Polynomial Neural Networks and Taylor Maps for Dynamical Systems Simulation and Learning

Andrei Ivanov[1] and **Anna Golovkina**[2] and **Uwe Iben**[3]

Abstract. The paper discusses the connection of Taylor maps and polynomial neural networks (PNN) for numerical solving of the ordinary differential equations (ODEs). Having the system of ODEs, it is possible to calculate weights of PNN that simulates the dynamics of these equations. It is shown that proposed PNN architecture can provide better accuracy with less computational time in comparison with traditional numerical solvers. Moreover, neural network derived from the ODEs can be used for simulation of system dynamics with different initial conditions, but without training procedure. Besides, if the equations are unknown, the weights of the PNN can be fitted in a data-driven way. In the paper, we describe the connection of PNN with differential equations theoretically along with the examples for both dynamics simulation and learning with data.

1 Introduction and Related Works

Traditional methods for solving systems of differential equations imply a numerical step-by-step integration of the system. For some problems such as stiff ordinary differential equations (ODEs), this integration procedure leads to time-consuming algorithms because of the limitations on the time step that is used to achieve the necessary accuracy of the solution. From this perspective, neural network (NN) as a universal function approximator can be applied for the construction of the solution in a more efficient way. This property induces wide research and consequently large number of papers devoted to the neural networks design for ODEs and partial differential equations (PDEs) solutions. We examine some of the publications in more details and include a short review and comparison of existing approaches as well as highlight the advantages of the proposed methodology.

Paper [16] proposes a method to solve initial and boundary value problems using feed-forward neural networks. The solution of the differential equation is written as a sum of two parts. The first part satisfies the initial/boundary conditions. The second part corresponds to a neural network output. The same technique is applied for solving the Stokes problem in [7, 9, 28].

Paper [29] considers a neural network training to satisfy the differential operator, initial condition, and boundary conditions for the PDE. The authors of [31] convert a PDE to a stochastic control problem and use deep reinforcement learning for an approximation of solution derivative with respect to the space coordinate.

Other approaches rely on the implementation of a traditional step-by-step integrating method in a neural network framework, cf. [2, 30]. Paper [30] proposes such an architecture. After fitting, the NN produces an optimal finite difference scheme for a specific system. The back-propagation technique through an ODE solver is proposed in [8]. The authors construct a certain type of NN that is analogous to a discretized differential equation. This group of methods requires a traditional numerical method to simulate dynamics.

Polynomial neural networks (PNN) are also widely highlighted in the literature, cf. [33, 32, 25]. For example, paper [33] proposes a polynomial architecture that approximates differential equations. The Legendre polynomial is chosen as a basis in [32]. But it should be noted, that in all these papers, the polynomial architectures are used as "black box" models and the authors do not indicate its connection to the ODEs. Thus, the advantages of NN can be exploited only partially.

In all the described approaches, NNs are trained to consider the initial conditions of the differential equations. This means that a NN has to be trained each time when the initial conditions are changed. The above-described techniques are applicable to the general form of differential equations but are able to provide only a particular solution of the system that is a strict limitation of applicability.

Some studies demonstrate the application of neural networks for physical systems learning [21, 15]. But the described methods require large volumes of either measured or simulated data for training of the NN.

In this paper, we consider nonlinear systems of ODEs with polynomial right-hand side:

$$\frac{d}{dt}\mathbf{X} = \mathbf{F}(t,\mathbf{X}) = \sum_{k=0}^{\infty} P^{1k}(t)\mathbf{X}^{[k]}, \qquad (1)$$

where t is an independent variable, $\mathbf{X} \in R^n$ is a state vector, and $\mathbf{X}^{[k]}$ means k-th Kronecker power of vector $\mathbf{X}$. For example, for $\mathbf{X} = (x_1, x_2)$ we have $\mathbf{X}^{[2]} = (x_1^2, x_1x_2, x_2^2)$, $\mathbf{X}^{[3]} = (x_1^3, x_1^2x_2, x_1x_2^2, x_2^3)$ after reduction of the same terms.

Such nonlinear systems arise in different fields such as automated control, robotics, mechanical and biological systems, chemical reactions, drug development, molecular dynamics, and so on. Moreover, it is often possible to transform a nonlinear equation (either ODE or PDE) to a polynomial form.

Based on the Taylor mapping technique, it is possible to build a PNN that approximates a general solution of (1). The weights of the PNN are calculated at once directly from the system of ODEs. By a Taylor map we mean transformation $\mathcal{M} : \mathbf{X}_0 = \mathbf{X}(t_0) \to \mathbf{X}(t_1)$ in form of

$$\mathbf{X}(t_1) = W_0 + W_1\,\mathbf{X}_0 + W_2\,\mathbf{X}_0^{[2]} + \ldots + W_k\,\mathbf{X}_0^{[k]}, \qquad (2)$$

where $\mathbf{X} \in R^n$, and matrices W_i are weights. Transformation (2) is linear in weights W_i and nonlinear with respect to $\mathbf{X}_0$.

[1] DESY, Germany, email: 05x.andrey@gmail.com, andrei.ivanov@desy.de
[2] Saint Petersburg State University, Russia, email: a.golovkina@spbu.ru
[3] OOO Robert Bosch, Russia, email: uwe.iben@de.bosch.com

In the literature, this map $\mathcal{M}$ can be referred to Taylor maps and models [19], tensor decomposition [10], matrix Lie transform [3], exponential machines [22], and others. In fact, the transformation (2) is just a polynomial regression with respect to the components of $\mathbf{X}$.

Though many numerical solvers for (1) can be considered as maps, they are commonly based on small time steps Δt and weight matrices $W_i = W_i(\Delta t, \mathbf{X}_0)$ that depend on $\mathbf{X}_0$. In this paper, by mapping approach we mean transformation (2) with the weight matrices $W_i = W_i(t_1 - t_0)$ that are estimated for a larger time interval $t_1 - t_0 > \Delta t$ and do not depend on $\mathbf{X}_0$. The greatest advantage of the mapping approach is the computational performance. Instead of step-by-step integration with numerical solvers, one can apply a map (2) that estimates dynamics in a large time interval for different initial conditions $\mathbf{X}_0$ at the same accuracy.

In the paper, we briefly describe an algorithm for calculating of weight matrices W_i in (2) for an arbitrary time step $t_1 - t_0$. Considering map (2) as a neuron (Fig. 1), it is possible to design a PNN that simulates the dynamics of the given differential equation. While the PNN is connected to differential equations, it is also possible to use this architecture for data-driven identification of physical systems.

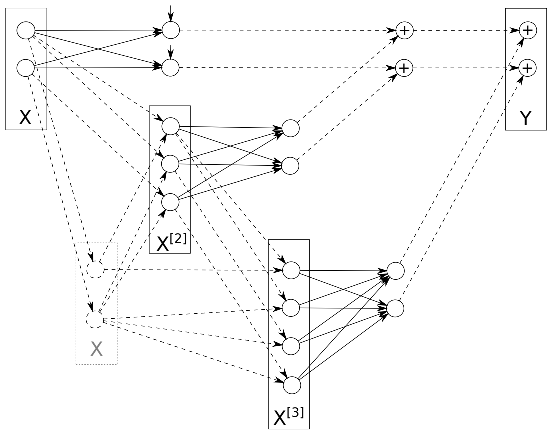

Figure 1. Polynomial neuron of third order nonlinearity

The rest of the paper is organized as follows. Sec. 2 describes the general approach for building Taylor maps for the systems of ODEs. Sec. 3 corresponds to the PNNs weights calculating for dynamics simulation of applied physical systems. Sec. 4 is devoted to training PNN. The regularization method along with the examples of data-driven dynamics learning without equation consideration are described.

2 Taylor maps for the system of ODEs

The solution of (1) with the initial condition $\mathbf{X}(t_0) = \mathbf{X}_0$ during the time interval $t - t_0$ in its convergence region can be presented in the power series [4, 3],

$$\mathbf{X}(t) = \mathcal{M}(t - t_0) \circ \mathbf{X}_0 = \sum_{k=0}^{\infty} M^{1k}(t)\mathbf{X}_0^{[k]}. \tag{3}$$

Theoretical estimations of accuracy and convergence of the truncated series in solving of ODEs can be found in [6]. In [5], it is shown how to calculate matrices M^{1k} by introducing new matrices P^{ij}. The main idea is replacing (1) by the equation

$$\frac{d}{dt}M^{ik}(t) = \sum_{j=i}^{k} P^{ij}(t)M^{jk}(t), \ 1 \le i < k. \tag{4}$$

The last equation does not depend on $\mathbf{X}_0$ and should be solved at once with initial condition $M^{kk}(t_0) = I^{[k]}, \ M^{jk}(t_0) = 0, j \neq k$, where I is the identity matrix. The truncated solution of (4) for desirable time interval $t_1 - t_0$ yields Taylor map (2) with $W_i = M^{1i}$.

One of the advantages of the described approach is simulation performance. Indeed, instead of step-by-step integration of the equation (1) with a small time step Δt every time when the initial condition $\mathbf{X}_0$ is changed, one should integrate map $\mathcal{M}(t)$ at once for the unique initial condition and the whole desirable time interval $t_1 - t_0$. Then the same map can be applied for different initial conditions.

3 Simulation of dynamical systems

Since the Taylor mapping approach is commonly used in accelerator physics [13, 26], we demonstrate the proposed method with the simplified example of charged particle motion. We also introduce deep PNN for simulation and control of charged particle beam dynamics, and discuss a shallow PNN architecture for simulation of a stiff ODE.

3.1 Charged particle dynamics

The particle dynamics in the electromagnetic fields can be described by a system of ODEs that has a complex nonlinear form. For simplicity, let us consider an approximation [27] of a particle motion in cylindrical deflector written in form of (1):

$$\begin{aligned} x' &= y, \\ y' &= -2x + x^2/R, \end{aligned} \tag{5}$$

where R is the equilibrium radius of particle bending, x is a deviation from this radius, and $x' = y$ is a derivative on the bending angle.

For example, let us consider a deflector with $R = 10\,\mathrm{m}$ that rotates a reference particle with initial conditions $x = 0$, $x' = 0$ on angle $\pi/4$. For simulation, we investigate dynamics of particle with nontrivial initial conditions that lead to particle oscillation.

Traditional approach to solve system (5) is step-by-step integration with numerical solvers. For this purpose, we use Runge–Kutta method of fourth order with fixed time step. To control the numerical error in this example, we use integrating angle 30 times smaller than bending angle. Larger steps introduce nonphysical dissipation in particle motion. In step-by-step integration, to track particle inside the deflector, one has to do 30 steps. In contrast to this, mapping technique allows to calculate output state in single step (see Fig. 2).

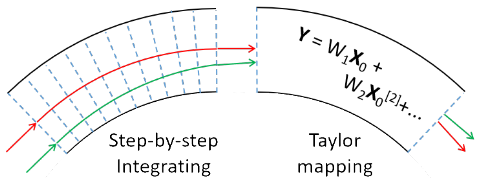

Figure 2. Step-by-step numerical integrating and Taylor mapping

Let us find a third order Taylor map (2) that transforms initial particle state $\mathbf{X}_0 = (x_0; y_0)$ at the entrance of the deflector to the resulting state $\mathbf{X}_1 = (x_1; y_1)$ at the end of it:

$$\begin{pmatrix} x \\ y \end{pmatrix} = W_1 \begin{pmatrix} x_0 \\ y_0 \end{pmatrix} + W_2 \begin{pmatrix} x_0^2 \\ x_0 y_0 \\ y_0^2 \end{pmatrix} + W_3 \begin{pmatrix} x_0^3 \\ x_0^2 y_0 \\ x_0 y_0^2 \\ y_0^3 \end{pmatrix}. \tag{6}$$

Combining this map with (5), one can write

$$\begin{gathered} \begin{pmatrix} x \\ y \end{pmatrix}' = W_1' \begin{pmatrix} x_0 \\ y_0 \end{pmatrix} + W_2' \begin{pmatrix} x_0^2 \\ x_0 y_0 \\ y_0^2 \end{pmatrix} + W_3' \begin{pmatrix} x_0^3 \\ x_0^2 y_0 \\ x_0 y_0^2 \\ y_0^3 \end{pmatrix}, \\ \begin{pmatrix} x \\ y \end{pmatrix}' = \begin{pmatrix} 0 & 1 \\ -2 & 0 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} 0 & 0 & 0 \\ 1/R & 0 & 0 \end{pmatrix} \begin{pmatrix} x^2 \\ xy \\ y^2 \end{pmatrix}. \end{gathered} \tag{7}$$

Grouping the terms with x_0 and y_0 in the same powers for the last system, one can obtain a system of ODEs that does not depend on $\mathbf{X}_0 = (x_0; y_0)$ and represents dynamics of weight matrices

$$\begin{aligned} W_1' &= f_1(W_1, W_2, W_3), \quad W_1(0) = I, \\ W_2' &= f_2(W_1, W_2, W_3), \quad W_2(0) = 0, \\ W_3' &= f_3(W_1, W_2, W_3), \quad W_3(0) = 0, \end{aligned} \tag{8}$$

where f_i is function arising after same power terms grouping. By integrating this system during the interval $[0; \pi/4]$, we can receive the desired map. For example, the map up to two digits is

$$\begin{gathered} \begin{pmatrix} x_1 \\ y_1 \end{pmatrix} = \begin{pmatrix} 0.44 & 0.63 \\ -0.13 \cdot 10 & 0.44 \end{pmatrix} \begin{pmatrix} x_0 \\ y_0 \end{pmatrix} + \\ \begin{pmatrix} 0.23 \cdot 10^{-1} & 0.12 \cdot 10^{-1} & 0.26 \cdot 10^{-2} \\ 0.40 \cdot 10^{-1} & 0.35 \cdot 10^{-1} & 0.12 \cdot 10^{-1} \end{pmatrix} \begin{pmatrix} x_0^2 \\ x_0 y_0 \\ y_0^2 \end{pmatrix} + \\ \begin{pmatrix} 0.21 \cdot 10^{-3} & 0.17 \cdot 10^{-3} & 0.47 \cdot 10^{-4} & 0.56 \cdot 10^{-5} \\ 0.83 \cdot 10^{-3} & 0.95 \cdot 10^{-3} & 0.32 \cdot 10^{-3} & 0.47 \cdot 10^{-4} \end{pmatrix} \begin{pmatrix} x_0^3 \\ x_0^2 y_0 \\ x_0 y_0^2 \\ y_0^3 \end{pmatrix}. \end{gathered}$$

Using this polynomial transformation, one can calculate state of the particle at the end of the deflector for arbitrary initial conditions $\mathbf{X}_0 = (x_0; y_0)$. The simulation in this case is 160 times faster than the Runge–Kutta based simulation.

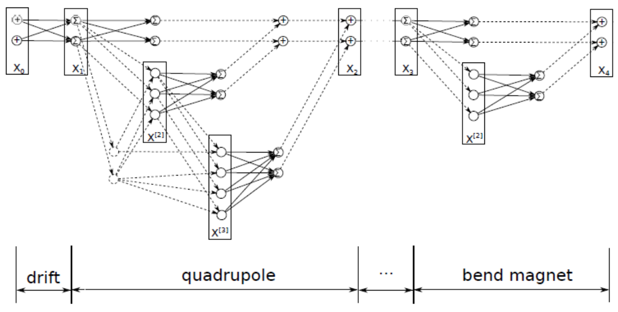

Figure 3. Deep PNN for simulation of a charged particle accelerator

For long-term charged particle dynamics investigation, the traditional step-by-step numerical methods are not suitable because of their performance limitations. Instead of solving differential equations directly, one can estimate Taylor maps of high orders nonlinearity for each control element in the accelerator. By combining such maps consequently, one can obtain a deep PNN that represents the whole accelerator lattice (see Fig. 3). In such PNN, each layer is represented by a Taylor map and corresponds to the physical element in the charged particle accelerator.

The described approach is used for simulation of nonlinear spin-orbit dynamics in electric dipole moment (EDM) search project [26, 13]. The particle dynamics was described by a nine-dimensional state vector. The deep PNN architecture has more than 100 polynomial layers that represent lattice of the accelerator. The computational performance is increased in 1500 times in comparison with traditional step-by-step integration.

Along with the computational performance, another advantage of the proposed model is the possibility to take into account uncertainties. It is easy to represent misalignments and field errors in physical accelerator by introducing additional polynomial layers inside the PNN. This feature may be essential for control of accelerators, while the PNN architecture provides a possibility to fit layers with measured data.

3.2 The Rayleigh-Plesset equation

The Rayleigh-Plesset equation governs the dynamics of a spherical gas bubble in an infinite body of in-compressible liquid. It is derived from the conservation of mass and momentum, which results in a highly nonlinear second order ODE for the bubble radius R. The pressure inside the bubble is denoted by p_B, and the pressure outside and far away from the bubble is denoted by p_∞. The density ρ of the surrounding liquid is assumed to be constant. The equation of motion has the form

$$R\frac{d^2R}{dt^2} + \frac{3}{2}R\left(\frac{dR}{dt}\right)^2 = \frac{1}{\rho}(p_B - p). \tag{9}$$

Viscous terms as well as surface tension are neglected. Provided that $p_B(t)$ is known and $p_\infty(t)$ is given, the Rayleigh-Plesset equation can be used to find the time-varying gas bubble radius $R(t)$ which includes collapses $(R \to 0)$ and rebounds.

Because we study the numerical solvers for the Rayleigh-Plesset equation, we assume that the pressure difference between p_B and p_∞ is constant, say $p_B = 2300$ Pa and $p_\infty = 1 \times 10^5$ Pa for simplicity. The solution contains very strong gradients during the collapse phase and the following rebound phase, i.e. it is a stiff ODE. In order to guarantee the accuracy of the numerical solution, very small time steps and a high order of the solver must be used. The Rayleigh-Plesset equation (9) is reformulated in a system of first-order ODEs with a polynomial right-hand side

$$\begin{aligned} \frac{dy_1}{dt} &= y_2, \\ \frac{dy_2}{dt} &= -\frac{(p_B - p)}{\rho} y_3 - \frac{3}{2} y_2^2, \\ \frac{dy_3}{dt} &= -y_3^2 y_2, \end{aligned} \tag{10}$$

where $y_1 = R$, $y_2 = \dot{R}$, $y_3 = 1/R$, and the initial conditions are $y_1(0) = R(t = 0) = R_0$, $y_2(0) = 0$, and $y_3(0) = 1/y_1(0)$.

Since the equation is a stiff ODE, we use an adaptive mapping approach. This means that we build Taylor maps of seventh order for different time intervals ranging from $\Delta t =$

$1 \times 10^{-4}\,\text{sec}$ to $\Delta t = 1 \times 10^{-19}\,\text{sec}$. Then we apply maps based on the relative error during the simulation. For this approach, the polynomial neurons can be organized in a shallow architecture that offers the possibility of simulating of a bubble motion with control of accuracy. To compare the computational performance, we run simulations for three initial conditions $R_0 \in \{0.85 \times 10^{-3}\,\text{m}, 1 \times 10^{-3}\,\text{m}, 1.15 \times 10^{-3}\,\text{m}\}, \dot{R}_0 = 0$ up to the collapsing time. The adaptive mapping approach is 2.5 times faster than the ode45 solver for stiff differential equations from [20] with similar accuracy (see Fig. 4, 5).

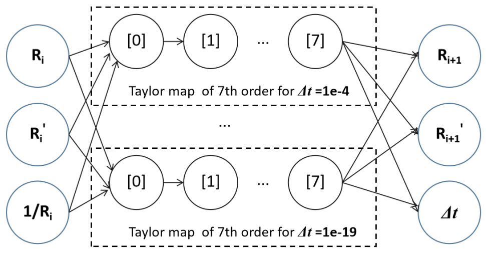
R_i
R_i'
1/R_i
[0]
[1]
...
[7]
Taylor map of 7th order for Δt =1e-4
...
Taylor map of 7th order for Δt =1e-19
R_i+1
R_i+1'
Δt

Figure 4. Adaptive mapping technique in a shallow PNN architecture

3.3 The Burgers' equation

The Burgers' equation is a fundamental PDE that is used in various areas, such as fluid mechanics, nonlinear acoustics, gas dynamics, and traffic flow. This equation is also often used as a benchmark for numerical methods. In [12], a feed-forward NN is trained to satisfy Burgers' equation and certain initial conditions, but the computational performance of the approach is not estimated. In this section, we demonstrate how to build a PNN that solves the Burgers' equation and does not require training to satisfy different initial conditions.

The Burgers' equation has a form

$$\frac{\partial u(t,x)}{\partial t} + u(t,x)\frac{\partial u(t,x)}{\partial x} = \nu \frac{\partial^2 u(t,x)}{\partial x^2}. \tag{11}$$

Following [1] for benchmarking, we use an analytic solution

$$u_1(t,x) = -2\frac{\nu}{\phi(t,x)}\frac{d\phi}{dx} + 4,$$
$$\phi(t,x) = exp\frac{-(x-4t)^2}{4\nu(t+1)} + exp\frac{-(x-4t-2\pi)^2}{4\nu(t+1)},$$

and a traditional numerical method

$$\frac{u_i^{n+1} - u_i^n}{\Delta t} + u_i^n \frac{u_i^n - u_{i-1}^n}{\Delta x} = \nu \frac{u_{i+1}^n - 2u_i^n + u_{i-1}^n}{\Delta x^2}, \tag{12}$$

where n stands for the time step, and i stands for the grid node.

The equation (12) presents a finite difference method (FDM) that consists of an Euler explicit time discretization scheme for the temporal derivatives, an upwind first order scheme for the nonlinear term, and finally a centered second order scheme for the diffusion term. The time step for benchmarking is fixed to $\Delta t = 2.5 \times 10^{-4}\,\text{sec}$ with the uniform spacing of $\Delta x = 2\pi/1000$, $\nu = 0.05\,\text{m}^2/\text{sec}$. Thus, for the numerical solution for times from $t = 0\,\text{sec}$ to $t = 0.5\,\text{sec}$ on $x \in [0, 2\pi]$, the method requires the mesh with 1000 steps on space coordinate x and 2000 time steps.

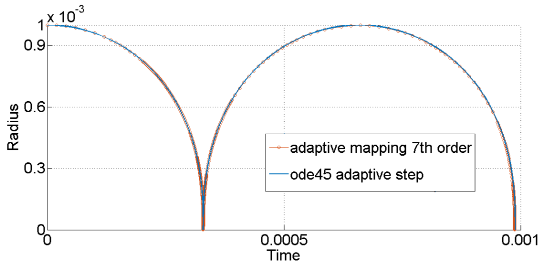

Figure 5. Simulation of the Rayleigh-Plesset equation: blue line for ode45, orange circles for PNN

It is indicated in [1] that the FDM introduces a dispersion error in the solution (see Fig. 6). Such error can be reduced by increasing the mesh resolution, but then the time step should be decreased to respect the stability constraints of the numerical scheme.

To build a PNN for solving the Burgers' equation we translate (11) to a system of ODEs and build a Taylor map in accordance with Sec. 2. Assuming that the right-hand side of (11) can be approximated by a function $f(x, u(t,x))$ and considering this approximated equation as a hyperbolic one, it is possible to derive the system of ODEs

$$\frac{d}{dt}\begin{pmatrix}\mathbf{X}\\ \mathbf{U}\end{pmatrix} = \begin{pmatrix}\mathbf{U}\\ f(x,\mathbf{U})\end{pmatrix}, \tag{13}$$

where $\mathbf{U} = (u_1, \ldots, u_{1000})$, $u_i(t) = u(t, x_i)$, and $\mathbf{X} = (x_1, \ldots, x_{1000})$ is vector of discrete stamps on space. This transformation from PDE to ODE is well known and can be derived using the method of characteristics and direct method [11]. If $f(x, u(t,x))$ is the same discretization as in (12), then the equation (13) leads to the system of 2000 ODEs

$$x_i' = u_i, \quad u_i' = f(\nu, u_{i+1}, u_i, u_{i-1}, x_{i+1}, x_i, x_{i-1}),$$

which can be easily expanded to the power series with respect to the $\mathbf{X}$ and $\mathbf{U}$ up to the necessary order of nonlinearity.

Since there are only first and second order approximations in the benchmarking FDM scheme, we build a Taylor map of only the first order in a time interval $\Delta t = 1.25 \times 10^{-3}\,\text{sec}$. This time step is five times larger than that is used in the benchmarks. The numerical solution provided by the resulting PNN is presented in Fig. 6, and the accuracy and performance are compared in Tab. 1.

Table 1. Comparison of the simulation of the Burgers' equation by FDM and PNN

Method	**Time step**	**Elapsed time**	**MSE for $\mathbf{u_1(0.5, x)}$**
FDM	$2.50 \times 10^{-4}\,\text{sec}$	$0.055\,\text{sec}$	$8.0 \cdot 10^{-2}$
PNN	$1.25 \times 10^{-3}\,\text{sec}$	$0.016\,\text{sec}$	$5.5 \cdot 10^{-3}$

The PNN numerically estimates dynamics in a larger time interval and provides better accuracy with less computational time in comparison with FDM of the same order of derivative approximations. If the FDM scheme is adjusted to a higher accuracy, the computational time will be increased even more. Accuracy is calculated as a mean square error (MSE) metric between the numerical solution and its analytic form at final time $t = 0.5\,\text{sec}$.

Since the derived from the differential equation PNN is an approximation of the general solution, it can be used for simulation of the

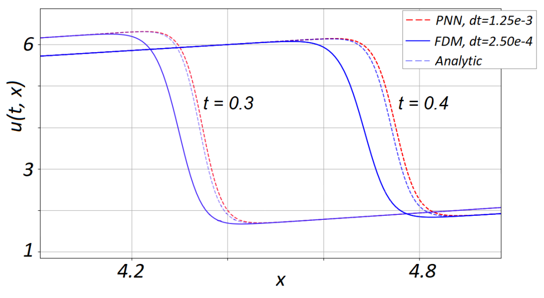

Figure 6. Burgers' equation: blue line for FDM, red dashed line for PNN, blue dashed line for analytic solution

dynamics with different initial conditions without weight modification. For example, for another analytic solution

$$u_2(t,x) = (1 + exp(0.5(x-0.5t)/\nu)^{-1},$$

the same PNN provides a numerical solution in time $t = 0.5$ sec with MSE error 7.2×10^{-8} sec, while FDM yields 1.2×10^{-7} sec. The elapsed times are the same since the mesh sizes are not changed.

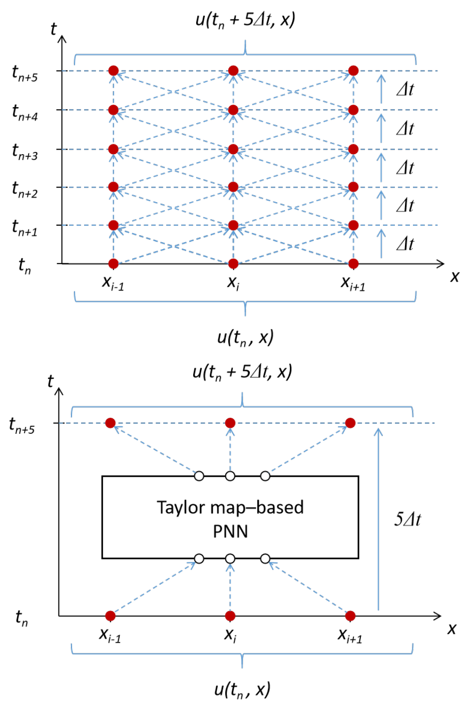

Figure 7. Numerical schemes for FDM, $\Delta t = 2.5 - 4sec$ (top Fig.) and PNN, $\Delta t = 1.25 \times 10^{-3}$ sec for (11)

4 Learning dynamical systems from data

In this section, we discuss the training of PNN. We demonstrate how one can adjust weights of the PNN for certain initial conditions of the Burgers' equation by additional training of the PNN derived from the equation. We also introduce a regularization of the PNN based on combinatorial binary optimization and demonstrate the proposed methods in a practical application of a data-driven system learning.

4.1 Training the PNN for certain initial conditions

The PNN derived from differential equation works for any initial condition with the same level of accuracy. The accuracy depends on the size of the PNN, namely the order of nonlinearities in the map (2). But it is still possible to train PNN for certain initial conditions.

In order to train the PNN for the Burgers' equations, we consider a deep architecture with 400 layers with shared weights. Each layer M is defined by the Taylor map (2) for time $dt = 1.25 \times 10^{-3}$ sec:

$$u(t=0,x) \to M \to \ldots \to M \to u(t=0.5,x).$$

The loss function is defined as the inconsistency of the numerical solution with differential equations. The initial values of weights are calculated from the PDE with the corresponding Taylor map.

Initially, the PNN has $4 \cdot 10^6$ weights with only $5 \cdot 10^3$ of them not equal to zero. The point of training is to adjust these weights to achieve better accuracy for the given initial conditions. We implemented a simplified training based on a coordinate descent method that yields a three-fold increase in accuracy.

4.2 Regularization of the PNN

It is common for physical systems that a Taylor map is presented by sparse matrices W_i. So it is important to identify non-zero weights in the polynomial neural network. Classical L_1 and L_2 regularization terms do not work in this case since they tend to decrease weight. The weights in PNN can be large by absolute value but the total amount of non-zero weights should be as small as possible. To solve this issue, one can introduce a binary mask B_i that is applied to the weight matrices:

$$\mathbf{X}_{i+1} = (B_0 * W_0) + (B_1 * W_1)\mathbf{X}_i + (B_2 * W_2)\mathbf{X}_i^{[2]} + \ldots, \quad (14)$$

where $(B_i * W_i)$ means element-wise multiplication.

During the fitting of weight matrices W_i, it is also necessary to find an optimal mask B_i. If the weights W_i are fixed after each training epoch, the map (14) becomes linear for the components of B_i.

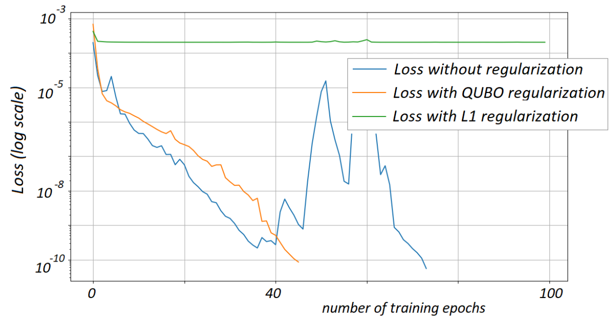

Figure 8. Loss functions and number of epochs for training PNN for the Van der Pol oscillator: without regularization (blue line), with L1 regularization (green line), and with the proposed QUBO–based regularization.

It is possible to formulate regularization as a high-order binary optimization problem and translate it into a quadratic unconstrained binary optimization (QUBO). For example, the loss function $||X_{i+1} - (\mathcal{B} * \mathcal{M}) \circ \mathbf{X}_i||$ translates directly to QUBO, while loss function that is introduced for Burgers' equation translates to the high-order unconstrained binary optimization problem. By introducing additional binary variables it is possible finally to formulate regularization as QUBO.

To demonstrate the advantage of QUBO–based regularization, we implement a simplified example with the Van der Pole oscillator. The equation is widely used in the physical sciences and engineering and can be used for the description of the pneumatic hammer, steam engine, periodic occurrence of epidemics, economic crises, depressions, and heartbeat. The equation has well-studied dynamics and is widely used for testing of numerical methods, e.g., [24].

The Van der Pol oscillator is defined as the system of ODEs $x'' = x' - x - x^2x'$ that can be presented in the form of

$$\begin{aligned} x' &= y, \\ y' &= y - x - x^2y. \end{aligned} \tag{15}$$

We generate a training data as a particular solution $\{\mathbf{X}_i\}_{i=1;n}$ of the system (15) with the initial condition $\mathbf{X}_0 = (-2, 4)$. After this solution was generated, the equation is not used further.

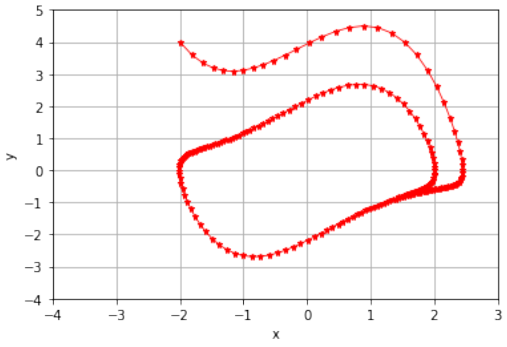

Figure 9. Training data for the NN as a numerically generated particular solution

Having this training data set (Fig. 9), the proposed PNN can be fitted with the mean squared error (MSE) as a loss function based on the norm

$$||\mathbf{X}_{i+1} - \mathcal{M} \circ \mathbf{X}_i|| = || \begin{pmatrix} x_{i+1} \\ y_{i+1} \end{pmatrix} - W_0 - \ldots - W_3 \begin{pmatrix} x_i^3 \\ x_i^2 y_i \\ x_i y_i^2 \\ y_i^3 \end{pmatrix} ||.$$

We implement the described technique in Keras/TensorFlow and fit a third order PNN with an Adamax optimizer. After each training epoch we applied QUBO–based regularization for the weights of the PNN.

Fig. 8 shows that QUBO–based regularization can significantly decrease the number of epochs required to achieve an appropriate level of accuracy. However, the time required to solve the QUBO problem is longer then training without regularization. It should be noted that using special hardware (e.g., quantum or digital annealers) instead of classical QUBO-solvers can potentially provide additional improvement in computation time.

The generalization property of the network can be investigated by examining prediction not only from the training data set but also for new initial conditions. Fig. 10 demonstrates predictions that are calculated starting also at the new points $(1, 2)$, $(2, -2)$, and $(-3, -3)$ that are not presented to the PNN during fitting. For the prediction starting from the training initial condition, the mean relative error of

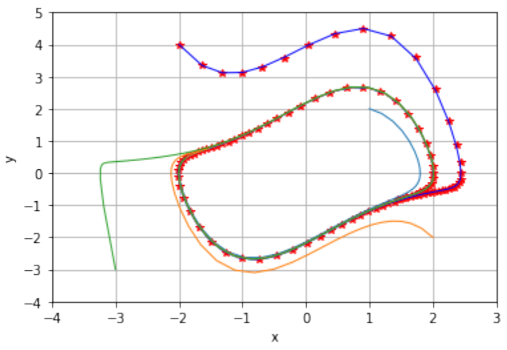

Figure 10. Prediction for new initial condition that are not presented during training

the predictions is $4.8 \cdot 10^{-5}$. For the new initial conditions, the mean error is even less than or equal to $9.8 \cdot 10^{-6}$.

4.3 Data-driven identification of a zinc-air energy storage system

Zinc-air batteries are one of the most promising energy storage systems. An effective mathematical model is a key part of a battery management system, responsible for its operational control and internal state evaluation. The model-based analysis is an effective tool in the design and manufacture of power supplies, as well as in the analysis of their behavior under different operating conditions [17].

There are several approaches for building a dynamic model of a battery. At present, the studies are mostly focused on mathematical models of zinc-air batteries derived from the electrochemical principles of the batteries' functioning. This implies utilization of complicated nonlinear PDEs that describe the battery behavior precisely but require time-consuming numerical procedures for their solution.

Another type is a "black box", or data-driven model, that is formed by analyzing experimental data using artificial neural networks. Since this approach does not imply the presence of state equations in the model, it works only on data that is similar to what the NN is trained on. This requires the availability of operational data from real environments of sufficient volume and quality, which in practice cannot always be guaranteed.

Thus, state space models with lumped parameters are more suitable for analysis, control, and optimization of power supplies since in this case, they include battery general characteristics such as voltage, current, state of charge, and so on. In the most general form without specification of any details, it can be presented as follows:

$$\begin{aligned} x(k+1) &= f(x(k), u(k)), \\ y(k) &= g(x(k), u(k)). \end{aligned} \tag{16}$$

It should be noted that real power sources exhibit strongly nonlinear effects that are to be included as a nonlinear part of the $f(\cdot)$ and $g(\cdot)$ functions of the mathematical model (16) from [23]. This implies manual derivation of the equations for dynamics description and identification of its parameters. In this work we take an alternative approach with a data-driven construction of a mathematical model from a set of controlled experiments [18]. We investigate whether it is possible to extract features from current and voltage measurements collected during battery discharge in various fixed loading conditions.

For demonstration of this approach, we consider only the voltage dynamics for discharge currents 600 mA, 700 mA, and 800 mA and

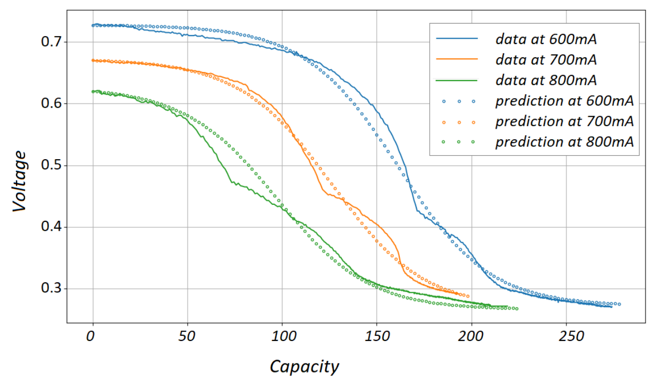

Figure 11. The input data (lines) and predictions (dots) provided by PNN

solve an identification problem only for regions where voltage decreases over time.

Initial data taken from the open source [18] is filtered with a simple moving average window and aligned to a constant time stamp of 1 msec. To present dynamics of the voltage decrease, we use a PNN with three layers (see Fig. 11). Each layer is represented by a Taylor map, where TM_1 and TM_3 are Taylor maps of second order, and TM_2 is a Taylor map of fifth order.

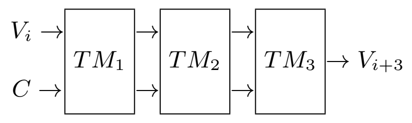

Figure 12. The PNN architecture for learning voltage dynamics

To train the PNN, the vector of voltage at the given time stamp and the value of discharge current are used as input data, and voltage after three time stamps is used as output data. Also, the regularization described above is used.

After training, the dynamics of the voltage decrease is calculated as the consequent prediction performed by the PNN. Starting with initial voltage V_0 at time $t = 0$, we calculate the dynamics using the formula:

$$V_0 \to V_1 = PNN(V_0) \to \ldots \to V_k = PNN(V_{k-1}).$$

In this way, the PNN plays the role of a model of the system that can calculate dynamics with different initial conditions V_0.

Fig. 11 shows the results of the voltage dynamics simulation with the trained PNN. While additional features such as derivatives of voltage or power can potentially increase the accuracy, the built PNN preserves physical properties of the dynamics. For example, the PNN successfully predicts the lower level of voltage that is the same for all currents in contrast to simple function approximations that can give aberrant negative values of voltage [23].

One of the advantages of polynomial architectures in comparison with other machine learning methods is its consistency with consistency of dynamical systems and differential equations. A PNN trained with data can provide a mathematical model with physical properties preservation. For example, in [14] it is shown that MLP and LSTM simply memorize data for simplified physical systems. They cannot predict dynamics for data that is not presented during fitting. The PNN architecture allows for extrapolation of dynamics of new initial conditions and preservation of physical properties of the dynamical systems.

5 Supplementary code

The program code contains the considered examples and can be found in the following GitHub repositories.

github.com/andiva/DeepLieNet: implementation of PNN in Keras/TensorFlow (sec. 1, 3.3, 3.2, and 4.3):
/demo/Deflector: cylindrical deflector simulation
/demo/accelerator.ipynb: storage ring simulation
/demo/Ray_Ples: simulation of the Rayleigh-Plesset equation
/demo/Battery: trained PNN for voltage dynamics
github.com/andiva/AQCC: PNN for simulation of the Burgers' equation (Sec. 3.3), training PNN for the Burgers' equation (Sec. 4.1), QUBO–based regularization and training of PNN for the Van der Pol oscillator (Sec. 4.2).

6 Conclusion

Since the Taylor maps and PNN have strong connections to systems of differential equations, these methods can be used for physical system simulation and data-driven identification. If the differential equations for the system are known, it is possible to calculate weights of the PNN with the necessary level of accuracy. In this way, the built PNN can be used for simulation of the dynamics instead of the traditional numerical solvers. It is shown that Taylor mapping and the PNN provide the same or even better accuracy with less computational time in comparison to step-by-step integration of the equations.

If the equations of the system are not known, the PNN can be trained from scratch with the available data. One can define an architecture of the PNN and fit weights directly with data. Instead of fitting of parametrized equations, this approach allows for fitting of weights of the PNN directly.

In the paper, we consider simulation of dynamical systems arising in practical applications with the PNN. For training the PNN, we introduce QUBO–based regularization and demonstrate data-driven dynamics learning with a simplified Van der Pol oscillator and apply these methods to the problem of battery development.

We do not consider the problem of accuracy, PNN architecture selection, or optimization methods for training in detail. This work should be done in further research. With the provided examples, we instead demonstrate the applicability of Taylor mapping methods that are commonly used in dynamical systems investigation to the theory of NN and data-driven system learning.

ACKNOWLEDGEMENTS

We would like to thank Prof. Dr. Sergei Andrianov for his help with explanation of Taylor mapping techniques for solving systems of differential equations.

REFERENCES

[1] Airbus, 'Airbus quantum computing challenge', *https://www.airbus.com/innovation/tech-challenges-and-competitions/airbus-quantum-computing-challenge.html*, (2019).

[2] A. Anastassi, 'Constructing runge-kutta methods with the use of artificial neural networks', *Tech. rep. https://arxiv.org/pdf/1106.1194.pdf*, (2013).

[3] S. Andrianov, 'A matrix representation of the lie transformation', *Proceedings of the Abstracts of the International Congress on Computer Systems and Applied Mathematics*, **14**, (1993).

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

[5] S. Andrianov, 'A role of symbolic computations in beam physics', *Computer Algebra in Sc, Comp., Lecture Notes in Computer Science*, **6244**, 19–30, (2010).

[6] S. Andrianov, 'The convergence and accuracy of the matrix formalism approximation', *Proceedings of ICAP2012, Rostock, Germany*, 93–95, (2012).

[7] M. Baymani, A. Kerayechian, and S. Effati, 'Artificial neural networks approach for solving stokes problem', *Applied Mathematics*, **1**, 288–292, (2010).

[8] R. Chen, Y. Rubanova, J. Bettencourt, and D. Duvenaud, 'Neural ordinary differential equation', *Tech. rep. https://arxiv.org/pdf/1806.07366.pdf*, (2019).

[9] M. Chiaramonte and M. Kiener, 'Solving differential equations using neural networks', *cs229.stanford.edu/proj2013/ChiaramonteKiener-SolvingDifferentialEquationsUsingNeuralNetworks.pdf*, (2013).

[10] S. Dolgov, 'A tensor decomposition algorithm for large odes with conservation laws', *Tech. report. https://arxiv.org/pdf/1403.8085.pdf*, (2014).

[11] L.C. Evans, 'Partial differential equations', *Providence, R.I.: American Mathematical Society*, (2010).

[12] M. Hayati and B. Karami, 'Feedforward neural network for solving partial differential equations', *Journal of Applied Sciences*, **7**(19), 2812–2817, (2007).

[13] A. Ivanov, S. Andrianov, and Y. Senichev, 'Simulation of spin-orbit dynamics in storage rings', *Journal of Physics: Conference Series*, **747**(1), (2016).

[14] A. Ivanov, A. Sholokhova, S. Andrianov, and R. Konoplev-Esgenburg, 'Lie transform–based neural networks for dynamics simulation and learning', *Tech. report. https://arxiv.org/abs/1802.01353, https://arxiv.org/pdf/1802.01353v1.pdf*, (2018/2019).

[15] Xiaowei Jia, Jared Willard, Anuj Karpatne, Jordan Read, Jacob Zwart, Michael Steinbach, and Vipin Kumar, 'Physics guided RNNs for modeling dynamical systems: A case study in simulating lake temperature profiles', in *Proceedings of the 2019 SIAM International Conference on Data Mining*, pp. 558–566. SIAM, (2019).

[16] I.E. Lagaris, A. Likas, and D.I. Fotiadis, 'Artificial neural networks for solving ordinary and partial differential equations', *Tech. rep. https://arxiv.org/pdf/physics/ 9705023.pdf*, (1997).

[17] W. Lao-Atiman, K. Bumroongsil, A. Arpornwichanop, P. Bumroongsakulsawat, S. Olaru, and S. Kheawhom, 'Model-based analysis of an integrated zinc-air flow battery/zinc electrolyzer system', *Frontiers in Energy Research*, **7**(FEB), (2019).

[18] W. Lao-Atiman, S. Olaru, A. Arpornwichanop, and S. Kheawhom, 'Discharge performance and dynamic behavior of refuellable zinc-air battery', *Scientific data*, **6**(1), 168, (2019).

[19] Berz M., 'From taylor series to taylor models', *https://bt.pa.msu.edu/pub/papers/taylorsb/taylorsb.pdf*, (1997).

[20] MATLAB, 'version 7.12.0 (r2012a)', (2018).

[21] N. Mohajerin and S. L. Waslander, 'Multistep prediction of dynamic systems with recurrent neural networks', *IEEE Transactions on Neural Networks and Learning Systems*, **30**(11), 3370–3383, (2019).

[22] A. Novikov, M. Trofimov, and I. Oseledets, 'Exponential machines', *Tech. report. https://arxiv.org/abs/1605.03795*, (2017).

[23] S. Olaru, A. Golovkina, W. Lao-atiman, and S. Kheawhom, 'A mathematical model for dynamic operation of zinc-air battery cells', in *Proc. of 7th IFAC Symposium on Systems Structure and Control (SSSC)*, Sinaia, Romania, (September 2019).

[24] S. Pan and K. Duraisamy, 'Long-time predictive modeling of nonlinear dynamical systems using neural networks', *Hindawi Complexity*, **4801012**, (2018).

[25] V. Schetinin, 'Polynomial neural networkslearnt toclassify eeg signals', *Tech. rep., https://arxiv.org/ftp/cs/papers/0504/0504058.pdf*, (1997).

[26] Y. Senichev, A. Ivanov, A. Lehrach, R. Maier, D. Zyuzin, and S. Andrianov, 'Spin tune parametric resonance investigation', *Proceedings of the Particle Accelerator Conference*, 3020–3022, (2014).

[27] Y. Senichev and S. Møller, 'Beam dynamics in electrostatic rings', (01 2000).

[28] A. Sharma, 'Neuralnetdiffeq.jl: A neural network solver for odes', *https://julialang.org/blog/2017/10/gsoc-NeuralNetDiffEq*, (2017).

[29] J. Sirignano and K. Spiliopoulos, 'Dgm: A deep learning algorithm for solving partial differential equations', *Journal of Computational Physics*, (2018).

[30] Y. Wang and C. Lin, 'Runge-kutta neural network for identification of dynamical systems in high accuracy', *IEEE Transactions on Neural Networks*, **9**(2), 294–307, (1998).

[31] E. Weinan, J. Han, and A. Jentzen, 'Deep learning-based numerical methods for high-dimensional parabolic partial differential equations and backward stochastic differential equations', *Tech. rep. https://arxiv.org/pdf/1706.04702.pdf*, (2017).

[32] Y. Yang, M. Hou, and J. Luo, 'A novel improved extreme learning machine algorithm in solving ordinary differential equations by legendre neural network methods', *Advances in Difference Equations*, **4**(1), 469, (2018).

[33] L. Zjavka, 'Differential polynomial neural network', *Journal of Artificial Intelligence*, **4 (1)**, 89–99, (2011).