JOURNAL OF OPTIMIZATION, DIFFERENTIAL EQUATIONS AND THEIR APPLICATIONS (JODEA)
Volume **31**, Issue **2**, December **2023**, pp. 50–66, DOI 10.15421/142311

ISSN (print) 2617–0108
ISSN (on-line) 2663–6824

UNIVARIATE TIME SERIES ANALYSIS WITH HYPER NEURAL ODE

Yevhen V. Koshel*, Vasiliy Ye. Belozyorov†

Abstract. Neural ordinary differential equations (NODE) are ordinary differential equations whose right-hand side is determined by a neural network. Hyper NODE (hNODE) is a special type of neural network architecture, which is aimed at creating such NODE system that regulates its own parameters based on known input data. The article uses a new approach to the study of one-dimensional time series, the basis of which is the hNODE system. This system takes into account the relationship between the input data and its latent representation in the network and uses an explicit parametrization when controlling the latent flow. The proposed model is tested on artificial time series of data. The influence of some activation functions (besides sigmoid and hyperbolic tangent) on the quality of the forecast is also considered.

Key words: hypernetwork, neural ODE, time series analysis, power activation function.

2010 Mathematics Subject Classification: 34A34, 34D20, 37D45, 93A30.

Communicated by Prof. P. Kogut

1. Introduction

Constructing and fitting models that can reliably predict time series data has been a subject of thorough research for many decades. The problem of time series forecasting is important in many fields, from economics and finance to meteorology and biology. The future time series values predicted by a model can be used for increasing the planning horizon or decreasing the incident response time, both of which can be invaluable for business and research.

There are two broad approaches to constructing a model — stochastic and deterministic [2]. The stochastic approach aims to identify the underlying statistical patterns in the time series data and use them to estimate its future values. The main downside of this approach is that it requires the researchers to make a priori assumptions about the statistical distribution of the data. The deterministic approach aims to create a model that doesn't contain random variables in its definition. Deterministic models come in many forms, but the most popular one is an

*Department of Applied Mathematics, Oles Honchar Dnipro National University, 72, Gagarin's Avenue, 49010, Dnipro, Ukraine, eugenefade@gmail.com

†Department of Applied Mathematics, Oles Honchar Dnipro National University, 72, Gagarin's Avenue, 49010, Dnipro, Ukraine, belozvye2017@gmail.com

artificial neural network (ANN). ANNs are universal function approximators that can be used to detect complex patterns in the data and perform either pattern recognition [1], time series forecasting [14], sequence transformation [15], new data generation [5], or other tasks. Even though the ANNs are data-driven and self-adaptive, the researchers are still required to select the proper architecture and size for the model to achieve optimal performance. Pattern recognition capabilities of ANNs were substantially improved by introduction of Convolutional neural networks [12], accurate time series forecasting as well as sequence-to-sequence conversion can be achieved by recurrent, long-short term memory, time-lagged, or seasonal neural networks [9], etc.

In this article, the neural ODE (NODE) architecture is of particular interest. NODE is a new family of residual neural network models that, instead of specifying a discrete sequence of hidden layers, parametrizes the derivative of the hidden state using a neural network [8]. The original paper proposes the NODE as an alternative to the residual neural networks that perform a series of composable transformations to their hidden state:

$$\mathbf{h}_{t+1} = \mathbf{h}_t + f(\mathbf{h}_t, \Theta), \tag{1.1}$$

where $t \in [0..T]$ and $\mathbf{h} \in \mathbb{R}^D$. By viewing the equation (1.1) as an Euler discretization of a continuous transformation, the authors transition from the discrete to a continuous representation of the original sequence:

$$\frac{d\mathbf{h}(t)}{dt} = f(\mathbf{h}, t, \Theta). \tag{1.2}$$

Thus, a residual neural network of infinite depth is achieved, the forward pass of which is calculated as follows:

$$F(x_0, T) = x_0 + \int_0^T f(x, \Theta, t)\, dt, \tag{1.3}$$

where x_0 is the initial point and Θ is a set of parameters.

This approach, however, is not limited to residual networks. The efficient way of backpropagating the errors via reverse-mode automatic differentiation developed by the authors of this model allows construction and training of any kind of neural network that contains a NODE as its component.

One of the disadvantages of the base NODE model is that it is only able to learn one continuous flow F. In other words, the set of parameters Θ is static and cannot react to the new inputs or change with time. This is fairly limiting, especially when building models for time series data that can change their qualitative characteristics over time. Another disadvantage is that the behavior of model (1.3) is largely determined by the dimensionality of the space of inputs because the model structure limits F to only be a homeomorphism of the input space onto itself which is very limiting in the context of univariate time series analysis.

To address these issues and expand the capabilities of NODE-based models, different approaches were proposed. Neural ODE Process [11] model aims to achieve the data-dependence of Θ by adopting a stochastic approach and maintaining an adaptive data-dependent distribution over the underlying ODE. The core idea of this approach is to transition to the latent representation of the modeled data using an encoder network, obtain the set of parameters from the provided context, evolve the initial point in the latent space, and finally decode each point in the resulting trajectory back to the input space using a decoder.

The neurally-controlled ODE [6] (N-CODE) model adopts a deterministic approach by introducing a coupled system for determining the set of parameters at each point in time during integration. It effectively merges the input data with the inferred set of parameters into a single system and evolves it, greatly increasing the dimensionality and expressiveness of the hidden representation of the data.

Both Neural ODE Process and N-CODE approaches are examples of hypernetworks because they infer their set of parameters at runtime based on the input data. However, the Neural ODE Process' way of representing the parameters and the input data is more flexible because they are not being coupled into a single space. This provides the designers of the model with the freedom to both define the dimensionality of the latent space to increase the range of possible behaviors of the model and constrain the values of the parameters to reduce the possibility of unwanted behaviors arising in the system.

The proposed hNODE model is effectively a simplified and deterministic Neural ODE Process with N-CODE-inspired approach to control.

2. hNODE definition

Consider a series of pairs of real values $X = \{(t_0, x_0), (t_1, x_1), \ldots, (t_n, x_n)\}$, $t_{i+1} - t_i = \Delta t$, $X_i = (t_i, x_i)$ that represent the univariate time series data augmented with timestamps at which the data was recorded. The goal of time series analysis is to extract meaningful information from X and enable forecasting of its future values. To accomplish this, a model F that maps the set X onto itself is to be constructed:

$$X^* = F(X, \Theta). \tag{2.1}$$

The model F, governed by the set of parameters Θ, must interpret the sequence X and infer the future values X^*. But if the underlying process that produces the data changes, the model F becomes useless because it is unable to adapt its parameters to continue producing reliable outputs. Consider a simple linear model:

$$F(x, A, b) = Ax + b, \tag{2.2}$$

which takes a number of observations from X and produces a prediction X^*. The parameters of the model are static and are optimized to fit the training data. Control functions $A'(x)$ and $b'(x)$ with parameters θ_A and θ_b, can be introduced

to adjust the parameters depending on the input data:

$$F^*(x, A, b, \theta_A, \theta_b) = (A + A'(x))x + (b + b'(x)) = \mathcal{A}(x)x + \mathcal{B}(x). \tag{2.3}$$

The expressions $\mathcal{A}$ and $\mathcal{B}$ are matrix and vector functions respectively that represent the rules for inferring parameter values for equation (2.2) based on the input values and the set of hyperparameters that define behaviors of functions $A'(x)$ and $b'(x)$. If we combine the hyperparameters into a set $\Theta = \{\theta_A, \theta_b\}$, and denote $\mathcal{H}(X, \Theta)$ as a map from the Cartesian product of the set of hyperparameters Θ and the input values X into the set $\Theta^* = \{A^*, b^*\}$ of adjusted parameters for (2.2), the equation (2.3) becomes

$$F^*(x, A, b, \Theta) = F(x, \mathcal{H}(x, \Theta)) \tag{2.4}$$

If instead of the equation (2.2) one were to use the model (1.3), the dimensionality of the inputs X might become a problem so to increase or decrease it, one may use an extra pair of vector functions — an encoder-decoder couple:

$$\begin{aligned} Y &= \mathcal{E}(X, \theta_{\mathcal{E}}) \\ X &= \mathcal{D}(Y, \theta_{\mathcal{D}}). \end{aligned} \tag{2.5}$$

Definition 2.1. A model $\mathcal{E}$ that disentangles the input data and maps the time series data into the latent space is called an *encoder*. A model $\mathcal{D}$ that interprets the points in the latent space and maps them back onto the time series data space called a *decoder*. A pair of models $\mathcal{E}$ and $\mathcal{D}$ such that $X \equiv \mathcal{D} \circ \mathcal{E}(X)$ is called an *encoder-decoder couple*.

So the model (2.4) becomes

$$F^*(x, A, b, \Theta) = \mathcal{D} \circ F(\mathcal{E}(x, \Theta), \mathcal{H}(x, \Theta)). \tag{2.6}$$

Finally, the hNODE model is defined as (2.6) where function F is the NODE model (1.3):

$$hNODE(X, \Theta) = \mathcal{D} \circ G(\mathcal{E}(X, \Theta), \mathcal{H}(X, \Theta)), \tag{2.7}$$

where $\mathcal{E} : X \times \Theta \to L$ is an encoder that maps the time series data X into the latent space L, $\mathcal{D} : L \to X$ is a decoder that interprets the points from L and maps them into X, $G : L \times \Theta^* \to L$ is a system of ordinary differential equations that evolves the state in the latent space, $\mathcal{H} : X \times \Theta \to \Theta^*$ is a function that produces control rule for G. A visual representation of model (2.7) is provided on Fig. 2.1

3. Activation functions in neural network modeling of time series

It is known that in neural network modeling three types of activation functions are most often used: sigmoid, hyperbolic tangent and rectified linear unit (ReLU). The increased attention to these functions is explained by a number of reasons,

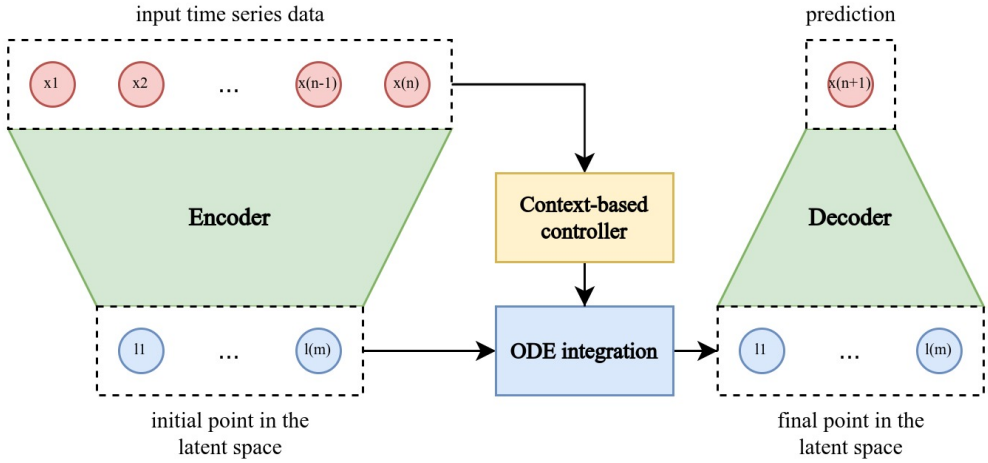

Fig. 2.1. Top-level representation of the model for predicting univariate time series data.

the main one being the stability that they provide to the neural network models. In this regard, in modeling problems it seems interesting to use other activation functions. Naturally, one of the requirements for such functions must be the stability of the resulting neural network models. Further, from the point of view of stability, we will consider power-law activation functions.

Definition 3.1. [4] A set of real functions $\mathbb{F} \subset \mathbf{C}(\mathbb{X})$ is called separating points of the set $\mathbb{X} \subset \mathbb{R}^n$ if for any different $\mathbf{x}_1, \mathbf{x}_2 \in \mathbb{X}$ $(\mathbf{x}_1 \neq \mathbf{x}_2)$, there exists a function $f \in \mathbb{F}$ such that $f(\mathbf{x}_1) \neq f(\mathbf{x}_2)$.

Let $f(w) \in \mathbb{F}$ be the function of one real variable w such that $f(0) = 0$ and

$$\text{either conditions for } f(w): \begin{cases} \text{if } w < 0 \text{ then } f(w) = -\psi(-w) < 0, \\ \text{if } w > 0 \text{ then } f(w) = \phi(w) > 0 \end{cases} \tag{3.1}$$

$$\text{or conditions for } f(w): \begin{cases} \text{if } w < 0 \text{ then } f(w) = \psi(-w) > 0, \\ \text{if } w > 0 \text{ then } f(w) = \phi(w) > 0 \end{cases} \tag{3.2}$$

are fulfilled. (Here $\phi(w), \psi(w)$ are differentiable functions of one variable w.)

Definition 3.2. [4] Representation (3.1), ((3.2)) is called an odd (even) activation function.

For example, the ReLU-like function can be represented in the form:

$$f(w) = \ln \frac{a \cdot \exp(b \cdot w)}{1 + (a-1) \cdot \exp(c \cdot w)}; a > 1, b \geq c > 0.$$

Consider the following system

$$\left\{\begin{array}{lcl} \dot{x}_1(t) &=& f_1(a_{11}x_1 + \ldots + a_{1n}x_n + b_1) \\ \dot{x}_2(t) &=& f_2(a_{21}x_1 + \ldots + a_{2n}x_n + b_2) \\ \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \\ \dot{x}_n(t) &=& f_n(a_{n1}x_1 + \ldots + a_{nn}x_n + b_n), \end{array}\right. \tag{3.3}$$

where a_{ij}, b_i are known real constants; $i, j = 1 \ldots n$.

Along with system (3.3), we will also consider the system

$$\left\{\begin{array}{lcl} \dot{x}_1(t) &=& a_{11}f_1(x_1) + \ldots + a_{1n}f_n(x_n) \\ \dot{x}_2(t) &=& a_{21}f_1(x_1) + \ldots + a_{2n}f_n(x_n) \\ \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \\ \dot{x}_n(t) &=& a_{n1}f_1(x_1) + \ldots + a_{nn}f_n(x_n). \end{array}\right. \tag{3.4}$$

Note that if $\det A \neq 0$, then with the help of the change of variables $\mathbf{x} \to A\mathbf{x} + \mathbf{b}$ it is always possible to pass from system (3.3) to system (3.4) and vice versa; $A \in \mathbb{R}^{n\times n}$, $\mathbf{b} \in \mathbb{R}^n$. Therefore, in what follows, we restrict ourselves to the study of system (3.4).

We introduce the notation

$$\mathbf{f}(\mathbf{x}) = (f_1(x_1), \ldots, f_n(x_n))^T. \tag{3.5}$$

Taking into account (3.5), we introduce the following function

$$V(x_1, \ldots, x_n) = \int f_1(s_1)\, ds_1 + \ldots + \int f_n(s_n)\, ds_n,$$

where the integral is understood in the sense of an indefinite integral.

The integral of the function $\mathbf{f}(\mathbf{x})$ is defined by formulas:

1. If $f_i(s_i)$ is even then

$$\int f_i(s_i)\, ds_i = \left\{\begin{array}{l} \displaystyle\int_0^{x_i} \phi_i(s_i)\, ds_i, \text{ if } x_i \geq 0 \\ \displaystyle\int_{x_i}^{0} \psi_i(s_i)\, ds_i, \text{ if } x_i < 0 \end{array}\right. .$$

2. If $f_i(s_i)$ is odd then

$$\int f_i(s_i)\, ds_i = \left\{\begin{array}{l} \displaystyle\int_0^{x_i} \phi_i(s_i)\, ds_i, \text{ if } x_i \geq 0 \\ \displaystyle -\int_{x_i}^{0} \psi_i(s_i)\, ds_i, \text{ if } x_i < 0 \end{array}\right. .$$

Theorem 3.1. *[3] Assume that in system* (3.3) *the matrix* $A = \{a_{ij}\}$ *is anti-symmetric and invertible:* $A^T + A = 0$ *and* $\det(A) \neq 0$. *Let all components of the vector function* $\mathbf{f}(\mathbf{x})$ *be odd activation functions. Then any solution* $\mathbf{x}(t, \mathbf{x}_0)$ *of system* (3.3) *is bounded and is either periodic or chaotic.*

Proof. It is known [4] that the integral of the odd activation function $f_i(x_i), i = 1, \ldots, n$, is the even activation function. Therefore, we have $V(x_1, \ldots, x_n) \geq 0$.

Compute the total derivative with respect to t of the function $V(x_1, \ldots, x_n)$:

$$\begin{aligned} \dot{V}_t(x_1, \ldots, x_n) &= 0.5\left[\dot{x}_1(t)\tfrac{\partial V(x_1,\ldots,x_n)}{\partial x_1} + \cdots + \dot{x}_n(t)\tfrac{\partial V(x_1,\ldots,x_n)}{\partial x_n}\right] \\ &\quad +0.5\left[\tfrac{\partial V(x_1,\ldots,x_n)}{\partial x_1}\dot{x}_1(t) + \cdots + \tfrac{\partial V(x_1,\ldots,x_n)}{\partial x_n}\dot{x}_1(t)\right] \\ &= 0.5\mathbf{f}^T(\mathbf{x})(A^T + A)\mathbf{f}(\mathbf{x}) = 0. \end{aligned} \tag{3.6}$$

Further, by virtue of inequalities (3.1), (3.2) and the fact that $V(x_1, \ldots, x_n) \geq 0$, we have

$$\lim_{x_i\to\infty}\int_{-x_i}^{x_i} f_i(s_i)\, ds_i = \lim_{x_i\to\infty}\int_{-x_i}^{0} f_i(s_i) ds_i + \lim_{x_i\to\infty}\int_{0}^{x_i} f_i(s_i)\, ds_i \geq 0; i = 1, \ldots, n.$$

This means that $\lim_{\|\mathbf{x}\|\to\infty} V(x_1, \ldots, x_n) \geq 0$. In addition, from (3.6) it follows that for a sufficiently large value $\|\mathbf{x}_0\|$, we have $V(x_1(t), \ldots, x_n(t)) = const = V(x_{10}, \ldots, x_{n0}) = V(\mathbf{x}_0) > 0$. This implies that the set $\mathbb{S} = \{V(x_1(t), \ldots, x_n(t)) - V(x_{10}, \ldots, x_{n0}) = 0\}$ is compact. Therefore, any trajectory $\mathbf{x}(t, \mathbf{x}_0)$ of system (3.3) (or (3.4)) is bounded and is either periodic (if $n \geq 2$) or chaotic (if $n \geq 3$). □

Theorem 3.2. *[3] Assume that in system* (3.3) *the matrix* $A+A^T$ *is non-negative definite. Let also all components of the vector function* $\mathbf{f}(\mathbf{x})$ *be odd activation functions. Then any solution* $\mathbf{x}(t, \mathbf{x}_0)$ *of system* (3.3) *is stable.*

Proof. It's clear that $V(0, \ldots, 0) = 0$. Then, under the conditions of Theorem 3.2, equality (3.6) must be replaced by inequality $\dot{V}_t(x_1, \ldots, x_n) \leq 0$. Now it remains to apply Lyapunov's theorem [2] on the stability of solutions of a system of ordinary differential equations to system (3.4). □

3.1. Generalization of the concept of power activation function

Introduce the following power functions [4]:

$$g(u) = \begin{cases} -(-u)^\beta & \text{if}(u < 0 \text{ and } \beta > 0);\ 0 \text{ if}(u < 0 \text{ and } \beta = 0) \\ u^\alpha & \text{if}(u \geq 0 \text{ and } \alpha > 0);\ 0 \text{ if}(u \geq 0 \text{ and } \alpha = 0) \end{cases} \tag{3.7}$$

or

$$g(u) = \begin{cases} (-u)^\beta & \text{if}(u < 0 \text{ and } \beta > 0);\ 0 \text{ if}(u < 0 \text{ and } \beta = 0) \\ u^\alpha & \text{if}(u \geq 0 \text{ and } \alpha > 0);\ 0 \text{ if}(u \geq 0 \text{ and } \alpha = 0). \end{cases} \tag{3.8}$$

It is clear that representation (3.7) ((3.8)) is an odd (even) activation function.

Formulas (3.7) and (3.8), which introduce power activation functions, have two drawbacks:

1. If $0 < \alpha \leq 1$ or $0 < \beta \leq 1$, then the functions (3.7) and (3.8) are non-differentiable;
2. Functions (3.7) and (3.8) do not take into account the shift of the argument.

In this connection, we introduce the following function (see Fig.3.1):

$$w(u,\alpha,\beta,b,c) = piecewise\Big[u+\frac{b}{c} < -c^{\frac{1}{\beta-1}}, -\frac{\beta-1}{\beta}c^{\frac{\beta}{\beta-1}} - \frac{1}{\beta}\Big(-\Big(u+\frac{b}{c}\Big)\Big)^{\beta}, u+\frac{b}{c} \leq c^{\frac{1}{\alpha-1}}, c\cdot\Big(u+\frac{b}{c}\Big), \frac{\alpha-1}{\alpha}c^{\frac{\alpha}{\alpha-1}} + \frac{1}{\alpha}\Big(u+\frac{b}{c}\Big)^{\alpha}\Big]. \tag{3.9}$$

Here $\alpha > 0$, $\beta > 0$, $\alpha \neq 1$, and $\beta \neq 1$ are degrees; $c > 0$ is the tangent of angle of inclination of a straight line $w = cu + b$; b a given bias of argument.

We put in formula (3.9) $b = 0$. Then we will have

$$w(u,\alpha,\beta,c) = piecewise\Big[u < -c^{\frac{1}{\beta-1}}, -\frac{\beta-1}{\beta}c^{\frac{\beta}{\beta-1}} - \frac{(-u)^{\beta}}{\beta}, u \leq c^{\frac{1}{\alpha-1}}, cu, \frac{\alpha-1}{\alpha}c^{\frac{\alpha}{\alpha-1}} + \frac{u^{\alpha}}{\alpha}\Big]. \tag{3.10}$$

Formula (3.10) can be obtained from formula (3.9) by introducing a new variable $z := u + b/c$, which in (3.10)) is denoted again as $u := z$.

In the optimization problem using gradient methods, it is necessary to use the derivative of the function $w(u,\alpha,\beta,c)$. In the case of $\alpha > 0$, $\beta > 0$, $\alpha \neq 1, \beta \neq 1$, and $c \geq 0$ this formula is as follows:

$$\dot{w}_u(u,\alpha,\beta,c) = piecewise\Big[u < -c^{\frac{1}{\beta-1}}, (-u)^{\beta-1}, u \leq c^{\frac{1}{\alpha-1}}, c, u^{\alpha-1}\Big] \tag{3.11}$$

If $\lim \beta \to 1$, then

$$w(u,\alpha,\beta,c) \to piecewise\Big[u \leq c^{\frac{1}{\alpha-1}}, cu, \frac{\alpha-1}{\alpha}c^{\frac{\alpha}{\alpha-1}} + \frac{u^{\alpha}}{\alpha}\Big],$$

$$\dot{w}_u(u,\alpha,\beta,c) \to piecewise\Big[u \leq c^{\frac{1}{\alpha-1}}, c, u^{\alpha-1}\Big];$$

If $\lim \alpha \to 1$, then

$$w(u,\alpha,\beta,c) \to piecewise\Big[u < -c^{\frac{1}{\beta-1}}, -\frac{\beta-1}{\beta}c^{\frac{\beta}{\beta-1}} - \frac{(-u)^{\beta}}{\beta}, cu\Big],$$

$$\dot{w}_u(u,\alpha,\beta,c) \to piecewise\Big[u < -c^{\frac{1}{\beta-1}}, (-u)^{\beta-1}, c\Big];$$

If $\lim \alpha \to 1$ and $\lim \beta \to 1$, then $w(u,\alpha,\beta,c) \to cu$ and $\dot{w}_u(u,\alpha,\beta,c) \to c$.

Note that formula (3.10) is transformed into formula (3.9) if we put in (3.10) $u := u + b/c$. Thus, we have $w(u + b/c, \alpha, \beta, c) \equiv w(u,\alpha,\beta,b,c)$.

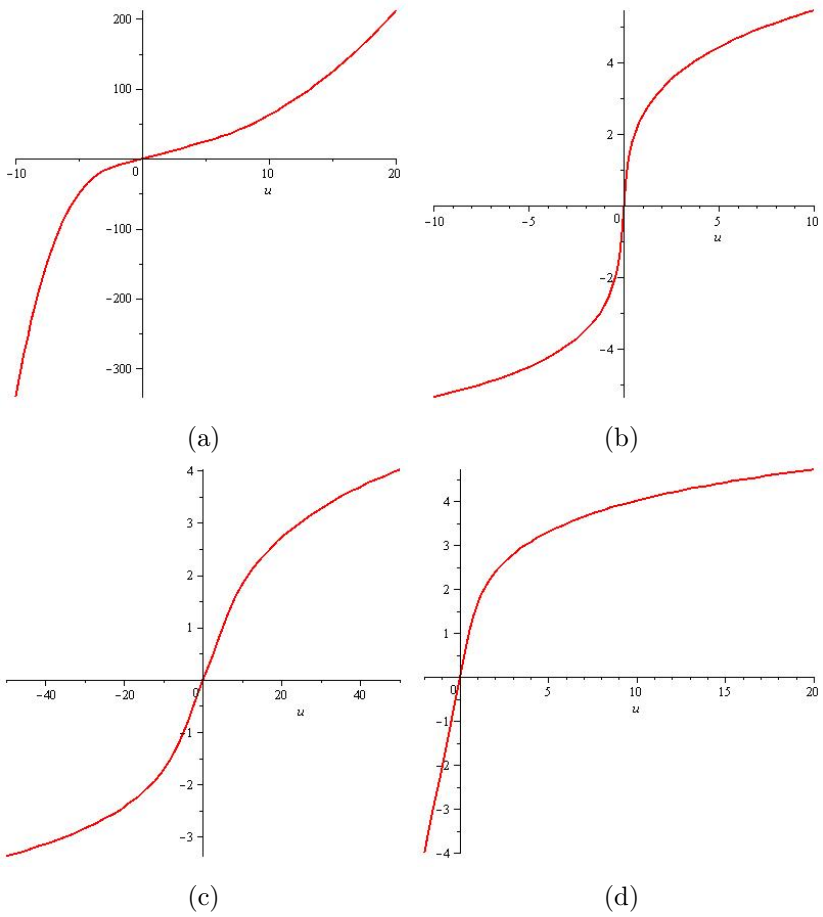

Fig. 3.1. The activation differentiable power function (3.10) for different values of the parameters α, β, and c: (a) $c = 5, \alpha = 2, \beta = 3$; (b) $c = 2, \alpha = 0.01, \beta = 2$; (c) $c = 7, \alpha = 0.3, \beta = 0.1$; (d) $c = 0.2, \alpha = 0.1, \beta = 0.01$.

Finally, if we put $c = 0$ in formula (3.10), then we obtain (with insignificant additions) function (3.7):

$$w(u, \alpha, \beta) = piecewise\Big[u < 0, -\frac{(-u)^{\beta}}{\beta}, \frac{u^{\alpha}}{\alpha}\Big], \tag{3.12}$$

$$\dot{w}_u(u, \alpha, \beta) = piecewise\Big[u < 0, (-u)^{\beta-1}, u^{\alpha-1}\Big]; \alpha > 1, \beta > 1.$$

Note that the functions (3.9) and (3.10) are differentiable on the whole interval $(-\infty, \infty)$ for any $\alpha > 0, \alpha \neq 1$ and $\beta > 0, \beta \neq 1$. At the same time, function (3.12) is non-differentiable for $0 < \alpha \leq 1$ or $0 < \beta \leq 1$, at point $u = 0$. (If $\alpha = \beta = 1$, then we get the linear function $w(u) = u$, which is useless for modeling with the help of neural networks.)

Thus, functions (3.9) and (3.10) are by a generalization of the power odd activation function (3.7) (or(3.12)). This generalization is that function (3.10)

(unlike function (3.7)) is differentiable. Therefore, it becomes possible to use these functions in the gradient methods of search algorithms.

3.2. Example

Consider the example of a generalized cubic root function that is differentiable on the entire real line. To do this we will use formulas (3.10) and (3.11) at $\alpha = \beta = 1/3$; $c = 1$. Then we have

$$w(u) = piecewise\Big[u < -1, 2 - 3(-u)^{1/3}, u \le 1, u, -2 + 3u^{1/3}\Big]$$

and

$$\dot{w}_u(u) = piecewise\Big[u < -1, (-u)^{-2/3}, u \le 1, 1, u^{-2/3}\Big].$$

Thus, the function $w(u)$ is differentiable over the entire interval $(-\infty, \infty)$. The derivative of this function $\dot{w}_u$ is continuous (but not differentiable!) and bounded also over the entire interval $(-\infty, \infty)$: $\dot{w}_u(u) \in (0, 1]$ (see Fig. 3.2).

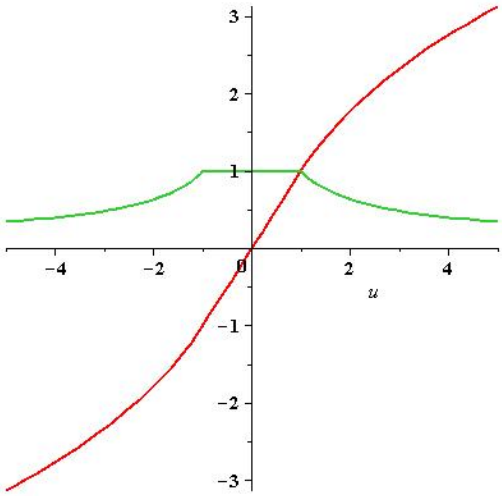

Fig. 3.2. Differentiable power activation function and its derivative: $w(u)$ (red), $\dot{w}_u(u)$(green)

4. Time series prediction

4.1. Direct prediction without explicit parametrization

As mentioned earlier, the main flaw of the basic NODE block is that it restricts the dimensionality of the model by requiring the produced trajectories to be in the same space as the modelled series. Since the NODE block is an autonomous system, in case of modeling the univariate time series data the model must be one-dimensional. This restricts the model's behavior to only 3 basic types depending

on its Lyapunov exponent and makes it impossible to model any periodic or quasi-periodic processes. One way to avoid this issue is to turn the univariate time series into multivariate data by embedding it with estimated minimum embedding dimension d and the lag times $\{\tau_1, \tau_2, \ldots, \tau_{d-1}\}$ by using any of the available delay embedding procedures [10, 13]. This operation will provide the basic context for the model to work with by producing unique combinations of points $X^* \in \mathbb{R}^d$, $X^* = \left\{x_n, x_{n-\tau_1}, \ldots, x_{n-\tau_{d-1}}\right\}$ that can be used to create autoregressive models of type $x_{n+1} = f(X^*, \Theta)$. However, if a regular NODE is used in such a model, the prediction will have dimension d. This can be partially solved by simply discarding all the dimensions except one. The model of this type will impose unnecessary restrictions on the underlying ODE which will essentially be required to fit every coordinate of its trajectories to the same set of data which simply was time-lagged. This is not how most high-dimensional ODEs normally behave. To resolve this issue, the ODE can be allowed to function in its own latent space L that does not impose any such restrictions. The initial values in this latent space can be produced by a more complicated encoder $\mathcal{E}$ that delay-embeds the time series data and applies extra transformations in an attempt to decorrelate the dimensions of X^*.

The encoding step removes the hard coupling of the ODE and the time series data. However, the trajectories produced by the ODE in its latent space L will have to be mapped back to the original one-dimensional time series space. This is handled by the decoder $\mathcal{D}$.

The encoder and decoder make the ODE completely decoupled from the original time series which allows the researchers to freely select its structure and dimensionality. For the purposes of modeling univariate time series data which often exhibits quasi-periodic behavior, it is reasonable to select the model that can capture such behaviors. One such model is based on the AntisymmetricRNN [7] which is given by the equation:

$$\mathbf{h}_n = \mathbf{h}_{n-1} + \epsilon\sigma((\mathbf{W_h} - \mathbf{W_h}^T - \gamma\mathbf{I})\mathbf{h}_{n-1} + \mathbf{V}_h x_n + \mathbf{b_h}), \tag{4.1}$$

where $\mathbf{h}$ is the hidden state, $\mathbf{x}$ is the input, and σ is the activation function.

The idea of AntisymmetricRNN is to structurally enforce the periodicity of the model's trajectories by making sure that the eigenvalues of its Jacobian have either zero or slightly negative real parts. The model (4.1) is designed to prevent the hidden state $\mathbf{h}$ from growing or diminishing rapidly as it is carried from one data point to another. The hNODE model doesn't have hidden state, and it aims to model processes that may exhibit drifting behavior which may require the model's Jacobian eigenvalues to have positive values. With these considerations, the latent ODE G in (2.7) then becomes:

$$G(l, \mathbf{W}, \mathbf{D}, \mathbf{b}) = l_0 + \int_0^T \sigma((\mathbf{W} - \mathbf{W}^T + \mathbf{D})l + \mathbf{b})\, dt, \tag{4.2}$$

where the parameters $\mathbf{W}$, $\mathbf{D}$, and $\mathbf{b}$ are produced by the hypermodel $\mathcal{H}(X, \Theta)$.

W is matrix with unrestricted values that is converted into its antisymmetric form by subtracting a transposed version of it from itself, **D** is a diagonal matrix, and **b** is the bias vector. Pўhe vector $l \in L$ is a point in the latent space L.

Considering the possibility of drift or other qualitative changes that may occur in the time series, the model (4.2) has to be equipped to react to such changes. And this is where the function $\mathcal{H}$ for generating control weights comes in. It produces a new set of parameters for (4.2) each time it is evaluated, allowing it change behavior based on where the current point is in the latent space. The activation function σ used in the examples below was selected as (3.10) with parameters $\alpha = 0.5$, $\beta = 0.3$, and $c = 5$; the plot of the activation function and its derivative is provided on Fig. 4.1.

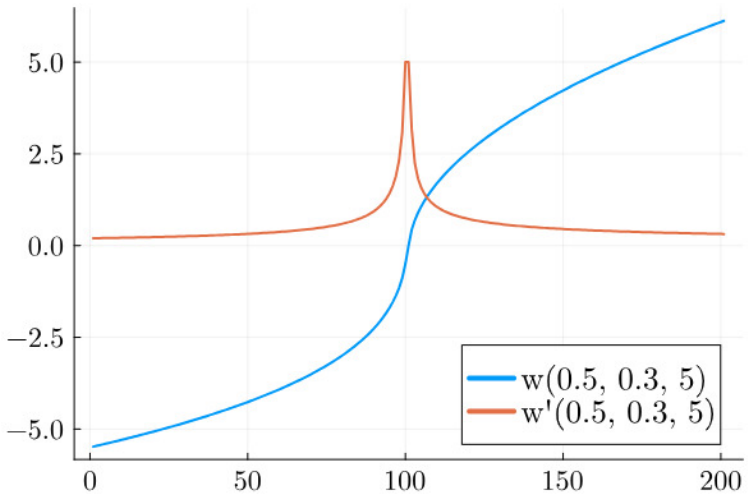

Fig. 4.1. Continuously differentiable activation function and its non-differentiable first derivative.

The final architecture of the hNODE with all the pieces assembled together is given on the Fig. 2.1.

A model with the architecture described above was used to learn and predict the values of the first coordinate of the Lorenz system's ($\dot{x} = \sigma(y - x); \dot{y} = x(\rho - z) - y; \dot{z} = xy - \beta z$) chaotic attractor. The attractor was generated with the parameters $\sigma = 10.0$, $\rho = 28$, $\beta = 8/3$ after which the y and z coordinates were discarded. The remaining x coordinate was delay-embedded with dimension $d = 3$ and delay $\tau = 1$. The resulting dataset split into chunks of size $d \cdot \tau + M$ ($M \geq 1$) and shaped as follows: $\{X_m, Y_m\}$, $m \in [(d-1) \cdot \tau, n - M]$, $X_m = \{x_m, x_{m-\tau}, \ldots, x_{m-(d-1)\cdot\tau}\}$, $Y_m = \{x_{m+1}, x_{m+2}, \ldots, x_{m+M}\}$. In other words, the first $d\tau$ points from a trajectory are picked as input context and the following M points are the expected output of the model.

The output of the model was produced recursively — the input data was used to predict the new point in the series after which the point was added to the

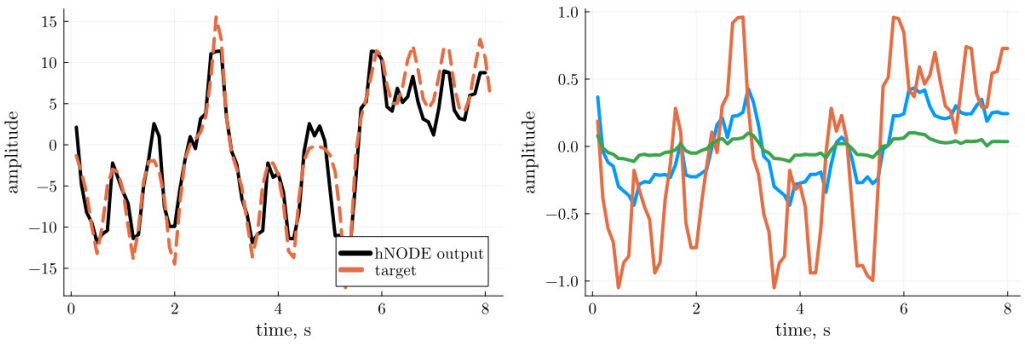

Fig. 4.2. hNODE generates Lorenz-like output (a) from the latent trajectories of the underlying NODE (b).

input data and the first point in the input was removed. This process was applied iteratively until the desired time series prediction was obtained.

4.2. Prediction with explicit parametrization with parameter injection

Since the time series data is completely divorced from the latent space of the nested NODE, it is possible to augment the time series context with extra data to guide the model to specific behaviors. For example, the hNODE model can be trained to generate a sine wave with a specific frequency that is passed to the $\mathcal{H}$ function along with the current context.

As in the previous section, the model (4.2) was used for representing the nested NODE. The time series generated as a sine wave of different frequencies sampled at 8 kHz was delay-embedded with dimension $d = 2$ and delay $\tau = 1$ and the resulting dataset was processed similarly to the Lorenz attractor one. Each input data point X_m was augmented with the frequency F_0 of the waveform it was selected from: $X_m = \{x_m, x_{m-\tau}, F_0\}$. But unlike the previous example, the model's output was produced by obtaining the full trajectory in the latent space and decoding each point all at once which demonstrates that the two approaches are both valid and yield satisfactory results. The model's output for different injected frequencies are provided on the Fig. 4.3.

5. Activation functions performance comparison

As discussed in previous sections, the selection of an activation function is an important consideration when building a model. An incorrectly chosen activation function can slow down the learning rate or even make the model unfit for the problem. To demonstrate this, we selected a basic example similar to the one in the previous section — a sine wave that the model has to approximate without any extra parameters being injected into it. We compare the most widespread activation functions, ReLU and hyperbolic tangent, against the cubic

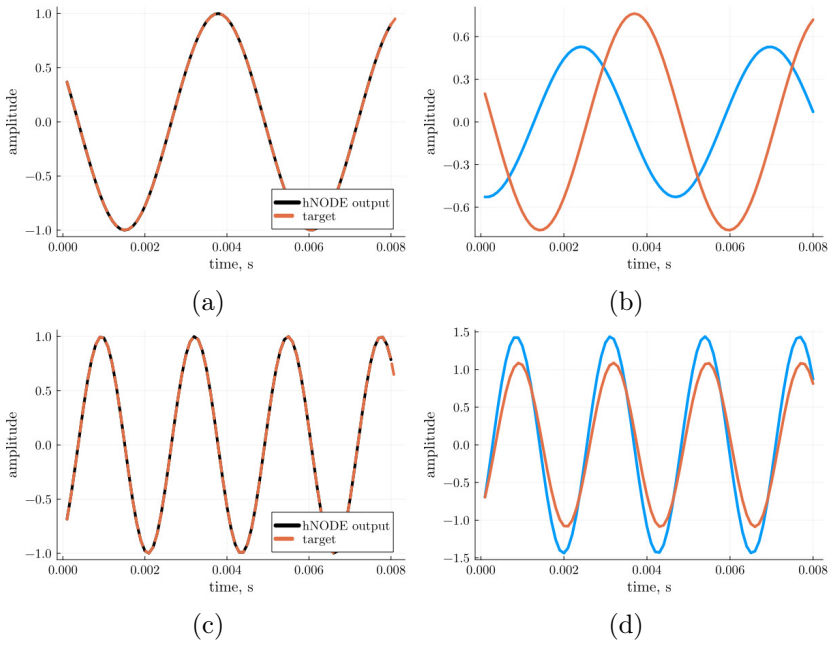

Fig. 4.3. hNODE generates sine waves for injected frequency parameter, the model's output is shown on (a, c) and the latent trajectories of the underlying NODE are shown on (b, d).

root ($cbrt(x) = \sqrt[3]{x}$) and the function (3.10) with parameters $\alpha = 0.5$, $\beta = 0.3$, and $c = 5$ to see how they perform when used inside the NODE nested in hNODE.

Exploiting the fact that the hNODE model consists of several standalone models, we train the encoder and decoder parts of it separately before proceeding to train the nested NODE in the latent space. Each NODE in the comparison was trained on two batches of data where the data points differ in length. The first batch contains pieces of latent trajectory of the length 5 and the second one 50. The rate of learning ε on the first batch is 0.01 and the second one is 0.001. Each model is trained on both batches back-to-back for 10 epochs. The results of the training are provided on Fig. 5.1.

As can be seen on the provided figures, the function (3.10) achieved the best accuracy and converged faster than any other activation function. The second-closest was the hyperbolic tangent which accelerated convergence towards the end of the training. The cubic root training plateaued at around the same point as the piecewise power function but with much worse loss values. The ReLU turned out to be the worst and turned out to be unfit for this particular modeling problem.

6. Conclusion

While the hNODE is versatile and can be adapted to a variety of applications, there are still some caveats that are mostly related to the nature of the underlying NODE model.

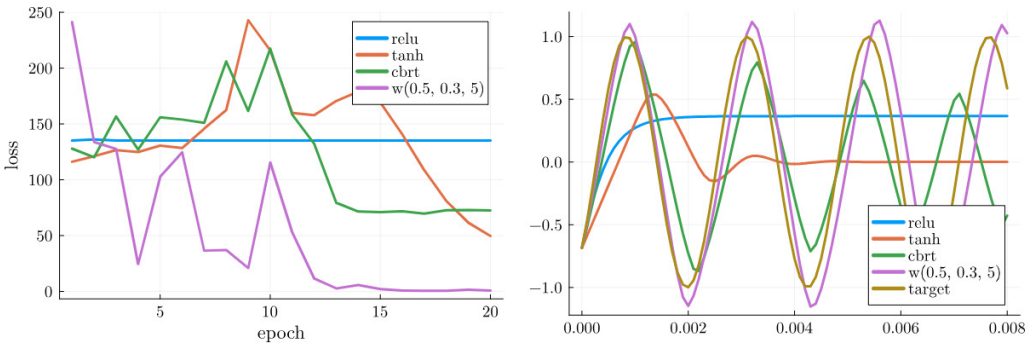

Fig. 5.1. Loss over epochs for different activation functions (a) and trajectories predicted when using those functions (b).

The first caveat is that the latent space trajectories produced by the encoder from the input data cannot intersect as this would violate the theorem about existence and uniqueness of the ODE solution that the model is trying to approximate. An example of malformed latent space trajectories are given on Fig. 6.1.

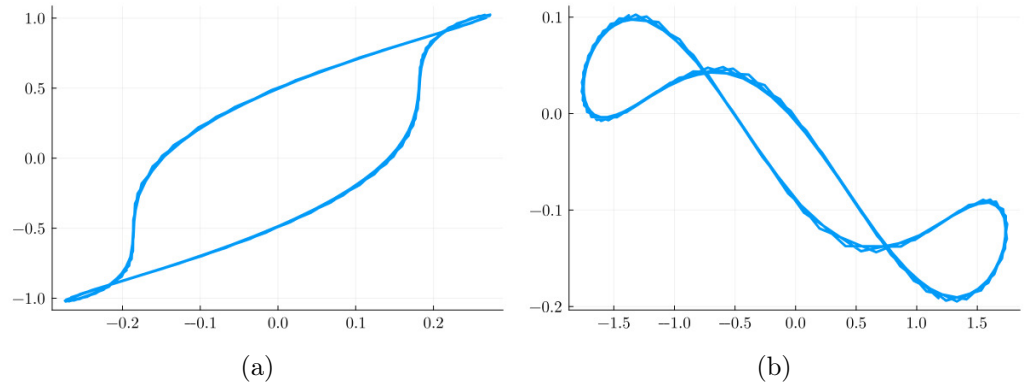

Fig. 6.1. Intersecting trajectories in the latent space generated for the sine data from the previous section.

The second caveat is the integration step used for obtaining numerical solutions for the nested NODE. It may seem reasonable to use the time intervals between the point in the input data as integration step. For example, one might use the inverse of the sampling frequency of the data but such inverse might be too small for the model to handle properly — even simple examples like the sine wave sampled at 8 kHz give unreasonably small integration step of 0.000125. This pitfall is more subtle than the previous one, but it can slow down the learning rate considerably. The example of using different integration steps to learn the example from the previous section with the piecewise power function is given on the Fig. 6.2.

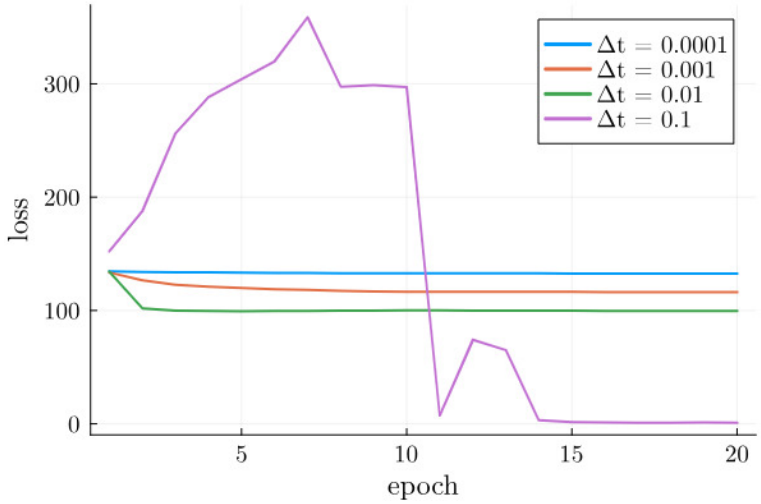

Fig. 6.2. Integration step Δt influencing the rate of learning.

In conclusion, we demonstrated the viability of the new approach to using NODE model by utilizing the latent space mappings using dedicated decoder and encoder models, and learning the dynamics of the modeled process in the latent space instead of the space of the process itself. It was also demonstrated that for the purposes of modeling time series in the latent space it can be beneficial to use continuously differentiable activation functions that do not approach constants when their argument approaches infinity.

References

1. O. I. Abiodun et al, *Comprehensive Review of Artificial Neural Network Applications to Pattern Recognition*, *IEEE Access*, **7** (2019), 158820-158846.
2. R. Adhikari and R. K. Agrawal, *An Introductory Study on Time Series Modeling and Forecasting*, *CoRR*, abs/1302.6613, 2013.
3. V. Ye. Belozyorov and D. V. Dantsev, *Modeling of chaotic processes by means of antisymmetric neural ODEs*, *Journal of Optimization, Differential Equations and Their Applications (JODEA)*, **30**(1)(2022), 1–41.
4. V. Ye. Belozyorov and D. V. Dantsev, *Stability of neural ordinary differential equations with power nonlinearities*, *Journal of Optimization, Differential Equations, and Their Applications*, **28** (2) (2020), 21–46.
5. M. Castelli and L. Manzoni, *Generative models in artificial intelligence and their applications*, *Applied Sciences*, **12** (9) (2022), 4127.
6. M. Chalvidal, M. Ricci, R. VanRullen and T. Serre, *Neural Optimal Control for Representation Learning*, *CoRR*, abs/2006.09545 (2020).
7. B. Chang, M. Chen, E. Haber and E. H. Chi, *AntisymmetricRNN: A Dynamical System View on Recurrent Neural Networks*, *New Journal of Physics*, 1902.09689 (2019).
8. R. T. Q. Chen, Y. Rubanova, J. Bettencourt and D. K. Duvenaud, *Neural Ordinary Differential Equations*, *Advances in Neural Information Processing Systems*, **31** (2018).

9. P. Hendikawati et al, *A survey of time series forecasting from stochastic method to soft computing*, Journal of Physics: Conference Series, **1613** (1) (2020).
10. K. H. Kraemer, G. Datseris, J. Kurths, I. Z. Kiss, J. L. Ocampo-Espindola and N. Marwan, *A unified and automated approach to attractor reconstruction*, New Journal of Physics, **23** (3) (2021), 033017.
11. A. Norcliffe, C. Bodnar, B. Day, J. Moss and P. Liò, *Neural ODE Processes*, *CoRR*, abs/2103.12413 (2018).
12. K. O'Shea and R. Nash, *An Introduction to Convolutional Neural Networks*, *CoRR*, abs/1511.08458 (2015).
13. F. Takens, *Detecting strange attractors in turbulence*, Dynamical Systems and Turbulence, Warwick, 1981, 366–381.
14. G. Zhang, B. E. Patuwo and M. Y. Hu, *Forecasting with artificial neural networks:: The state of the art*, International Journal of Forecasting, **14** (1) (1998), 35-62.
15. J.-X. Zhang, Z.-H. Ling, L.-J. Liu, Y. Jiang and L.-R. Dai, *Sequence-to-Sequence Acoustic Modeling for Voice Conversion*, IEEE/ACM Transactions on Audio, Speech, and Language Processing, **27** (3) (2019), 631-644.

Received 30.08.2023