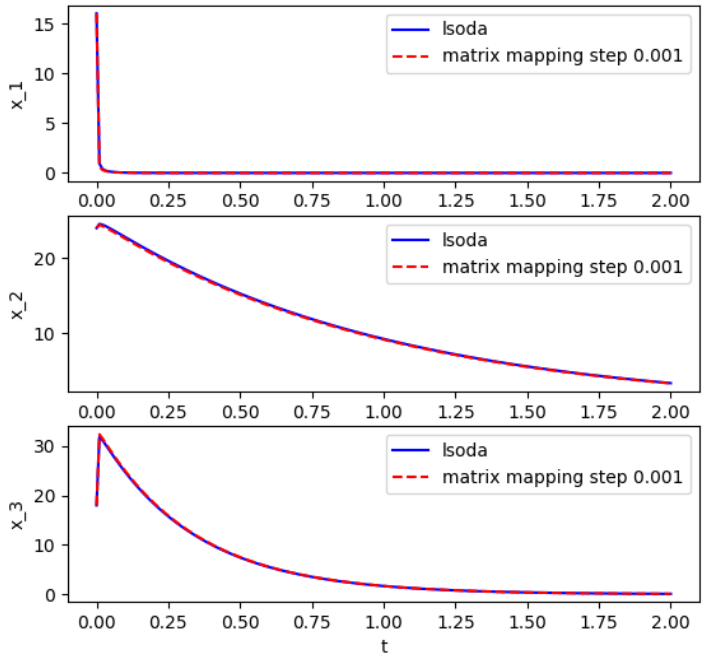

Fig. 2. Generalized Lotka-Volterra system solution with *lsoda* and matrix mapping for 0.001 s time step

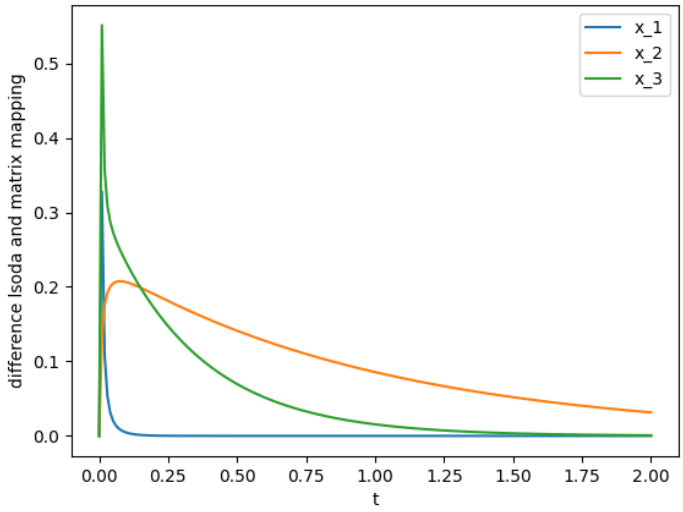

Fig. 3. The difference between a numerical solution with *lsoda* and matrix mapping for 0.001 s time step of Lotka-Volterra system

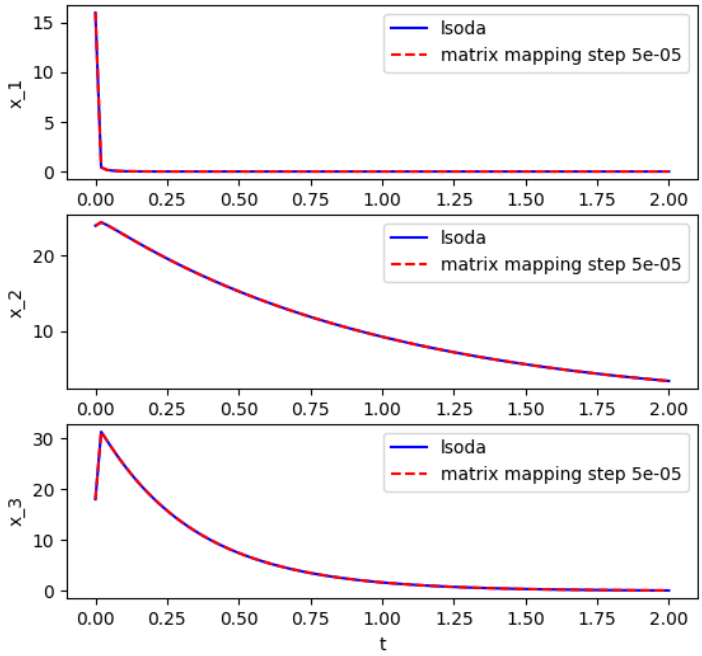

Fig. 4. The difference between numerical solution with *lsoda* and matrix mapping for $5 \cdot 10^{-5}$ s time step of Lotka-Volterra system

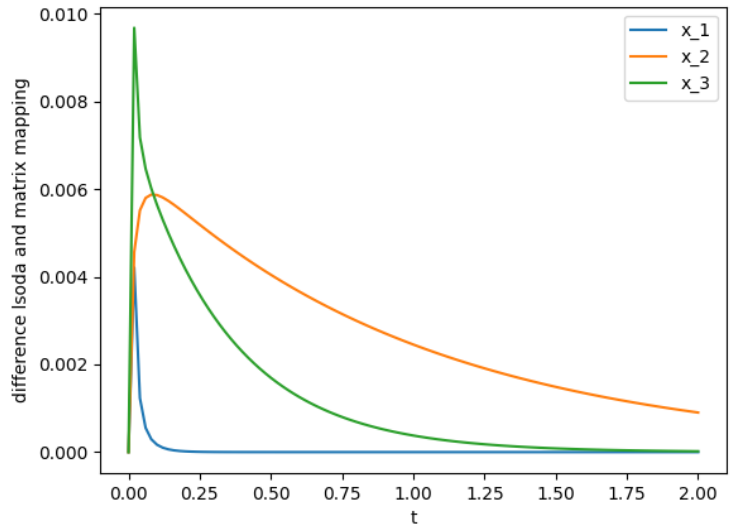

Fig. 5. The difference between numerical solution with *lsoda* and matrix mapping for $5 \cdot 10^{-5}$ s time step of Lotka-Volterra system

Figure 2 shows system (16) trajectories $x_1(t)$, $x_2(t)$, $x_3(t)$ computed with traditional *lsoda* solver and the introduced matrix mapping approach for the time step 0.001 and third order of nonlinearity. The matrices $R^{1i}\ i = \overline{1..3}$ computed according the proposed algorithm are described by the formulas (17). Figure 3 illustrates the difference between the numerical solutions obtained with the two considered methods.

$$
\begin{aligned}
R^{11} &= \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0.999 & 0 \\ 0 & 0 & 0.997 \end{pmatrix} \\
R^{12} &= \begin{pmatrix} 0.002 & -0.001 & 0 & 0 & 0 & 0 \\ 0 & 0.001 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} \\
R^{13} &= \begin{pmatrix} 4 \cdot 10^{-6} & -3.5 \cdot 10^{-6} & -0.003 & 5 \cdot 10^{-7} & \dots & 0 \\ 0 & 1.5 \cdot 10^{-6} & 0 & -5 \cdot 10^{-7} & \dots & 0 \\ 0 & 0 & 0.003 & 0 & \dots & 0 \end{pmatrix}
\end{aligned}
\tag{17}
$$

Comparing the graphics depicting the divergence between the numerical solutions and the system trajectories, we can conclude that divergence is maximal for the maximal absolute values of the solution derivative. *Lsoda* is an adaptive solver that uses the initial integration step 10^{-5} s.

Let us illustrate the convergence of the matrix mapping solution by decreasing the time-step to the *lsoda* solution. Figure 4 and 5 shows system (16) trajectories $x_1(t)$, $x_2(t)$, $x_3(t)$ and difference with *lsoda* correspondingly for the time step $5 \cdot 10^{-5}$ s and third order of nonlinearity.

Comparing graphics in Figures 3 and 5 show that the maximal divergence between the solutions obtained with the traditional and proposed method is decreased by 50 times with the reduction of the time-step by 20 times. That means, if necessary, we can get a more accurate approximation of ODE with PNN initialized with matrices (17) that don't depend on ODEs' initial values and thus can reflect the system dynamics in a whole range of input variations.

4.2 Inverse problem

Let us now illustrate the ability of PNN to solve the inverse problem for the given ODE (16) — learning a dynamic model from the data. We consider two cases:

- the structure of the system is known, but the exact values of parameters a, b, c are unavailable;
- no knowledge about the system is available.

Training data set

Here we consider the system (16) with parameters $a = 10, b = 11, c = 5$. The training data are a single trajectory consisting of 1000 points simulated by numerical solving equation (16) by *lsoda* routine with initial condition $\mathbf{X}_0 = [16,10,20]^T$. Time discretization is 5 ms. The aim is to train PNN from these data to represent the underlying dynamical process accurately. Then, the learned PNN can be built in as a block to deep learning architectures.

A. Knowledge-based learning

In the first case, we consider learning with imperfect knowledge about a system. We suppose that the parameters of the Lotka-Volterra system are not known in advance, but the system structure is available. As the system structure is known, we can presume that parameters values are $a = 2.9851$, $b = 3$, $c = 2$. Thus, this incorrect system is our knowledge. Then, we can represent ODE according to the described algorithm up to the 3rd order of nonlinearity. The matrices $R^{1i}, i = 1..3$ are found via the formulas (17) already computed for the forward problem. We use them to initialize the PNN weights. Further, they are fine-tuned during training to correspond to the real a, b, c values.

We produce test data by numerically solving (16) with different initial conditions ($\mathbf{X}_0 = [6,2,4]^T$ and $\mathbf{X}_0 = [10,6,8]^T$) to verify the trained PNN model's generalization power to correctly describe predator-prey dynamics.

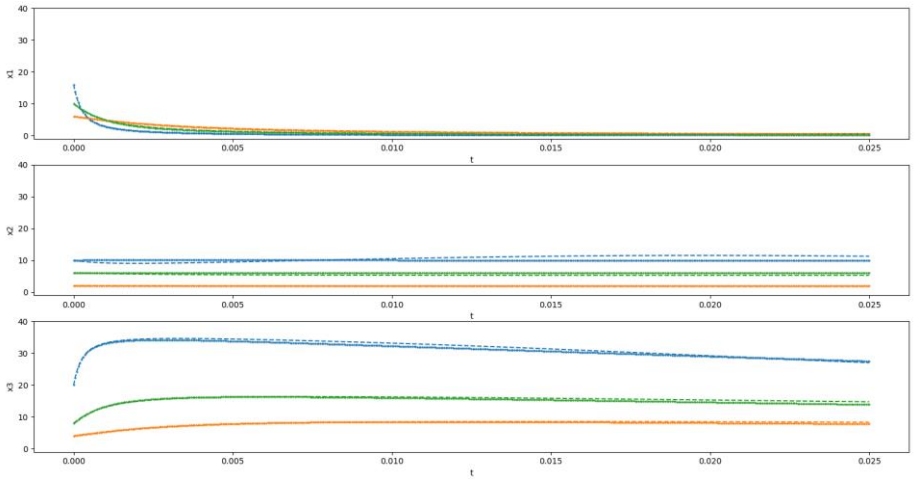

Fig. 6. Reconstructed predator-prey dynamics (dashed lines) with initialized PNN (blue dot line – training data (500 samples, 100 epochs), green and orange dot lines – test data)

Figure 6 shows that even when our initial guess about parameter values a, b, c is quite far from their original values, PNN was able to adapt during learning only with 500 training data points and 100 epochs of training. The PNN's predictions for the test data set simulated with new initial values justify its ability to recover correct system dynamics (cf. green and orange lines in Figure 6).

B. Learning without knowledge

The second case supposes that we don't know anything about the system equations but still want to construct the ODE model on a PNN basis using the same training data as for the case A. Figure 7 shows the predictions done by the trained PNN for train and test data initial values. Compared to knowledge-based learning, learning without knowledge requires more training data to converge its predictions to the actual dynamics.

However, it is seen from Figure 7 that there are still deviations in the second species (x_2) predictions leaving room for further improvement of PNN training from zero weights. One possibility is introducing regularization that aims to minimize the number of non-zero coefficients in the weight matrices during training.

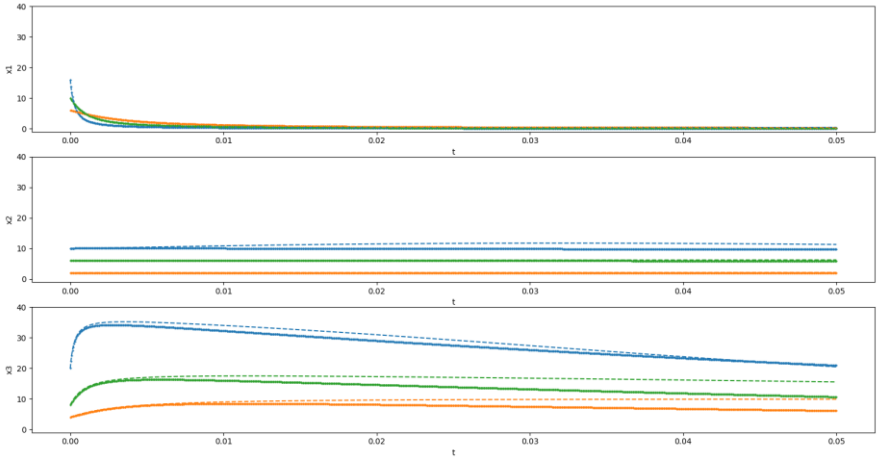

Fig. 7. Reconstructed predator-prey dynamics (dashed lines) with not initialized PNN (blue dot line – training data (1000 samples, 100 epochs), green and orange dot lines – test data)

Comparison results

Standard mean squared error is used as a loss function for training and validation, additionally, mean absolute and mean absolute percentage errors estimate the accuracy of the prediction. Table 1 gathers the indicated metrics for cases A and B during training and validation. Figure 8 compares epoch loss for PNN trained with 500 and 1000 data points with and without initialization.

Table 1. Comparison metrics for train and test cases

Experiment	MSE	MAE	MAPE
case A/500/Train	9.7814e-04	0.0166	2.2911
case A/500/Val 1	3.2742e-06	0.0012	0.0430
case A/500/Val 2	3.1238e-05	0.0031	0.1343
case A/1000/Train	4.1810e-04	0.0095	2.1466
case A/1000/Val 1	1.5368e-06	8.0159e-04	0.0268
case A/1000/Val 2	7.8331e-06	0.0014	0.1053
case B/500/Train	0.0013	0.0209	2.7056
case B/500/Val 1	5.8122e-06	0.0016	0.0319
case B/500/Val 2	4.4473e-05	0.0034	0.0918
case B/1000/Train	4.5774e-04	0.0097	2.0666

As expected, knowledge-based learning gives lower loss function and accuracy metrics values and requires less data for training. Learning from scratch in the same

conditions performs worse, but still, PNN replicates the actual system dynamics even for unseen initial conditions.

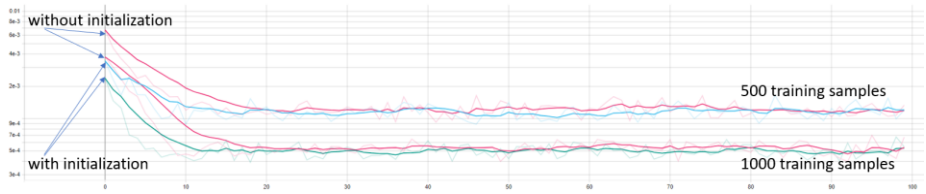

Fig. 8. Comparison of loss value decrease during 100 epochs training with 500 and 1000 data points for initialized PNN (case A) and not initialized PNN (case B)

5 Conclusion

This study addresses the issue of incorporating prior knowledge about the dynamical system into a neural network using an approach different from those adopted in the literature. It allows direct transferring of ODEs as initial weights to neural polynomial architecture (PNN) based on the Taylor mapping technique for solving ODEs. Using differential equations models in the neural networks allows ODEs to be combined with neural network approaches such as learning and automatic feature extraction and exploit them in building new predictive tools with enhanced interpretability and extrapolating ability. The paper presents an iterative algorithm for PNN's initial weights construction from the given or supposed underlying ODEs. The algorithm's convergence is illustrated by numerical solutions to forward problems for the generalized Lotka-Volterra equation.

On the other hand, PNN can also be an effective tool to tackle inverse problems where the equations are unknown, but we want the neural model to describe the dynamics in the data correctly. In this case, PNN can be trained from scratch, or if we have an assumption about the system equations, we can use this information to initialize PNN. Then, the PNN's weights are fine-tuned using NN learning algorithms. The paper compares the convergence of learning with and without initialization. As expected, knowledge-based learning brings lower loss values while at the same time requiring less data for training. Learning from scratch in the same conditions performs worse, but the authors suppose it can be improved by introducing regularization for weights sparsity. That would be the topic of further consideration.

Acknowledgment. The research presented in this paper was funded by Saint Petersburg State University, project ID: 90317740.

References

1. Andrianov, S.: Symbolic computation of approximate symmetries for ordinary differential equations. Mathematics and Computers in Simulation. 57, 3-5, 147–154 (2001).

2. Andrianov, S.N.: Dynamical modeling of control systems for particle beams. Saint Petersburg State University, SPb (2004).
3. Chen, R.T.Q. et al.: Neural ordinary differential equations. In: Proceedings of the 32nd international conference on neural information processing systems. pp. 6572–6583 Curran Associates Inc., Red Hook, NY, USA (2018).
4. Dufera, T.T.: Deep neural network for system of ordinary differential equations: Vectorized algorithm and simulation. Machine Learning with Applications. 5, 100058 (2021). https://doi.org/10.1016/j.mlwa.2021.100058.
5. Elsadany, A.A. et al.: Dynamical analysis, linear feedback control and synchronization of a generalized Lotka-Volterra system. International Journal of Dynamics and Control. 6, 1, 328–338 (2017). https://doi.org/10.1007/s40435-016-0299-x.
6. Golovkina, A., Kozynchenko, V.: Parametric identification of a dynamical system with switching. In: Gervasi, O. et al. (eds.) Computational science and its applications – ICCSA 2022 workshops. pp. 557–569 Springer International Publishing, Cham (2022).
7. Greydanus, S. et al.: Hamiltonian neural networks. In: Proceedings of the 33rd international conference on neural information processing systems. Curran Associates Inc., Red Hook, NY, USA (2019).
8. Han, C.-D. et al.: Adaptable hamiltonian neural networks. Physical Review Research. 3, 2, (2021). https://doi.org/10.1103/physrevresearch.3.023156.
9. He, J., Xu, J.: MgNet: A unified framework of multigrid and convolutional neural network. Science China Mathematics. 62, 7, 1331–1354 (2019). https://doi.org/10.1007/s11425-019-9547-2.
10. Huang, Y.-W.: Neural networks for chemical engineers edited by a. B. Bulsari (Lappeenranta university of technology, Finland). Journal of the American Chemical Society. 118, 37, 8987–8987 (1996). https://doi.org/10.1021/ja955254c.
11. Ivanov, A. et al.: Polynomial neural networks and Taylor maps for dynamical systems simulation and learning. Frontiers in Artificial Intelligence and Applications. 1230–1237 (2019). https://doi.org/10.3233/FAIA200223.
12. Karniadakis, G.E. et al.: Physics-informed machine learning. Nature Reviews Physics. 3, 6, 422–440 (2021). https://doi.org/10.1038/s42254-021-00314-5.
13. Lusch, B. et al.: Deep learning for universal linear embeddings of nonlinear dynamics. Nature Communications. 9, 1, (2018). https://doi.org/10.1038/s41467-018-07210-0.
14. Lutter, M. et al.: Deep lagrangian networks: Using physics as model prior for deep learning. In: International conference on learning representations. (2019).
15. Roehrl, M.A. et al.: Modeling system dynamics with physics-informed neural networks based on lagrangian mechanics. IFAC-PapersOnLine. 53, 2, 9195–9200 (2020). https://doi.org/10.1016/j.ifacol.2020.12.2182.
16. Rubanova, Y. et al.: Latent ODEs for irregularly-sampled time series. In: Proceedings of the 33rd international conference on neural information processing systems. Curran Associates Inc., Red Hook, NY, USA (2019).
17. Vaidyanathan, S.: Adaptive control and synchronization of a generalized Lotka–Volterra system. International Journal on Bioinformatics & Biosciences. 1, 1, 12 (2011).
18. Wang, Y.-J., Lin, C.-T.: Runge-Kutta neural network for identification of dynamical systems in high accuracy. IEEE Transactions on Neural Networks. 9, 2, 294–307 (1998). https://doi.org/10.1109/72.661124.
19. Weinan, E.: A proposal on machine learning via dynamical systems. Communications in Mathematics and Statistics. 5, 1, 1–11 (2017). https://doi.org/10.1007/s40304-017-0103-z.