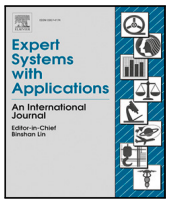
Expert Systems With Applications 273 (2025) 126784

Contents lists available at ScienceDirect

Expert Systems With Applications

journal homepage: www.elsevier.com/locate/eswa



Neural Chronos ODE: Modelling bidirectional temporal patterns in time-series data

C. Coelho [a,*], M. Fernanda P. Costa [a], L.L. Ferrás [a,b,c]

[a] Centre of Mathematics (CMAT), University of Minho, Braga, 4710 - 057, Portugal
[b] Centro de Estudos de Fenómenos de Transporte, Faculty of Engineering, University of Porto, Porto, 4200-465, Portugal
[c] ALiCE, Faculty of Engineering, University of Porto, Porto, 4200-465, Portugal

ARTICLE INFO

MSC:
65L05
65Z05
68T07
34A12
34A34

Keywords:
Neural Networks
Time-series
Neural ODE
Forecasting future
Unveiling past
Neural CODE
Numerical methods

ABSTRACT

This work introduces Neural Chronos Ordinary Differential Equations (Neural CODE), a deep neural network architecture that fits a continuous-time ODE dynamics for predicting the chronology of a system both forward and backward in time. To train the model, we solve the ODE as an initial value problem and a final value problem, similar to Neural ODEs. We also explore two approaches to combining Neural CODE with Recurrent Neural Networks by replacing Neural ODE with Neural CODE (CODE-RNN), and incorporating a bidirectional RNN for full information flow in both time directions (CODE-BiRNN). Variants with other update cells namely GRU and LSTM are also considered and referred to as: CODE-GRU, CODE-BiGRU, CODE-LSTM, CODE-BiLSTM. Experimental results demonstrate that Neural CODE outperforms Neural ODE in learning the dynamics of a spiral forward and backward in time, even with sparser data. We also compare the performance of CODE-RNN/-GRU/-LSTM and CODE-BiRNN/-BiGRU/-BiLSTM against ODE-RNN/-GRU/-LSTM on three real-life time-series data tasks: imputation of missing data for lower and higher dimensional data, and forward and backward extrapolation with shorter and longer time horizons. Our findings show that the proposed architectures converge faster, with CODE-BiRNN/-BiGRU/-BiLSTM consistently outperforming the other architectures on all tasks, achieving a notably smaller mean squared error—often reduced by up to an order of magnitude.

1. Introduction

Neural Ordinary Differential Equations (Neural ODEs) are a type of Neural Networks (NNs) that fit the solution of an ODE to the continuous dynamics of time-series data. Unlike traditional NNs, ODEs are continuous-time functions, allowing predictions to be made at any point in time and handling data taken at arbitrary times (Chen et al., 2018; Rubanova et al., 2019).

Time-series data is commonly used to describe phenomena that change over time and have a fixed order, with consecutive values conveying dependence. However, traditional NNs treat each input as independent, making it difficult to perceive the time relation between consecutive inputs (Hewamalage et al., 2021).

Recurrent Neural Networks (RNNs) were introduced to handle this issue by adding a recurrence mechanism, allowing for the learning of dependencies between current and previous values and handling arbitrary length input and output sequences (Elman, 1990). However, in certain tasks such as natural language processing (NLP) and speech recognition, the current value depends on both previous and future values too, making it important to use future information when making predictions (Mahadevaswamy & Swathi, 2023; Schuster & Paliwal, 1997; Senthilkumar et al., 2022).

Bidirectional Recurrent Neural Networks (BiRNNs) were introduced as an extension to traditional RNNs, allowing for processing of data sequences in both forward and backward directions. This enables combining information from past and future values to make predictions (Schuster & Paliwal, 1997).

Real-world time-series data may have missing values or be sampled irregularly, creating challenges for RNNs that are designed to recognise order but not time intervals (Yang et al., 2022). Preprocessing is often required to convert the data into regularly sampled intervals, which can lead to additional errors and loss of information due to the frequency of sampling (Chen et al., 2018; Rubanova et al., 2019).

To address these challenges, ODE-RNNs were proposed. These are RNNs where state transitions are governed by a continuous-time model adjusted by a Neural ODE, eliminating the need for preprocessing by capturing the time dependence of sequential data (Rubanova et al., 2019).

* Corresponding author.
E-mail addresses: cmartins@cmat.uminho.pt (C. Coelho), mfc@math.uminho.pt (M.F.P. Costa), lferras@fe.up.pt (L.L. Ferrás).

https://doi.org/10.1016/j.eswa.2025.126784
Received 19 December 2024; Received in revised form 29 January 2025; Accepted 4 February 2025
Available online 12 February 2025

Table 1
Evolution of the continuous-time architectures proposed: Neural CODE, CODE-RNN/-GRU/-LSTM and CODE-BiRNN/-BiGRU/BiLSTM.

Authors	Continuous-time architectures	State transitions	Recurrence mechanism
Chen et al. (2018)	Neural ODE	ODE	–
This work	**Neural CODE**	**CODE**	–
Rubanova et al. (2019)	ODE-RNN	ODE	RNN
Rubanova et al. (2019)	ODE-GRU	ODE	GRU
Lechner and Hasani (2020)	ODE-LSTM	ODE	LSTM
This work	**CODE-RNN**	**CODE**	**RNN**
This work	**CODE-GRU**	**CODE**	**GRU**
This work	**CODE-LSTM**	**CODE**	**LSTM**
This work	**CODE-BiRNN**	**CODE**	**BiRNN**
This work	**CODE-BiGRU**	**CODE**	**BiGRU**
This work	**CODE-BiLSTM**	**CODE**	**BiLSTM**

Neural ODEs solve Initial Value Problems (IVPs) by computing the solution curve forward in time from an initial instant t_0 to a final instant t_f. However, in practice, predicting a system's behaviour requires leveraging data both forwards and backwards in time. This motivates the development of Neural Chronos[1] ODE (Neural CODE), a NN architecture that adjusts an ODE to data using information from both initial and final value problems. Neural CODE can predict the state of a system at previous time-steps, unlike traditional Neural ODEs

We compare Neural CODE to Neural ODE in predicting spiral dynamics in both time directions and with varying amounts of training data. Our results show that Neural CODE achieves better generalisation and robustness when predicting forward in time, thanks to the higher amount of information retrieved from the data. Neural CODE also improves backward predictive performance, as expected.

To leverage the advantages of Neural CODE, we refined the ODE-RNN model of Rubanova et al. (2019) by replacing the Neural ODE by a Neural CODE into ODE-RNN (CODE-RNN). Two variants of the architecture were created by changing the update cell type to GRU and LSTM denoted by CODE-GRU and CODE-LSTM, respectively. However, Neural CODE requires both an initial and final condition to solve forward and backward, respectively. This is problematic for the CODE-RNN architecture, which has a single RNN cell that receives an intermediate hidden state $\boldsymbol{h}'_i$ from a Neural CODE and outputs a single hidden state $\boldsymbol{h}_i$. We address this issue by adding a second RNN cell to the CODE-RNN architecture, resulting in a new architecture CODE-BiRNN. This architecture uses two separate RNN cells to process data forwards and backwards in time, allowing for the incorporation of both initial and final conditions. CODE-BiRNN is further refined by creating two variants, CODE-BiGRU and CODE-BiLSTM, which use GRU and LSTM update cells, respectively. Table 1 shows the state transitions and the recurrence mechanism used in each of the continuous-times architectures studied and proposed in this work.

The main contributions of this work are:

- Introduction of Neural CODE, a NN continuous-time architecture able to predict the chronology of a system both forward and backward in time by solving an IVP and a FVP respectively;
- Neural CODE minimises a loss function using predictions computed by an IVP and a FVP, effectively exploiting the interdependencies within the data;
- Introduction of two recurrent architectures, CODE-RNN and CODE-BiRNN, which enhance Neural CODE by updating the ODE dynamics with all training observations, overcoming the limitation of Neural CODE only using the initial and final conditions for training;
- Experiments show Neural CODE outperforms Neural ODE in modelling synthetic and real-world datasets, even with sparse data. CODE-RNN/-GRU/-LSTM and CODE-BiRNN/-BiGRU/-BiLSTM converge faster and consistently outperform ODE-based counterparts on real-world tasks.

[1] Greek word for time.

Section 2 is devoted to the three novel deep NN architectures. First, the Neural CODE architecture is presented along with the mathematical formulation of the initial and final value problems, followed by the proof of existence and uniqueness of solution. Then, the two recurrent architectures CODE-RNN and CODE-BiRNN, are explained in detail, followed by the presentation of their respective variants, CODE-GRU, CODE-LSTM, CODE-BiGRU, CODE-BiLSTM.

In Section 3 we evaluate and compare the performance of the newly proposed NNs namely, Neural CODE, CODE-RNN, CODE-GRU, CODE-LSTM, CODE-BiRNN, CODE-BiGRU and CODE-BiLSTM with the baseline NNs Neural ODE, ODE-RNN, ODE-GRU and ODE-LSTM.

The paper ends with the conclusions and future work in Section 4. Furthermore, we provide an Appendix that presents a brief review of essential concepts such as Neural ODE, RNN, ODE-RNN and BiRNN.

2. Method

In this section, we introduce the innovative architecture called Neural CODE, providing the mathematical details that underlie its workings. Additionally, we demonstrate the versatility of Neural CODE by applying it to enhance the ODE-RNN family through the introduction of two new distinct architectures: CODE-RNN and CODE-BiRNN.

We define $\boldsymbol{X} = (\boldsymbol{x}_1, \boldsymbol{x}_2, \dots, \boldsymbol{x}_N)$ as a time-series of length N, where $\boldsymbol{x}_i \in \mathbb{R}^d$ represents the data vector at time step i ($i = 1, \dots, N$). $\boldsymbol{Y} = (\boldsymbol{y}_1, \boldsymbol{y}_2, \dots, \boldsymbol{y}_N)$ denotes the ground-truth output time-series, where $\boldsymbol{y}_i \in \mathbb{R}^{d^*}$ is the response vector at time step i. Similarly, $\hat{\boldsymbol{Y}} = (\hat{\boldsymbol{y}}_1, \hat{\boldsymbol{y}}_2, \dots, \hat{\boldsymbol{y}}_N)$ represents the sequential output data produced by a NN, with $\hat{\boldsymbol{y}}_i \in \mathbb{R}^{d^*}$ denoting the predicted response vector at time step i.

2.1. Neural CODE

By allowing both past and future information to influence the current prediction, Neural CODE aims to enhance the model's ability to handle long-range dependencies and capture subtle patterns in the data.

In the Neural CODE framework, the continuous dynamics of the system are modelled by the following differential equation:

$$\frac{d\boldsymbol{h}(t)}{dt} = \boldsymbol{f}_\theta(\boldsymbol{h}(t), t),$$

where $\boldsymbol{f}_\theta$ is a NN parametrised by a set of weights and biases, $\boldsymbol{\theta}$ (the architecture of the NN will be presented in the results section). Here, t represents time, and $\boldsymbol{h}(t)$ denotes the hidden state dynamics, which is the solution to the differential equation. The idea is to optimise the function $\boldsymbol{f}_\theta$, by adjusting the parameters $\boldsymbol{\theta}$ using the information given by capturing the dependence between the previous and the current values through forward hidden states $\overrightarrow{\boldsymbol{h}}_i$, as well as the dependence between the next and the current values through backward hidden states $\overleftarrow{\boldsymbol{h}}_i$.

In this context, the Initial Value Problem (IVP) (1),

$$\frac{d\boldsymbol{h}(t)}{dt} = \boldsymbol{f}_\theta(\boldsymbol{h}(t), t) \text{ with } \boldsymbol{h}(t_0) = \boldsymbol{h}_0 \tag{1}$$

with initial condition $\boldsymbol{h}(t_0) = \boldsymbol{h}_0$, is solved in a forward way within the time interval $[t_0, t_f]$,

$$\vec{\boldsymbol{h}}_i = ODESolve(\boldsymbol{f}_\theta, \boldsymbol{h}_f, [t_0, t_f]),$$

using a numerical ODE solver.

For considering possible relationships between the next and current values, it is defined a Final Value Problem (FVP) as follows:

$$\frac{d\boldsymbol{h}(t)}{dt} = \boldsymbol{f}_\theta(\boldsymbol{h}(t), t) \text{ with } \boldsymbol{h}(t_f) = \boldsymbol{h}_f, \tag{2}$$

where the final condition is given by the value at the last time step, $\boldsymbol{h}_f$. Furthermore, the FVP is solved in a backward way within the reverse time interval $[t_f, t_0]$:

$$\overleftarrow{\boldsymbol{h}}_i = ODESolve(\boldsymbol{f}_\theta, \boldsymbol{h}_f, [t_f, t_0]).$$

Note that, by solving the IVP (1) and FVP (2) two sets of solutions are obtained. The first set, denoted as $\vec{\boldsymbol{h}}_i$, represents the solutions computed in the positive direction of time. The second set, $\overleftarrow{\boldsymbol{h}}_i$, corresponds to the solutions computed in the negative direction of time. For a visual representation, see Fig. 1.

Algorithm 1 Neural CODE training process.

Input: start time t_0, end time t_f, initial condition $(\boldsymbol{h}_0, t_0)$, final condition $(\boldsymbol{h}_f, t_f)$, maximum number of iterations $MAXITER$;
Choose $Optimiser$;
$\boldsymbol{f}_\theta \leftarrow DynamicsNN()$;
Initialise θ;
for $k = 1 : MAXITER$ **do**
  $\vec{\boldsymbol{h}}_i \leftarrow ODESolve(\boldsymbol{f}_\theta, \boldsymbol{h}_0, [t_0, t_f])$;
  $\overleftarrow{\boldsymbol{h}}_i \leftarrow ODESolve(\boldsymbol{f}_\theta, \mathbf{h}_f, [t_f, t_0])$;
  Evaluate loss $\mathcal{L}_{\text{total}}$ using (3);
  $\nabla\mathcal{L}_{\text{total}} \leftarrow$ Compute gradients of $\mathcal{L}_{\text{total}}$;
  $\theta \leftarrow Optimiser.Step(\nabla\mathcal{L}_{\text{total}})$;
end for
Return: θ;

To optimise the parameters θ of the NN we designed a loss function given by the sum of two terms, namely the mean squared error (MSE) of the forward and backward predictions, $\mathcal{L}(\vec{\boldsymbol{h}}_i)$ and $\mathcal{L}(\overleftarrow{\boldsymbol{h}}_i)$ respectively:

$$\begin{aligned}\mathcal{L}_{\text{total}}(\theta) =& \mathcal{L}\left(\boldsymbol{h}(t_0) + \int_{t_0}^{t_f} \boldsymbol{f}_\theta(\boldsymbol{h}(t), t)dt\right) + \mathcal{L}\left(\boldsymbol{h}(t_f) + \int_{t_f}^{t_0} \boldsymbol{f}_\theta(\boldsymbol{h}(t), t)dt\right) \\ =& \mathcal{L}\left(ODESolve(\boldsymbol{f}_\theta, \boldsymbol{h}_0, [t_0, t_f])\right) + \mathcal{L}\left(ODESolve(\boldsymbol{f}_\theta, \boldsymbol{h}_f, [t_f, t_0])\right) \\ =& \mathcal{L}(\vec{\boldsymbol{h}}_i) + \mathcal{L}(\overleftarrow{\boldsymbol{h}}_i)\end{aligned} \tag{3}$$

where $\mathcal{L}(\vec{\boldsymbol{h}}_i) = MSE\left(\vec{\hat{\boldsymbol{y}}}, \vec{\boldsymbol{y}}\right)$ and $\mathcal{L}(\overleftarrow{\boldsymbol{h}}_i) = MSE\left(\overleftarrow{\hat{\boldsymbol{y}}}, \overleftarrow{\boldsymbol{y}}\right)$, with $\vec{\boldsymbol{y}}$ and $\overleftarrow{\boldsymbol{y}}$ the ground-truth in the forward and backward directions, and $\vec{\hat{\boldsymbol{y}}}$, $\overleftarrow{\hat{\boldsymbol{y}}}$ the model predictions in the forward and backward directions.

By minimising the loss function $\mathcal{L}_{\text{total}}$, the optimisation process in Neural CODE aims to find the parameters θ that can produce solutions to the IVP that closely match the true values and, simultaneously, solutions to the FVP that also closely match the true values.

Similarly to Neural ODEs, the adjoint sensitivity method is used to compute the gradients of the loss function (Chen et al., 2018; Pontryagin, 2018).

After the optimisation process has been completed in training a Neural CODE (Algorithm 1), the ultimate outcome is the ODE dynamics, adjusted by a NN $\boldsymbol{f}_\theta$. To make predictions, a numerical ODE solver is used, which can solve an IVP by starting at the initial condition and solving forward in time (Algorithm 2), or a FVP by starting at the final condition and solving backward in time (Algorithm 3).

One advantage of Neural CODE is the ability to model the ODE dynamics to effectively capture the data dynamics when solved in both forward (IVP (1)) and backward (FVP (2)) time directions. Therefore, the proposed approach allows for the computation of valid predictions within or outside the training time interval. Within the training time interval, $t \in [t_0, t_f]$, the method is capable of performing data completion tasks, where missing data points are estimated. Moreover, the approach also facilitates prediction of past data, $t \in [t_m, t_f]$, where t_m represents a time preceding time t_0, and prediction of future data, denoted as $t \in [t_0, t_M]$, where t_M represents a time beyond time t_f.

Algorithm 2 Neural CODE for making future predictions, by solving IVP (1).

Input: NN $\boldsymbol{f}_\theta$, start time t_0, end time t_M, initial condition $(\boldsymbol{h}_0, t_0)$;
$\vec{\boldsymbol{h}}_i \leftarrow ODESolve(\boldsymbol{f}_\theta, \boldsymbol{h}_0, [t_0, t_M])$;
$\vec{\hat{\boldsymbol{Y}}} \leftarrow \vec{\boldsymbol{h}}_i$;
Return: $\vec{\hat{\boldsymbol{Y}}}$;

Algorithm 3 Neural CODE for making past predictions, by solving FVP (2).

Input: NN $\boldsymbol{f}_\theta$, start time t_f, end time t_m, final condition $(\boldsymbol{h}_f, t_f)$;
$\overleftarrow{\boldsymbol{h}}_i \leftarrow ODESolve(\boldsymbol{f}_\theta, \mathbf{h}_f, [t_f, t_m])$;
$\overleftarrow{\hat{\boldsymbol{Y}}} \leftarrow \overleftarrow{\boldsymbol{h}}_i$;
Return: $\overleftarrow{\hat{\boldsymbol{Y}}}$;

Remark. We treat the solution $\boldsymbol{h}(t_i) = \boldsymbol{h}_i$ obtained from the ODE solver at time t_i as the predicted value $\hat{\boldsymbol{y}}_i$. However, in some cases, the solutions $\boldsymbol{h}(t_i)$ derived from solving the IVP or FVP may not directly correspond to the desired predictions $\hat{\boldsymbol{y}}_i$. In such situations, an additional NN can be used to process the intermediate solutions $\boldsymbol{h}(t_i)$ and generate the final predictions $\hat{\boldsymbol{y}}_i$.

2.1.1. Existence and uniqueness of the solution

Neural CODE aims to adjust the dynamics of an ODE that makes valid predictions when solved both as an IVP (1) and a FVP (2), originating an Endpoint Value Problem (EVP) (Wang et al., 2018).

To prove the existence and uniqueness of a solution to the EVP we use the Picard–Lindelöf theorem for both the initial and final value problems, (1) and (2) respectively.

Theorem 1. *Let $\boldsymbol{f}_\theta(\boldsymbol{h}, t) : \mathbb{R}^d \times [t_0, t_f] \to \mathbb{R}^d$, be a continuous vector-valued function that is Lipschitz continuous, i.e, there exists a constant K such that for all $\boldsymbol{h}_1, \boldsymbol{h}_2 \in \mathbb{R}^d$ and $t \in [t_0, t_f]$,*

$$\|\boldsymbol{f}_\theta(\boldsymbol{h}_1, t) - \boldsymbol{f}_\theta(\boldsymbol{h}_2, t)\| \leq K\|\boldsymbol{h}_1 - \boldsymbol{h}_2\|.$$

Then, for any initial value $\boldsymbol{h}_0 \in \mathbb{R}^d$, the IVP (1) has a unique solution on the interval $[t_0, t_f]$.

Proof. The theorem can be proved by Picard's Existence and Uniqueness Theorem.

Remark. It should be noted that:

1. By constructing $\boldsymbol{f}_\theta$, given by a NN, we ensure the continuity of the function in both variables since it is a composition of continuous functions. For any pair $(\boldsymbol{h}_i, t_i) \in \mathbb{R}^d \times \in [t_0, t_f]$, there is a neighbourhood such that $\boldsymbol{f}_\theta(\boldsymbol{h}, t)$ is continuous within it.
2. Since the function $\boldsymbol{f}_\theta(\boldsymbol{h}, t)$, given by a NN, is a composition of Lipschitz functions of all layers, each with an associated Lipschitz constant, and the weights are finite, the function is Lipschitz continuous in $\boldsymbol{h}$ with K the product of the Lipschitz constants of its layers, i.e.,

   $$\|\boldsymbol{f}_\theta(\boldsymbol{h}_1, t) - \boldsymbol{f}_\theta(\boldsymbol{h}_2, t)\| \leq K\|\boldsymbol{h}_1 - \boldsymbol{h}_2\|$$

   holds for all $(\boldsymbol{h}_1, t), (\boldsymbol{h}_2, t) \in \mathbb{R}^d \times \in [t_0, t_f]$ (Gouk et al., 2021).

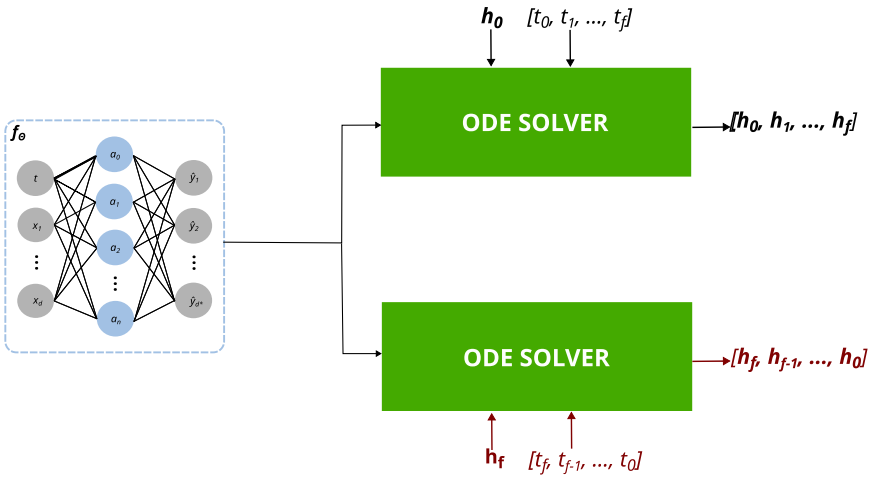


Fig. 1. Diagram of a Neural CODE model capable of making forward predictions by solving an IVP and backward predictions by solving a FVP, in time.

Theorem 2. *Let $f_\theta(h,t) : \mathbb{R}^d \times [t_0, t_f] \to \mathbb{R}^d$, be a continuous vector-valued function that is Lipschitz continuous, i.e, there exists a constant K such that for all $h_1, h_2 \in \mathbb{R}^d$ and $t \in [t_0, t_f]$,*

$$\|f_\theta(h_1,t) - f_\theta(h_2,t)\| \leq K\|h_1 - h_2\|.$$

Then, for any final value $h_f \in \mathbb{R}^d$, the FVP (2) has a unique solution on the interval $[t_0, t_f]$.

Proof. As the form of an IVP (1) and a FVP (2) are similar (Wang et al., 2018), Theorem 2 can also be proved using the Picard's Existence and Uniqueness Theorem. We use time reversibility, and make the change of variable $H(s) = h(t_f - s)$ which results in the IVP, $\frac{dH}{ds} = -f_\theta(H, t_f - s)$, $H(0) = h(t_f) = h_f$. □

Since all the conditions needed for the Picard–Lindelöf theorem are satisfied by both IVP (1) and FVP (2), then it is guaranteed the existence and uniqueness of the solutions.

2.2. Recurrent architectures based on neural CODE

We now demonstrate the versatility of Neural CODE by applying it to enhance the ODE-RNN family through the introduction of two new distinct recurrent architectures: CODE-RNN and CODE-BiRNN. The reason to introduce recurrent architectures based-on Neural CODE is that this architecture adjusts an ODE dynamics which is determined by the initial (IVP (1)) and final condition (FVP (2)). By employing a recurrence mechanism we aim to update the ODE dynamics at each observation, having as many initial and final conditions as time steps, similar to ODE-RNN. The incorporation of a Neural CODE in ODE-RNN makes CODE-RNN and CODE-BiRNN architectures offer distinct advantages in modelling temporal sequences by leveraging Neural CODE's bidirectional capabilities.

CODE-RNN enhances traditional ODE-RNN by integrating Neural CODE's ability to solve both IVPs and FVPs. This enables the model to predict future states while reconstructing past states with greater accuracy. At each time step, CODE-RNN updates its ODE dynamics using both forward and backward information, thereby improving its ability to model temporal dependencies.

CODE-BiRNN extends this concept by incorporating Neural CODE into a bidirectional RNN framework. Similar to traditional bidirectional RNNs, it processes input sequences in both forward and backward directions simultaneously. However, CODE-BiRNN leverages Neural CODE's capacity to model continuous-time dynamics, allowing it to capture hidden states at arbitrary time steps. This ability significantly enhances its flexibility and effectiveness in handling complex temporal patterns. The incorporation of Neural CODE's bidirectional information extraction enables a more comprehensive understanding of temporal dynamics by incorporating both past and future contexts. This is particularly beneficial in applications where the full context of a sequence is critical.

2.2.1. CODE-RNN

As mentioned, with the aim of further improving the performance of predictions in both forward and backward directions of Neural CODE, we propose the recurrent architecture CODE-RNN. This new architecture is built by refining and redesigning the ODE-RNN architecture of Rubanova et al. (2019) in which we replaced the Neural ODE by a Neural CODE.

In CODE-RNN, the intermediate hidden state h'_i in the RNN cell update (A.7), is obtained by combining the forward intermediate hidden state, $\overrightarrow{h'}_i$, and the backward intermediate hidden state, $\overleftarrow{h'}_i$.

The forward intermediate hidden state, $\overrightarrow{h'}_i$, is obtained as the solution of an IVP within the time interval between the previous and current time steps, $[t_{i-1}, t_i]$:

$$\frac{d\overrightarrow{h'}(t)}{dt} = f_\theta(\overrightarrow{h'}(t), t) \text{ with } \overrightarrow{h'}(t_{i-1}) = \overrightarrow{h'}_{i-1}. \tag{4}$$

On the other hand, the backward intermediate hidden state, $\overleftarrow{h'}_i$, is obtained as the solution of a FVP within the time interval between the current and previous time steps, $[t_i, t_{i-1}]$:

$$\frac{d\overleftarrow{h'}(t)}{dt} = f_\theta(\overleftarrow{h'}(t), t) \text{ with } \overleftarrow{h'}(t_{i-1}) = \overleftarrow{h'}_{i-1}. \tag{5}$$

Thus, the solutions are given by:

$$\overrightarrow{h'}_i = ODESolve(f_\theta, \overrightarrow{h'}_{i-1}, [t_{i-1}, t_i]),$$

$$\overleftarrow{h'}_i = ODESolve(f_\theta, \overleftarrow{h'}_{i-1}, [t_i, t_{i-1}]).$$

Then, the two intermediate hidden states $\overrightarrow{h'}_i$ and $\overleftarrow{h'}_i$ are merged to form the full intermediate state, $h'_i = \overrightarrow{h'}_i \oplus \overleftarrow{h'}_i$. Hence, the RNN cell update is:

$$h_i = RNNCell(h'_i, x_i) \text{ with } h'_i = \overrightarrow{h'}_i \oplus \overleftarrow{h'}_i.$$

The training process of CODE-RNN is described in Algorithm 4. It is important to note that in CODE-RNN, the initial and final conditions of both the IVP and FVP are determined by the forward $\overrightarrow{h'}_i$ and backward

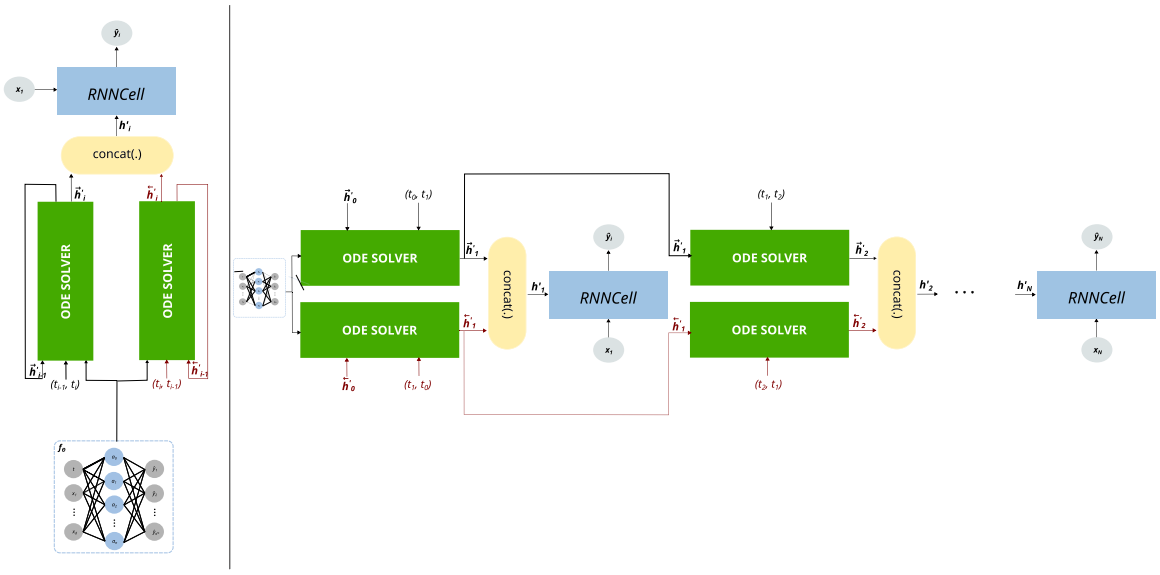


Fig. 2. Training scheme of CODE-RNN and CODE-RNN unfolded through time (on the right). It uses a Neural CODE to adjust an ODE dynamics through the NN f_θ.

$\overleftarrow{h'}_i$ intermediate hidden states. This is in contrast to BiRNNs, where the hidden states that propagate from one time step to another also incorporate the input values x_i. Fig. 2 depicts the architecture of CODE-RNN.

To optimise the parameters θ of CODE-RNN, the loss function is defined by MSE:

$$\mathcal{L}(\theta) = MSE(\hat{Y}, Y). \tag{6}$$

This is a standard loss function commonly used in NNs, as opposed to the loss function (3) specifically designed for Neural CODE.

Algorithm 4 CODE-RNN training process.

Input: Data points and their timestamps $\{(x_i, t_i)\}_{i=1,\dots,N}$, maximum number of iterations $MAXITER$;
$\overrightarrow{h}_0 \leftarrow \mathbf{0}$, $\overleftarrow{h}_0 \leftarrow \mathbf{0}$;
Choose $Optimiser$;
$f_\theta \leftarrow DynamicsNN()$;
Initialise θ;
for $k = 1 : MAXITER$ **do**
  for $i = 1 : N$ **do**
    $\overrightarrow{h'}_i \leftarrow ODESolve(f_\theta, \overrightarrow{h'}_{i-1}, [t_{i-1}, t_i])$;
    $\overleftarrow{h'}_i \leftarrow ODESolve(f_\theta, \overleftarrow{h'}_{i-1}, [t_i, t_{i-1}])$;
    $h'_i \leftarrow \overrightarrow{h'}_i \oplus \overleftarrow{h'}_i$;
    $h_i \leftarrow RNNCell(h'_i, x_i)$;
  end for
  $\hat{Y} \leftarrow h_i$ for all $i = 1, \dots, N$;
  Evaluate loss $\mathcal{L}$ using (6);
  $\nabla\mathcal{L} \leftarrow$ Compute gradients of $\mathcal{L}$;
  $\theta \leftarrow Optimiser.Step(\nabla\mathcal{L})$;
end for
Return: θ;

To make predictions in both forward and backward directions using CODE-RNN, the IVP and FVP are solved to construct the intermediate hidden state h'_i, which is used to compute the output of the RNN update cell. The process of making predictions forward and backward in time follows a similar procedure, with the only difference being the order of the time steps t_i. When making predictions in the forward direction, the time steps are given as $[t_{i-1}, t_i]$ for the IVP (4) and $[t_i, t_{i-1}]$ for the FVP (5), for $i = 1, \dots, M$. Algorithm 5 describes the process of making future predictions with CODE-RNN. However, when making predictions in the backward direction, the time steps are given as $[t_{i-1}, t_i]$ for the IVP (4), and $[t_i, t_{i-1}]$ for the FVP (5), for $i = N, \dots, m$. Algorithm 6 describes the process of making past predictions with CODE-RNN. Similar to Neural CODE, predictions can be computed within ($t \in [t_0, t_f]$) or outside the training time interval for the past ($t \in [t_m, t_f]$) or the future ($t \in [t_0, t_M]$).

Note that, once more we consider that the hidden state h_i outputted by the RNN cell is the prediction $\hat{y}_i$.

Algorithm 5 CODE-RNN for making future predictions.

Input: start time t_0, end time t_M;
$\overrightarrow{h}_0 \leftarrow \mathbf{0}$, $\overleftarrow{h}_0 \leftarrow \mathbf{0}$;
for $i = 1 : M$ **do**
  $\overrightarrow{h'}_i \leftarrow ODESolve(f_\theta, \overrightarrow{h'}_{i-1}, [t_{i-1}, t_i])$;
  $\overleftarrow{h'}_i \leftarrow ODESolve(f_\theta, \overleftarrow{h'}_{i-1}, [t_i, t_{i-1}])$;
  $h'_i \leftarrow \overrightarrow{h'}_i \oplus \overleftarrow{h'}_i$;
  $h_i \leftarrow RNNCell(h'_i, x_i)$;
end for
$\hat{Y} \leftarrow h_i$ for all $i = 1, \dots, M$;
Return: $\hat{Y}$;

Algorithm 6 CODE-RNN for making past predictions.

Input: start time t_N, end time t_m;
$\overrightarrow{h}_{N-1} \leftarrow \mathbf{0}$, $\overleftarrow{h}_{N-1} \leftarrow \mathbf{0}$;
for $i = N : m$ **do**
  $\overrightarrow{h'}_i \leftarrow ODESolve(f_\theta, \overrightarrow{h'}_{i-1}, [t_{i-1}, t_i])$;
  $\overleftarrow{h'}_i \leftarrow ODESolve(f_\theta, \overleftarrow{h'}_{i-1}, [t_i, t_{i-1}])$;
  $h'_i \leftarrow \overrightarrow{h'}_i \oplus \overleftarrow{h'}_i$;
  $h_i \leftarrow RNNCell(h'_i, x_i)$;
end for
$\hat{Y} \leftarrow h_i$ for all $i = N, \dots, m$;
Return: $\hat{Y}$;

In the literature, several variants of update cells have emerged to enhance the performance of the RNN cell. The recurrent architectures based on Neural CODE can incorporate any of these variants. In this study, we conducted experiments with two additional update cells: GRU (CODE-GRU) and LSTM (CODE-LSTM).

In contrast to ODE-RNNs, the hidden state h_i resulting from the RNN cell update is not transmitted to the next cell through the feedback loop. Our approach uses intermediate hidden states in the forward ($\overrightarrow{h'}_i$) and backward ($\overleftarrow{h'}_i$) directions. This is necessary because the initial

and final conditions for solving the ODE cannot be the same, which would occur if we used the hidden state $\boldsymbol{h}_i$ in the feedback loop. This limitation reveals a significant drawback of CODE-RNN/-GRU/-LSTM, as the input information $\boldsymbol{x}_i$ is not considered or passed to the next time step iteration; it is only used to compute the output prediction. To address this issue, we propose an upgraded architecture called CODE-BiRNN.

2.3. CODE-BiRNN

We introduce CODE-BiRNN, an enhanced bidirectional recurrent architecture based on Neural CODE, which is an improvement over CODE-RNN. The aim of this enhancement is to incorporate the input information $\boldsymbol{x}_i$ into the hidden state of the feedback loop, thus updating the ODE dynamics at each observation.

To accomplish this, we made modifications to the architecture. Instead of having a single cell update that generates a single hidden state, $\boldsymbol{h}_i$, we redesigned the structure to include two independent cell updates. The first update computes the forward hidden states, $\overrightarrow{\boldsymbol{h}}_i$, while the second update computes the backward hidden states, $\overleftarrow{\boldsymbol{h}}_i$.

The forward intermediate hidden state $\overrightarrow{\boldsymbol{h}'}_i$ is given by solving the IVP (7) within the time interval between the previous and current time steps, $[t_{i-1}, t_i]$:

$$\frac{d\overrightarrow{\boldsymbol{h}'}(t)}{dt} = \boldsymbol{f}_\theta(\overrightarrow{\boldsymbol{h}}(t), t) \text{ with } \overrightarrow{\boldsymbol{h}}(t_{i-1}) = \overrightarrow{\boldsymbol{h}}_{i-1}. \tag{7}$$

The backward intermediate hidden state, $\overleftarrow{\boldsymbol{h}'}_i$, is obtained by solving the FVP (8) within the time interval between the current and previous time steps, $[t_i, t_{i-1}]$:

$$\frac{d\overleftarrow{\boldsymbol{h}'}(t)}{dt} = \boldsymbol{f}_\theta(\overleftarrow{\boldsymbol{h}}(t), t) \text{ with } \overleftarrow{\boldsymbol{h}}(t_{i-1}) = \overleftarrow{\boldsymbol{h}}_{i-1}. \tag{8}$$

Thus, unlike CODE-RNN, the forward, $\overrightarrow{\boldsymbol{h}'}_i$, and backward, $\overleftarrow{\boldsymbol{h}'}_i$, intermediate hidden states are computed using the previous forward $\overrightarrow{\boldsymbol{h}}_{i-1}$ (9) and backward $\overleftarrow{\boldsymbol{h}}_{i-1}$ (10) hidden states as initial and final value, respectively.

$$\overrightarrow{\boldsymbol{h}'}_i = ODESolve(\boldsymbol{f}_\theta, \overrightarrow{\boldsymbol{h}}_{i-1}, [t_{i-1}, t_i]), \tag{9}$$

$$\overleftarrow{\boldsymbol{h}'}_i = ODESolve(\boldsymbol{f}_\theta, \overleftarrow{\boldsymbol{h}}_{i-1}, [t_i, t_{i-1}]). \tag{10}$$

The two intermediate hidden states $\overrightarrow{\boldsymbol{h}'}_i$ and $\overleftarrow{\boldsymbol{h}'}_i$, are then passed onto separate RNN cells which output the forward $\overrightarrow{\boldsymbol{h}}_i$ (11) and backward $\overleftarrow{\boldsymbol{h}}_i$ (12) hidden states:

$$\overrightarrow{\boldsymbol{h}}_i = RNNCell(\overrightarrow{\boldsymbol{h}'}_i, \boldsymbol{x}_i), \tag{11}$$

$$\overleftarrow{\boldsymbol{h}}_i = RNNCell(\overleftarrow{\boldsymbol{h}'}_i, \boldsymbol{x}_i), \tag{12}$$

To get the predicted output at a given time step, $\hat{\boldsymbol{y}}_i$, the full hidden state, $\boldsymbol{h}_i$, is computed by performing an aggregation/merging operation of the forward $\overrightarrow{\boldsymbol{h}}_i$ and backward $\overleftarrow{\boldsymbol{h}}_i$ hidden states of the same time step t_i:

$$\hat{\boldsymbol{y}}_i = \sigma(\boldsymbol{W}^{[\text{out}]}\boldsymbol{h}_i + \boldsymbol{b}^{[\text{out}]}) \text{ with } \boldsymbol{h}_i = \overrightarrow{\boldsymbol{h}}_i \oplus \overleftarrow{\boldsymbol{h}}_i.$$

By incorporating two separate hidden layers within the RNN, processing the data in opposite temporal directions, our network inherits the name BiRNN. Algorithm 7 outlines the implementation details.

Replacement of the RNN by a BiRNN makes it possible to have the full independent recurrent process for each time direction, Fig. 3.

Similar to CODE-RNN, in CODE-BiRNN, the optimisation process aims to find the optimal parameters $\boldsymbol{\theta}$ by minimising the MSE between the predicted values and the true values.

Algorithm 7 CODE-BiRNN training process.

Input: Data points and their timestamps $\{(\boldsymbol{x}_i, t_i)\}_{i=1,\dots,N}$, maximum number of iterations $MAXITER$;
$\overrightarrow{\boldsymbol{h}}_0 \leftarrow \boldsymbol{0}$, $\overleftarrow{\boldsymbol{h}}_0 \leftarrow \boldsymbol{0}$;
Choose $Optimiser$;
$\boldsymbol{f}_\theta \leftarrow DynamicsNN()$;
Initialise $\boldsymbol{\theta}$;
for $k = 1 : MAXITER$ **do**
  for $i = 1 : N$ **do**
    $\overrightarrow{\boldsymbol{h}'}_i \leftarrow ODESolve(\boldsymbol{f}_\theta, \overrightarrow{\boldsymbol{h}}_{i-1}, [t_{i-1}, t_i])$;
    $\overrightarrow{\boldsymbol{h}}_i \leftarrow RNNCell(\overrightarrow{\boldsymbol{h}'}_i, \boldsymbol{x}_i)$;
  end for
  for $i = 1, \dots, N$ **do**
    $\overleftarrow{\boldsymbol{h}'}_i \leftarrow ODESolve(\boldsymbol{f}_\theta, \overleftarrow{\boldsymbol{h}}_{i-1}, [t_i, t_{i-1}])$;
    $\overleftarrow{\boldsymbol{h}}_i \leftarrow RNNCell(\overleftarrow{\boldsymbol{h}'}_i, \boldsymbol{x}_i)$;
  end for
  $\boldsymbol{h}_i \leftarrow \overrightarrow{\boldsymbol{h}}_i \oplus \overleftarrow{\boldsymbol{h}}_i$;
  $\hat{\boldsymbol{Y}} \leftarrow \boldsymbol{h}_i$ for all $i = 1, \dots, N$;
  Evaluate loss $\mathcal{L}$ using (6);
  $\nabla\mathcal{L} \leftarrow$ Compute gradients of $\mathcal{L}$;
  $\boldsymbol{\theta} \leftarrow Optimiser.Step(\nabla\mathcal{L})$;
end for
Return: $\boldsymbol{\theta}$;

To make predictions in both forward and backward directions using CODE-BiRNN, a similar approach is followed as in CODE-RNN. The IVP (7) and FVP (8) need to be solved to construct the forward and backward outputs of the RNN cells, respectively, which are then used to generate the final hidden state by combining the contributions from both directions. The process of making predictions in forward and backward directions with CODE-BiRNN is analogous to that of CODE-RNN, with the directional difference lying in the order of the time steps. When making predictions in the forward direction, the time steps are given in the pairs $[t_{i-1}, t_i]$ for the IVP and $[t_i, t_{i-1}]$ for the FVP, for $i = 1, \dots, M$. Algorithm 8 describes the process of making future predictions with CODE-BiRNN. On the other hand, when making predictions in the backward direction, the time steps are given in the pairs $[t_{i-1}, t_i]$ for the IVP, and $[t_i, t_{i-1}]$ for the FVP, for $i = N, \dots, m$. Algorithm 9 describes the process of making past predictions with CODE-BiRNN. Predictions can be computed within ($t \in [t_0, t_f]$) or outside the training time interval for the past ($t \in [t_m, t_f]$) or the future ($t \in [t_0, t_M]$).

Note that, once more we consider that the hidden state $\boldsymbol{h}_i$ that results of aggregating/merging the forward and backward outputs of the RNN cells is the prediction $\hat{\boldsymbol{y}}_i$.

Algorithm 8 CODE-BiRNN for making future predictions.

Input: start time t_0, end time t_M;
$\overrightarrow{\boldsymbol{h}}_0 \leftarrow \boldsymbol{0}$, $\overleftarrow{\boldsymbol{h}}_0 \leftarrow \boldsymbol{0}$;
for $i = 1 : M$ **do**
  $\overrightarrow{\boldsymbol{h}'}_i \leftarrow ODESolve(\boldsymbol{f}_\theta, \overrightarrow{\boldsymbol{h}}_{i-1}, [t_{i-1}, t_i])$;
  $\overrightarrow{\boldsymbol{h}}_i \leftarrow RNNCell(\overrightarrow{\boldsymbol{h}'}_i, \boldsymbol{x}_i)$;
  $\overleftarrow{\boldsymbol{h}'}_i \leftarrow ODESolve(\boldsymbol{f}_\theta, \overleftarrow{\boldsymbol{h}}_{i-1}, [t_i, t_{i-1}])$;
  $\overleftarrow{\boldsymbol{h}}_i \leftarrow RNNCell(\overleftarrow{\boldsymbol{h}'}_i, \boldsymbol{x}_i)$;
  $\boldsymbol{h}_i \leftarrow \overrightarrow{\boldsymbol{h}}_i \oplus \overleftarrow{\boldsymbol{h}}_i$;
end for
$\hat{\boldsymbol{Y}} \leftarrow \boldsymbol{h}_i$ for all $i = 1, \dots, M$;
Return: $\hat{\boldsymbol{Y}}$;

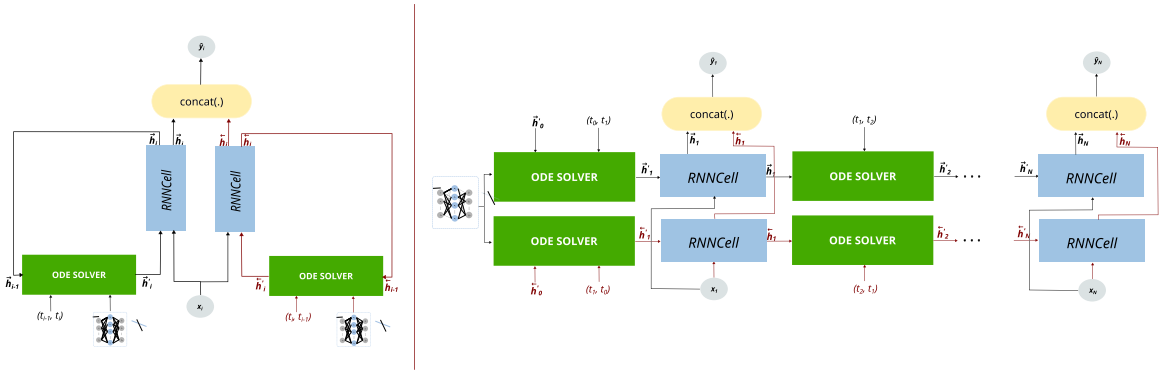


Fig. 3. Scheme of CODE-BiRNN and CODE-BiRNN unfolded through time.

Algorithm 9 CODE-BiRNN for making past predictions.

Input: start time t_N, end time t_m;
$\overrightarrow{\boldsymbol{h}}_{N-1} \leftarrow \mathbf{0}$, $\overleftarrow{\boldsymbol{h}}_{N-1} \leftarrow \mathbf{0}$;
for $i = N : m$ **do**
  $\overrightarrow{\boldsymbol{h'}}_i \leftarrow ODESolve(f_\theta, \overrightarrow{\boldsymbol{h}}_{i-1}, [t_{i-1}, t_i])$;
  $\overrightarrow{\boldsymbol{h}}_i \leftarrow RNNCell(\overrightarrow{\boldsymbol{h'}}_i, \boldsymbol{x}_i)$;
  $\overleftarrow{\boldsymbol{h'}}_i \leftarrow ODESolve(f_\theta, \overleftarrow{\boldsymbol{h}}_{i-1}, [t_i, t_{i-1}])$;
  $\overleftarrow{\boldsymbol{h}}_i \leftarrow RNNCell(\overleftarrow{\boldsymbol{h'}}_i, \boldsymbol{x}_i)$;
  $\boldsymbol{h}_i \leftarrow \overrightarrow{\boldsymbol{h}}_i \oplus \overleftarrow{\boldsymbol{h}}_i$;
end for
$\hat{\boldsymbol{Y}} \leftarrow \boldsymbol{h}_i$ for all $i = N, \ldots, m$;
Return: $\hat{\boldsymbol{Y}}$;

The bidirectional recurrent architecture based on Neural CODE can use any update cell. In this work we present results with RNN (CODE-BiRNN), GRU (CODE-BiGRU) and LSTM (CODE-BiLSTM) cells.

3. Numerical experiments

In order to evaluate the performance of the architectures presented in this work, a set of experiments were conducted using various datasets and tasks. Specifically, we compared Neural CODE to Neural ODE by reconstructing spiral ODE dynamics, both forward and backward in time, using two synthetic datasets that differ in the number of training and testing points. By varying the number of data points, we aimed to investigate how Neural CODE performs in scenarios where data availability is limited, which is often the case in real-world applications.

Then, CODE-RNN/-GRU/-LSTM, CODE-BiRNN/-BiGRU/-BiLSTM were tested and ODE-RNN/-GRU/-LSTM were used as baselines. These comparison were carried out using three real-world time-series datasets: regularly-sampled data from the climate domain, sparsely regularly-sampled data from the hydrological domain, and irregularly-sampled data from the stock market domain.

The evaluation covered three distinct tasks: missing data imputation, future extrapolation, and backward extrapolation (past discovery).

- The first task, missing data imputation, aimed to assess the models' ability to fill in missing values in the time-series data, being crucial when dealing with incomplete datasets where the missing information needs to be estimated;
- The second task, future extrapolation, focused on predicting future values beyond the observed time range, being essential for forecasting and decision-making in many domains;
- The third task, past discovery, aimed to evaluate the models' capacity to uncover and reconstruct past patterns in the time-series data. This task is particularly relevant for analysing historical trends and understanding the dynamics of the data over time.

Furthermore, in the context of the missing data imputation task, we conducted testing using two different settings: one involving a single input/output feature, and the other involving higher-dimensional input/output vectors. This allowed us to leverage the tests conducted on Neural CODE and examine its behaviour when learning from higher-dimensional data. By evaluating the model's performance in capturing the dynamics of the time-series in higher-dimensional settings, we gained insights into its ability to handle and learn from datasets with increased complexity. In the context of future and backward extrapolation, the testing was conducted using sequences of 7 or 15 observations to predict 7 or 15 observations. This allowed us to study the performance of the proposed architectures when predicting for longer time-horizons.

Code implementations of the proposed architectures can be found at https://github.com/CeciliaCoelho/NeuralChronosODE.

3.1. Case study 1: Synthetic spiral ODE dynamics

We compared the performance of Neural CODE and Neural ODE in restoring the dynamics of a spiral ODE both forward and backward in time. By comparing the performance of Neural CODE and Neural ODE on this specific task, we aimed to gain insights into the strengths and limitations of each model in terms of their ability to capture and reproduce complex dynamics. To create the training and testing sets, we defined a simple linear ODE (13) and solved it numerically to retrieve data points in the time interval [0, 25]. For each time step, values for the two coordinates of the spiral are available (x, y).

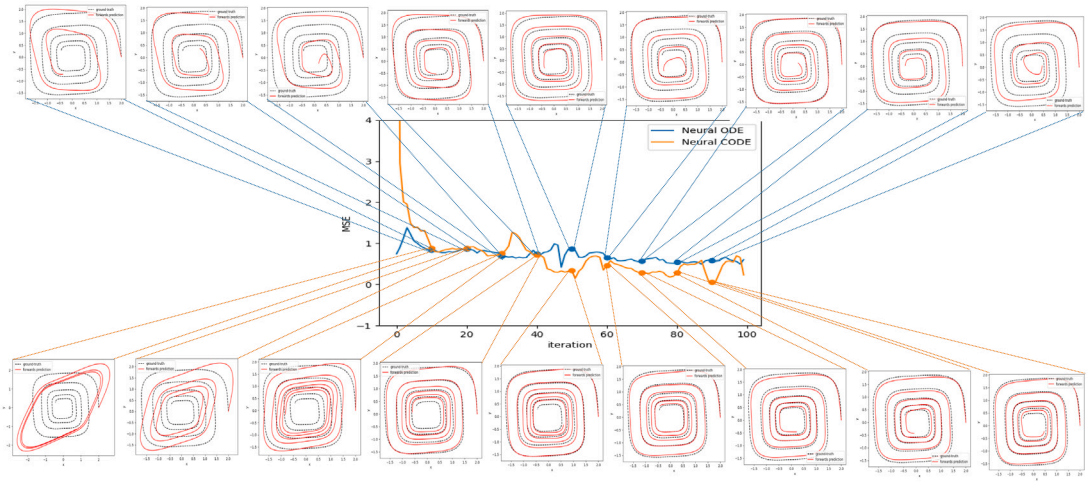
We adapted the code from the *Python Torchdiffeq* (Chen, 2018) for our implementation:

$$\begin{cases} \dfrac{dx}{dt} = -0.1x^3 + 2.0y^3 \\ \dfrac{dy}{dt} = -2.0x^3 - 0.1y^3 \end{cases} \tag{13}$$

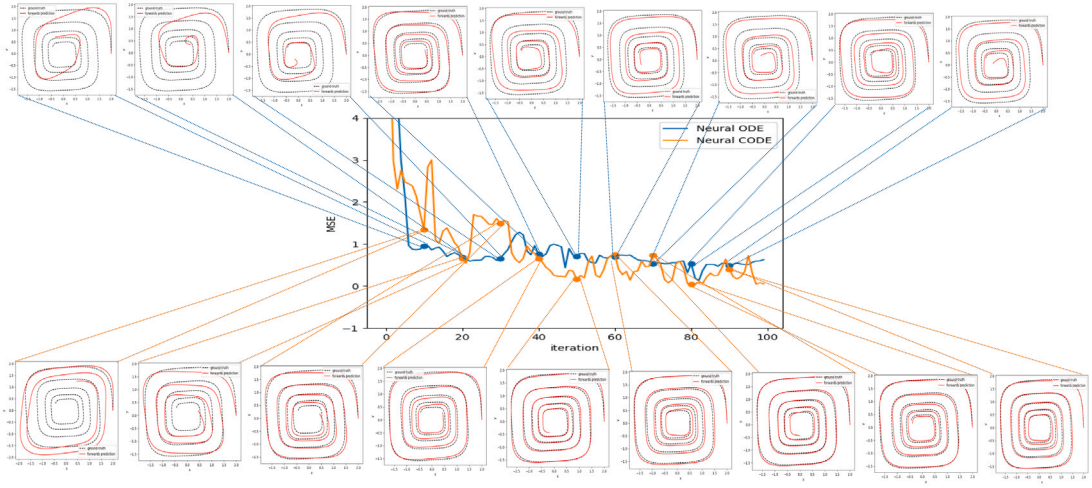
Two datasets were created: one with 2000 training points and 1000 testing points, denoted by 2000/1000 dataset, and another with 1000 training points and 2000 testing points, denoted by 1000/2000 dataset. We aimed to study the impact of dataset size on the models' performance.

For training, we used a batch size of 1, a sequence length of 10 time steps, 2000 iterations for MAXITER, and the Adam optimiser with a learning rate of 0.001. For the $ODESolve$, we employed the Runge–Kutta method of order 5 (Dormand–Prince–Shampine) with default configurations. The loss function of Neural CODE is based on the MSE (see (3)).

The NN architecture used in this study consists of three layers. The input layer contains 2 neurons, representing the input features, (x_i, y_i).



(a) 2000/1000 dataset.



(b) 1000/2000 dataset.

Fig. 4. Training MSE values for forward predictions, with 20 iterations frequency, with the respective forward learnt spiral dynamics, (a) 2000/1000 dataset and (b) 1000/2000 dataset, for Neural ODE (in blue) and Neural CODE (in orange).

The hidden layer is composed of 50 neurons, each using the hyperbolic tangent activation function. Finally, the output layer consists of 2 neurons, representing the output or prediction of the model, $(\hat{x}_i, \hat{y}_i)$.

To account for random weight initialisation, we trained and tested each model three times ($R = 3$). To evaluate the performance of the models, we compute the average of the MSE (MSE_{avg}) and standard deviation (std_{avg}) values for the test sets, from the three runs. Additionally, to analyse the evolution of the adjusted ODE during training, we plotted the MSE for forward and backward prediction on the training set at every 20 iterations (Figs. 4 and 5, respectively).

Forward reconstruction. Table 2 presents the numerical results of the Neural ODE and Neural CODE for the forward reconstruction of the spiral. The first column displays the number of points in the training and testing sets, the second displays the time interval considered for the reconstruction task. The next two columns display, for each model, the MSE_{avg} and std_{avg} values.

The results presented in Table 2 demonstrate that Neural CODE outperforms Neural ODE in reconstructing the spiral through the solution of an IVP. It is worth noting that as the number of training points decreases, there is an expected increase in the prediction error.

In Fig. 4, the plots illustrating the MSE evolution during training demonstrate that Neural CODE achieved lower MSE values in the 2000 iterations, offering faster convergence. Additionally, Neural CODE exhibited faster adaptation to spiral dynamics compared to Neural ODE, providing a good model at iteration 30. By examining the MSE curves for both datasets, depicted in Figs. 4(a) and 4(b), we can conclude that using a smaller training dataset leads to more unstable learning for Neural ODE, while Neural CODE has similar performance regardless of the amount of data.

Backward reconstruction. Table 3 presents the numerical results of the Neural ODE and Neural CODE for the backward reconstruction of the spiral. The first column displays the number of points in the training and testing sets, the second displays the time interval considered for the reconstruction task. The next two columns display, for each model, the MSE_{avg} and std_{avg} values.

The results presented in Table 3 demonstrate that when using the fitted ODEs to predict backward in time by solving a FVP (2), Neural CODE demonstrates superior performance compared to Neural ODE for both datasets: 2000/1000 and 1000/2000. In contrast to the forward reconstruction results presented in Table 2, Neural CODE performs better or at least similarly when trained on a smaller number of data points (1000/2000 dataset).

In Fig. 5, the evolution of MSE during training provides evidence of the increased difficulty faced by Neural ODE when solving the

Table 2
Numerical results on the test spiral sets for predictions forward in time by Neural ODE and Neural CODE, by solving IVP (1).

Training/testing points	$[t_0, t_M]$	Neural ODE $MSE_{avg} \pm std_{avg}$	Neural CODE $MSE_{avg} \pm std_{avg}$
2000/1000	[0, 25]	4.75e−1 ± 1.21e−1	**1.56e−1 ± 8.37e−2**
1000/2000	[0, 25]	5.89e−1 ± 4.48e−2	**3.57e−1 ± 2.29e−1**

Table 3
Numerical results on the test spiral sets for predictions backward in time by Neural ODE and Neural CODE by solving FVP (2).[2]

Training/testing points	$[t_f, t_m]$	Neural ODE $MSE_{avg} \pm std_{avg}$	Neural CODE $MSE_{avg} \pm std_{avg}$
2000/1000	[25, 0]	8.95e0 ± 6.50e0	**2.31 ± 7.88e−1**
1000/2000	[25, 0]	7.87e0 ± 6.82e0	**1.37 ± 7.99e−1**

FVP. This discrepancy can be attributed to the fact that Neural CODE minimises a function that accounts for backward prediction, while Neural ODE does not.

For the 2000/1000 dataset, both architectures exhibit similar loss evolution up to iteration 20. Beyond this point, Neural ODE becomes unstable, displaying large spikes that indicate erratic training behaviour. In contrast, for the 1000/2000 dataset, Neural ODE demonstrates less unstable training. However, Neural CODE consistently shows more stable training and faster convergence across both scenarios.

The improved predictive performance of Neural CODE with the 1000/2000 dataset can be attributed to its ability to achieve a better-generalised representation. This is likely due to the limited number of training points, which reduces the risk of overfitting.

In Figs. 6 and 7, the visualisation of spiral dynamics and the predicted values $(\hat{x}_i, \hat{y}_i)$ with the testing set, of the 2000/1000 dataset, clearly demonstrates that Neural CODE outperforms Neural ODE in terms of prediction accuracy in both forward and backward reconstruction tasks. Furthermore, for the 1000/2000 dataset, the visual representations, in Figs. 8 and 9, confirm our earlier conclusions. In forward time predictions, Neural CODE exhibits superior modelling capabilities compared to Neural ODE. Additionally, the backward dynamics of Neural CODE better aligns with the true dynamics, albeit deviating from the periodic function that models the x and y values. It is important to note that these visualisations were generated using the models at the final training iteration. Upon examining the evolution plots of MSE, it becomes evident that implementing an early stopping criterion would be beneficial.

3.2. Case study 2: Real-world time-series

3.2.1. Neural CODE vs. neural ODE

We evaluated and compared Neural CODE against a Neural ODE baseline, when modelling two real-world time-series datasets (available in *Kaggle*):

- Population Time Series Data (regularly sampled), denoted by PD (Bureau, 2020);
- Air Passenger Data (regularly sampled), denoted by AP (Yeafi, 2022)

For each dataset, the performance was evaluated at two different tasks:

(i) Forward (future) extrapolation: given the initial condition, predict the states at the time steps of the testing set;
(ii) Backward extrapolation: given the final condition, predict the states at the time steps of the testing set reversed in time.

[2] For doing predictions backward in time using a Neural ODE model, the time interval is given in the reverse order to the $ODESolve$.

Remark: For doing predictions backward in time with Neural ODE and Neural CODE, the $ODESolve$ receives the time interval in the reverse order and the final condition, at the last time-step.

The same training conditions were applied to all models for all datasets. The datasets were randomly split into 80% training and 20% testing while maintaining the sequence order. The training was carried out with a batch size of 1, 50 epochs, and the Adam optimiser with a learning rate of 0.0005. For the $ODESolve$ we use the Runge–Kutta method of order 5 (Dormand–Prince–Shampine) with the default configurations. The NN architecture that builds the ODE dynamics used in this study consists of three layers. The input layer contains 1 neuron, representing the input features of the model. The hidden layer is composed of 256 neurons, each using the exponential linear unit activation function. Finally, the output layer consists of 1 neuron, representing the output of the model. To account for random weight initialisation, we trained and tested each model three times ($R = 3$). To evaluate the performance of the models, we computed the average of the MSE (MSE_{avg}) and standard deviation (std_{avg}) values for the test sets, from the three runs. Furthermore, to analyse and compare the convergence of Neural CODE with the Neural ODE baseline, the training losses were plotted.

Population Time Series Data:

The dataset consists of 816 data points with 1 daily feature representing the number of individuals of a population over time. This data was made available by the U.S Census Bureau and is regularly-sampled (Bureau, 2020).

The experimental results are summarised in Table 4, which presents information on the final training loss, as well as the MSE_{avg} and std_{avg} for both testing tasks. Fig. 10 shows the evolution of the loss through training for Neural ODE and Neural CODE.

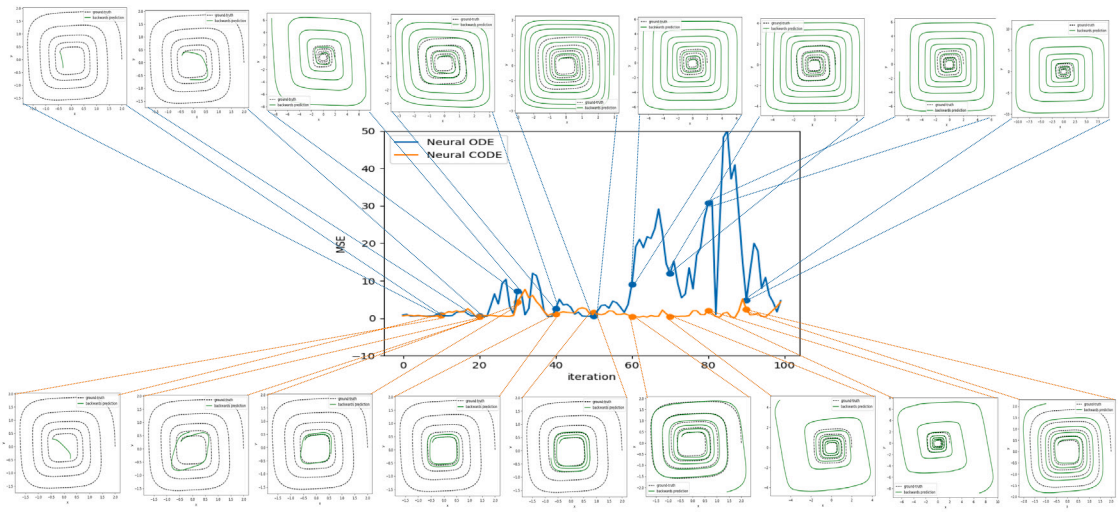
Table 4 shows that while Neural ODE achieves a slightly lower training loss compared to Neural CODE, this advantage does not translate to better testing performance. Neural CODE achieves lower MSE_{avg} in both future and backward extrapolation tasks.

Fig. 10 illustrates that Neural CODE reaches lower loss values quicker than Neural ODE; however, the subsequent training process exhibits some instability. This behaviour is likely due to the minimisation of (3), which comprises two terms, whereas Neural ODE only minimises the error of the fit to the data in the positive direction of time.

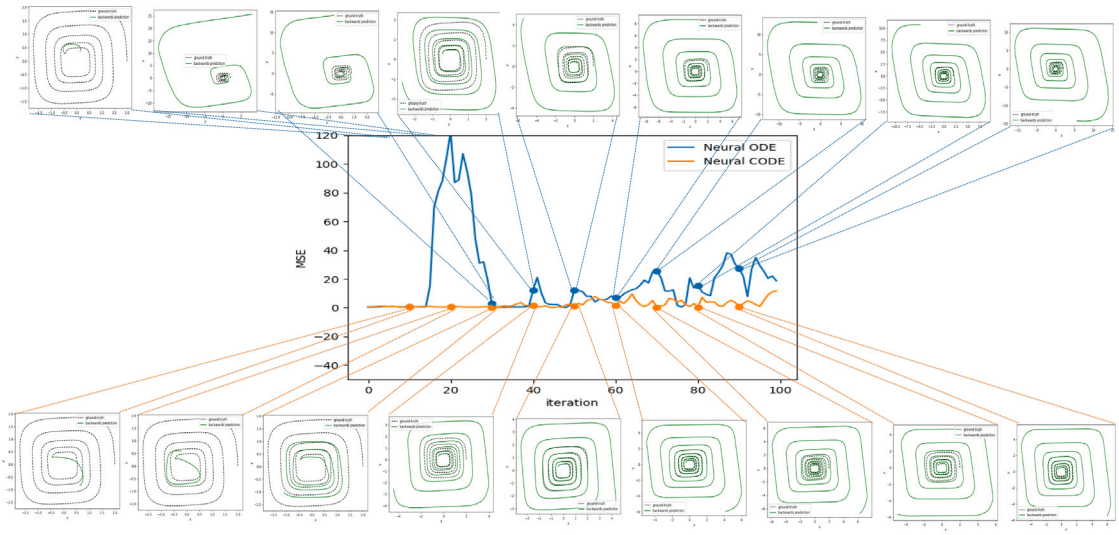
Air Passenger Data:

The dataset consists of 144 data points with 1 monthly feature representing the number of air passengers and is regularly-sampled (Yeafi, 2022).

Table 5 shows that Neural ODE achieves a training loss that is an order of magnitude lower than that of Neural CODE. However, this advantage does not carry over to testing performance, where both Neural ODE and Neural CODE exhibit MSE_{avg} values of the same order of magnitude, with Neural CODE performing slightly better at both tasks.

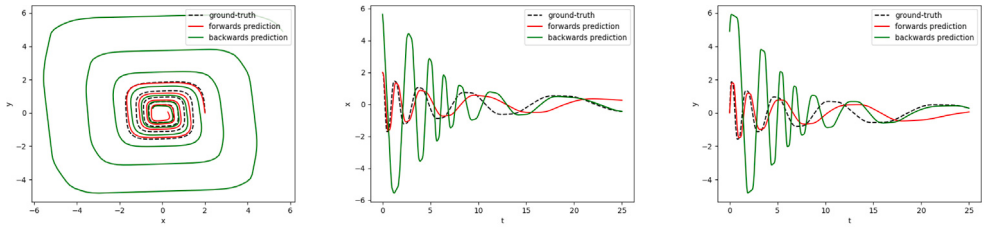


(a) 2000/1000 dataset.



(b) 1000/2000 dataset.

Fig. 5. Training MSE values for backward predictions, with 20 iterations frequency, with the respective backward learnt spiral dynamics, (a) 2000/1000 dataset and (b) 1000/2000 dataset, for Neural ODE (in blue) and Neural CODE (in orange).

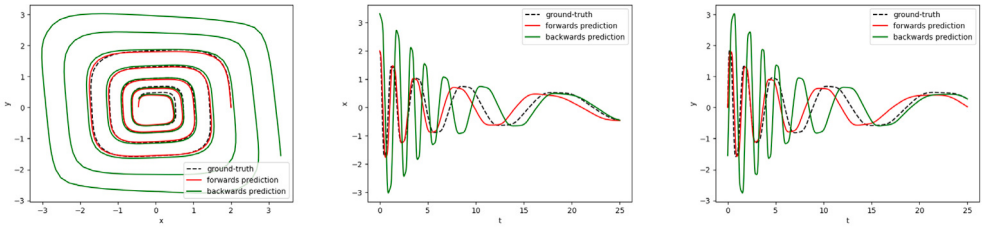


(a) Spiral dynamics. (b) x values prediction. (c) y values prediction.

Fig. 6. Neural ODE at reconstructing the dynamics of a spiral ODE using the 2000/1000 dataset. In dashed black the true dynamics, in red the predictions forward and in green the predictions backward in time.

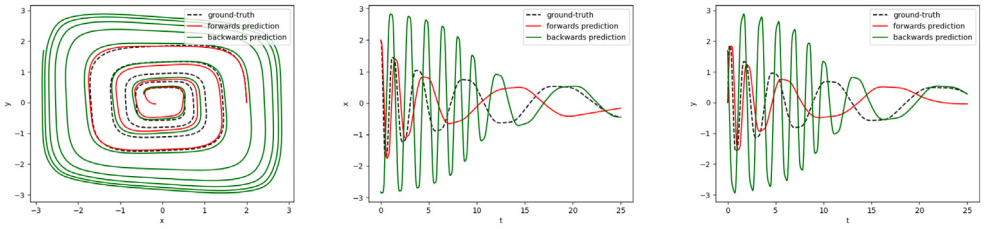
Table 4
Numerical results on the PD when predicting forwards and backwards in time by Neural ODE and Neural CODE.

Task	$[t_0, t_M]$	Neural ODE $MSE_{avg} \pm std_{avg}$	Neural CODE $MSE_{avg} \pm std_{avg}$
Training loss	–	**1.41e−4 ± 8.03e−5**	5.86e−4 ± 1.84e−4
Future extrapolation	[0, 816]	4.24e−3 ± 1.64e−3	**2.16e−3 ± 1.19e−3**
Backwards extrapolation	[816, 0]	3.51e−3 ± 4.39e−6	**3.49e−3 ± 9.28e−6**



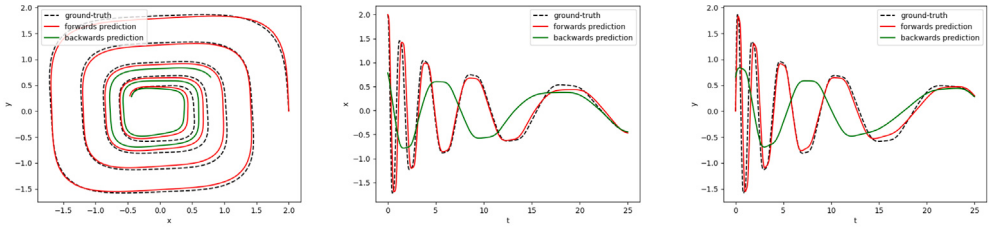
(a) Spiral dynamics. (b) x values prediction. (c) y values prediction.

Fig. 7. Neural CODE at reconstructing the dynamics of a spiral ODE using the 2000/1000 dataset. In dashed black the true dynamics, in red the predictions forward and in green the predictions backward in time.



(a) Spiral dynamics. (b) x values prediction. (c) y values prediction.

Fig. 8. Neural ODE at reconstructing the dynamics of a spiral ODE using the 1000/2000 dataset. In dashed black the true dynamics, in red the predictions forward and in green the predictions backward in time.



(a) Spiral dynamics. (b) x values prediction. (c) y values prediction.

Fig. 9. Neural CODE at reconstructing the dynamics of a spiral ODE using the 1000/2000 dataset. In dashed black the true dynamics, in red the predictions forward and in green the predictions backward in time.

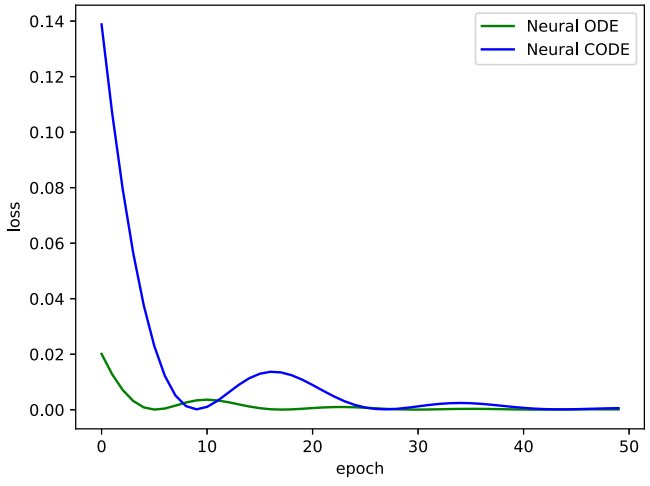


Fig. 10. Training loss through the epochs for the PD dataset.

Fig. 11 further illustrates that Neural CODE reaches lower loss values more quickly, while Neural ODE demonstrates greater overall improvement in loss values over the course of training. These findings continue to suggest that Neural CODE may be effective in mitigating overfitting.

Computational Cost:

Neural CODE is similar to Neural ODE; however, instead of solving an IVP with a numerical method just once, it performs an additional pass to solve an FVP. As a result, the computational cost of Neural CODE is effectively doubled compared to Neural ODE due to the second pass on the solver.

To verify this experimentally, we measured the training time (in seconds) for both Neural CODE and Neural ODE across three runs, calculating the average and the corresponding standard deviation (std_{avg}), for the PD and AP datasets (see Table 6).

These results confirm our theoretical expectation, showing that Neural CODE's training time is approximately 2.24 times that of Neural ODE for the PD dataset and 2.13 times for the AP dataset. The slight variation from an exact 2:1 ratio can be attributed to factors such as system overhead, differences in dataset complexity, minor fluctuations in computational performance, NNs' parameters initialisation and discretisation scheme of the adaptive-solver.

Despite the increased computational cost, Neural CODE demonstrates superior performance in both forward and backward predictions, especially in scenarios with sparse data, as well as showing to

Table 5
Numerical results on the AP dataset when predicting forwards and backwards in time by Neural ODE and Neural CODE.

Task	$[t_0, t_M]$	Neural ODE $MSE_{avg} \pm std_{avg}$	Neural CODE $MSE_{avg} \pm std_{avg}$
Training loss	–	**3.49e−3 ± 1.90e−4**	2.46e−2 ± 2.05e−3
Future extrapolation	[0, 144]	2.33e−2 ± 2.83e−3	**1.62e−2 ± 6.07e−6**
Backwards extrapolation	[144, 0]	2.17e−2 ± 1.13e−4	**2.16e−2 ± 3.43e−5**

Table 6
Training time of Neural ODE and Neural CODE when using the PD and AP datasets.

Dataset	Neural ODE $t_{avg} \pm std_{avg}$	Neural CODE $t_{avg} \pm std_{avg}$
PD	1.14e+2 ± 3.60e−1	2.55e+2 ± 6.80e−1
AP	2.13e+1 ± 7.00e−1	4.51e+1 ± 5.30e−1

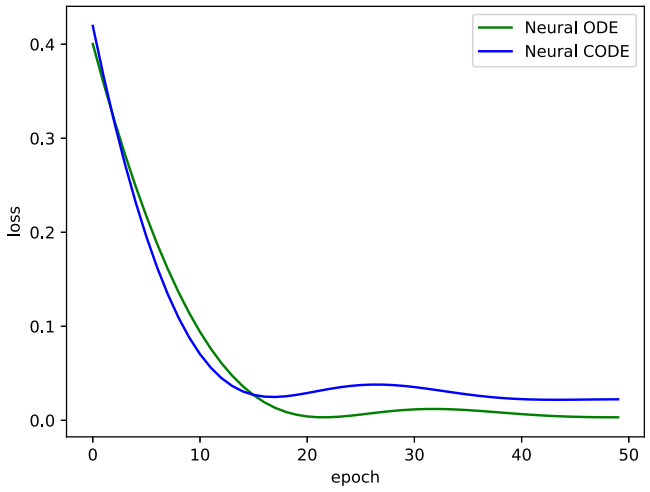


Fig. 11. Training loss through the epochs for the AP dataset.

mitigate overfitting.

3.2.2. Recurrent models

We evaluated and compared the recurrent models CODE-RNN/-GRU/-LSTM and CODE-BiRNN/-BiGRU/-BiLSTM, using ODE-RNN/-GRU/-LSTM baselines, when modelling three real-world time-series datasets with different characteristics (available in *Kaggle*):

- Daily Climate time-series Data (regularly sampled), denoted by DC, V. Rao (2020);
- Hydropower modelling with hydrological data (regularly sampled with sparse data), denoted by HM, De Felice (2021);
- DJIA 30 Stock time-series (irregularly sampled), denoted by DJIA, szrlee (2018).

For each dataset, the performance of the models was analysed at three distinct tasks:

(i) Missing data imputation: predicting an observation $\hat{\boldsymbol{y}}_{i+1}$ at t_{i+1} between observations at t_i and t_{i+2}, for $i = 1, 3, 6 \ldots$.
Remark: Recurrent architectures based-on Neural ODE (ODE-RNN/-GRU/-LSTM) only receive the observation at t_i and predict the observation at t_{i+1} while architectures based-on Neural CODE (CODE-RNN/-GRU/-LSTM and CODE-BiRNN/-BiGRU/-BiLSTM) receive both observations at t_i (for the IVP) and at t_{i+2} (for the FVP) to predict the observation at t_{i+1}, being able to capture more information.
(ii) Forward (future) extrapolation: given a sequence of length 7 or 15 predict the observations for the next 7 or 15 time-steps.
(iii) Backward extrapolation: given a sequence of length 7 or 15 predict the observations for the past 7 or 15 time-steps.
Remark: For doing predictions backward in time with ODE/-RNN/-GRU/-LSTM, the $ODESolve$ receives the time interval in the reverse order.

The same training conditions were applied to all models for all datasets and tasks. The datasets were divided into 75% training and 25% testing points. The training was conducted with a batch size of 1, 50 epochs, and the Adam optimiser with a learning rate of 0.0005. For the $ODESolve$ we use the Runge–Kutta method of order 5 (Dormand–Prince–Shampine) with the default configurations.

The NN architecture that builds the ODE dynamics used in this study consists of three layers. The input layer contains 1 neuron (or 4 neurons when predicting 4 features or 39 neurons when predicting 39 features), representing the input features of the model. The hidden layer is composed of 256 neurons, each using the exponential linear unit activation function. Finally, the output layer consists of 1 neuron (or 4 neurons when predicting 4 features or 39 neurons when predicting 39 features), representing the output of the model.

All architectures use a single RNN, GRU or LSTM cell with an input layer containing 1 neuron (or 4 neurons when predicting 4 features or 39 neurons when predicting 39 features), representing the size of output given by the ODE solver. The hidden layer is composed of 256 neurons and the output layer contains 1 neuron (or 4 neurons when predicting 4 features or 39 neurons when predicting 39 features).

To account for random weight initialisation, we trained and tested each model three times ($R = 3$). To evaluate the performance of the models, we computed the average of the MSE (MSE_{avg}) and standard deviation (std_{avg}) values for the test sets, from the three runs. Furthermore, to analyse and compare the convergence of the proposed networks with the baselines, the training losses of CODE-RNN/-GRU/-LSTM and CODE-BiRNN/-BiGRU/-BiLSTM were plotted, along with ODE-RNN/-GRU/-LSTM baselines, for the missing data imputation and future extrapolation tasks.

Remark. For the backward extrapolation task, the training losses were not plotted since the Neural ODE-based architectures (ODE-RNN/-GRU/-LSTM) solely compute the losses in the forward direction of time.

Daily Climate time-series Data:

This dataset consists of 1462 data points with 4 daily features: mean temperature, humidity, wind speed, and mean pressure. These features were experimentally measured in Delhi, India for weather forecasting (V. Rao, 2020).

Missing data imputation We performed the missing data imputation task to predict data points with 1 feature (mean temperature) as well as all 4 available features, denoted by 1/1 and 4/4 respectively. The numerical results are shown in Table 7 for architectures ODE-RNN, CODE-RNN and CODE-BiRNN, Table 8 for architectures ODE-GRU, CODE-GRU and CODE-BiGRU, and Table 9 for architectures ODE-LSTM, CODE-LSTM and CODE-BiLSTM. The evolution of the loss during training for the predictions with 1/1 and 4/4 features are depicted in

Table 7
Numerical results on the DC dataset at the missing data imputation task, for ODE-RNN, CODE-RNN and CODE-BiRNN.

Seen/predict features	$[t_0, t_f]$	ODE-RNN $MSE_{avg} \pm std_{avg}$	CODE-RNN $MSE_{avg} \pm std_{avg}$	CODE-BiRNN $MSE_{avg} \pm std_{avg}$
1/1	[0, 1462]	1.15e−2 ± 4.15e−6	1.14e−2 ± 4.86e−6	**7.20e−3 ± 6.03e−6**
4/4	[0, 1462]	1.80e−2 ± 2.50e−5	1.80e−2 ± 2.10e−5	**1.23e−2 ± 4.54e−5**

Table 8
Numerical results on the DC dataset at the missing data imputation task, for ODE-GRU, CODE-GRU and CODE-BiGRU.

Seen/predict features	$[t_0, t_f]$	ODE-GRU $MSE_{avg} \pm std_{avg}$	CODE-GRU $MSE_{avg} \pm std_{avg}$	CODE-BiGRU $MSE_{avg} \pm std_{avg}$
1/1	[0, 1462]	1.15e−2 ± 4.54e−6	1.14e−2 ± 6.74e−6	**7.10e−3 ± 4.85e−6**
4/4	[0, 1462]	1.77e−2 ± 2.47e−5	1.80e−2 ± 7.15e−6	**1.21e−2 ± 2.24e−5**

Table 9
Numerical results on the DC dataset at the missing data imputation task, for ODE-LSTM, CODE-LSTM and CODE-BiLSTM.

Seen/predict features	$[t_0, t_f]$	ODE-LSTM $MSE_{avg} \pm std_{avg}$	CODE-LSTM $MSE_{avg} \pm std_{avg}$	CODE-BiLSTM $MSE_{avg} \pm std_{avg}$
1/1	[0, 1462]	1.11e−2 ± 1.30e−5	0.0110 ± 2.42e−5	**7.90e−3 ± 8.29e−6**
4/4	[0, 1462]	1.73e−2 ± 1.31e−5	0.0173 ± 6.20e−6	**1.29e−2 ± 7.84e−5**

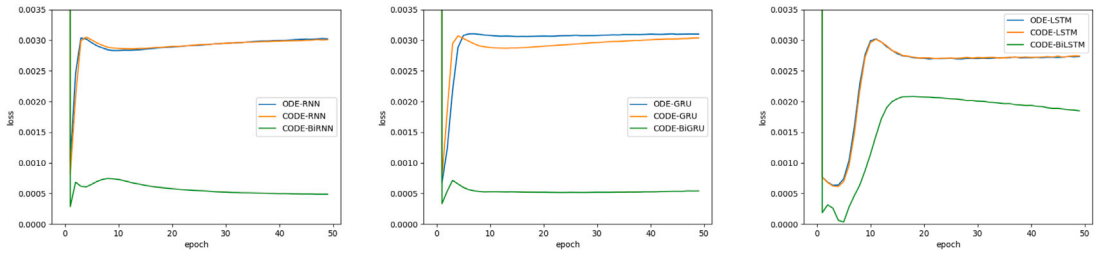


Fig. 12. Training loss through the epochs for the missing data imputation task for the DC dataset, for 1/1. (a) ODE-RNN, CODE-RNN, CODE-BiRNN; (b) ODE-GRU, CODE-GRU, CODE-BiGRU; (c) ODE-LSTM, CODE-LSTM, CODE-BiLSTM.

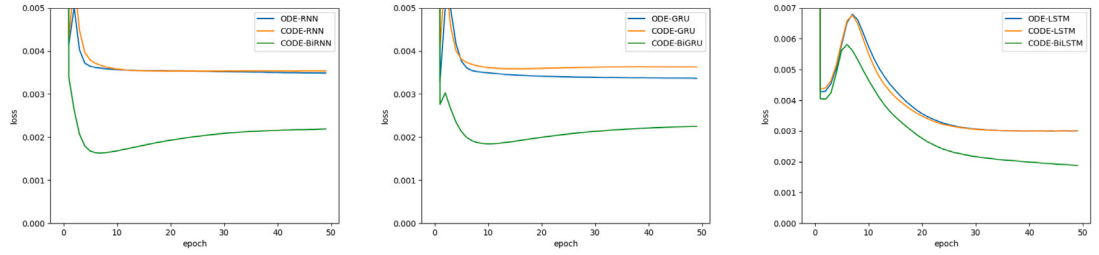


Fig. 13. Training loss through the epochs for the missing data imputation task for the DC dataset, for 4/4. (a) ODE-RNN, CODE-RNN, CODE-BiRNN; (b) ODE-GRU, CODE-GRU, CODE-BiGRU; (c) ODE-LSTM, CODE-LSTM, CODE-BiLSTM.

Figs. 12 and 13, respectively.

Based on Tables 7–9, the CODE-BiRNN, CODE-BiGRU, and CODE-BiLSTM models demonstrate superior performance, outperforming the other approaches. Among these three variants, they present similar performance. They achieve MSE_{avg} values smaller by an order of magnitude for 1/1 and 4/4. When comparing ODE-RNN/-GRU/-LSTM with CODE-RNN/-GRU/-LSTM the performances are similar. In general, all architectures exhibit a higher MSE_{avg} values when the number of features increased.

Figs. 12 and 13 highlight the superior performance of CODE-BiRNN, CODE-BiGRU, and CODE-BiLSTM models, which exhibit faster convergence and achieve lower loss values. When comparing the loss values of ODE-RNN, ODE-GRU, ODE-LSTM, and CODE-RNN, CODE-GRU, CODE-LSTM, their behaviour appears similar. Furthermore, the analysis of loss values throughout the training process aligns with the test results presented in Tables 7–9.

Future extrapolation We performed the future extrapolation task to predict the mean temperature for the next 7 or 15 days after receiving the mean temperature for past 7 or 15 days, denoted by 7/7 and 15/15. The numerical results are shown in Table 10 for architectures ODE-RNN, CODE-RNN and CODE-BiRNN, Table 11 for architectures ODE-GRU, CODE-GRU, CODE-BiGRU and Table 12 for architectures ODE-LSTM, CODE-LSTM and CODE-BiLSTM. The evolution of the losses during training for predictions with 7/7 and 15/15 are depicted in Figs. 14 and 15, respectively.

The results in Tables 10–12 indicate that CODE-BiRNN, CODE-BiGRU, and CODE-BiLSTM consistently deliver superior predictive performance for both 7/7 and 15/15 scenarios. Among these variants, CODE-BiGRU achieves the best overall performance. Additionally, CODE-RNN and ODE-RNN demonstrate comparable results, with CODE-RNN slightly outperforming CODE-GRU and CODE-LSTM for the 7/7 case. Moreover, CODE-RNN, CODE-GRU, and CODE-LSTM

Table 10
Numerical results on the DC dataset at the future extrapolation task, for ODE-RNN, CODE-RNN and CODE-BiRNN.

Days seen/predict	$[t_0, t_f]$	ODE-RNN $MSE_{avg} \pm std_{avg}$	CODE-RNN $MSE_{avg} \pm std_{avg}$	CODE-BiRNN $MSE_{avg} \pm std_{avg}$
7/7	[0, 1462]	1.56e−2 ± 7.00e−4	1.51e−2 ± 3.00e−4	**5.90e−3 ± 2.40e−3**
15/15	[0, 1462]	6.30e−2 ± 3.82e−3	3.81e−2 ± 5.60e−4	**5.60e−4 ± 4.00e−5**

Table 11
Numerical results on the DC dataset at the future extrapolation task, for ODE-GRU, CODE-GRU and CODE-BiGRU.

Days seen/predict	$[t_0, t_f]$	ODE-GRU $MSE_{avg} \pm std_{avg}$	CODE-GRU $MSE_{avg} \pm std_{avg}$	CODE-BiGRU $MSE_{avg} \pm std_{avg}$
7/7	[0, 1462]	1.41e−2 ± 4.00e−5	1.53e−2 ± 3.00e−4	**2.00e−3 ± 1.00e−4**
15/15	[0, 1462]	9.52e−2 ± 1.57e−2	3.81e−2 ± 2.00e−4	**1.34e−3 ± 2.10e−4**

Table 12
Numerical results on the DC dataset at the future extrapolation task, for ODE-LSTM, CODE-LSTM and CODE-BiLSTM.

Days seen/predict	$[t_0, t_f]$	ODE-LSTM $MSE_{avg} \pm std_{avg}$	CODE-LSTM $MSE_{avg} \pm std_{avg}$	CODE-BiLSTM $MSE_{avg} \pm std_{avg}$
7/7	[0, 1462]	1.88e−2 ± 2.00e−4	3.87e−2 ± 3.00e−4	**4.70e−3 ± 2.00e−4**
15/15	[0, 1462]	7.24e−2 ± 4.40e−4	8.51e−2 ± 3.80e−3	**1.08e−2 ± 3.30e−4**

Table 13
Numerical results on the DC dataset at the backward extrapolation task, for ODE-RNN, CODE-RNN and CODE-BiRNN.

Days seen/predict	$[t_f, t_0]$	ODE-RNN $MSE_{avg} \pm std_{avg}$	CODE-RNN $MSE_{avg} \pm std_{avg}$	CODE-BiRNN $MSE_{avg} \pm std_{avg}$
7/7	[1462, 0]	6.70e−2 ± 2.40e−2	**1.50e−2 ± 2.00e−4**	2.23e−2 ± 9.30e−3
15/15	[1462, 0]	5.89e−2 ± 3.98e−3	3.38e−2 ± 5.60e−4	**6.93e−3 ± 4.00e−5**

Table 14
Numerical results on the DC dataset at the backward extrapolation task, for ODE-GRU, CODE-GRU and CODE-BiGRU.

Days seen/predict	$[t_f, t_0]$	ODE-GRU $MSE_{avg} \pm std_{avg}$	CODE-GRU $MSE_{avg} \pm std_{avg}$	CODE-BiGRU $MSE_{avg} \pm std_{avg}$
7/7	[1462, 0]	1.48e−2 ± 1.00e−4	1.51e−2 ± 2.00e−4	**8.20e−3 ± 1.00e−4**
15/15	[1462, 0]	9.28e−2 ± 1.99e−2	3.37e−2 ± 1.70e−4	**7.44e−3 ± 1.2e−4**

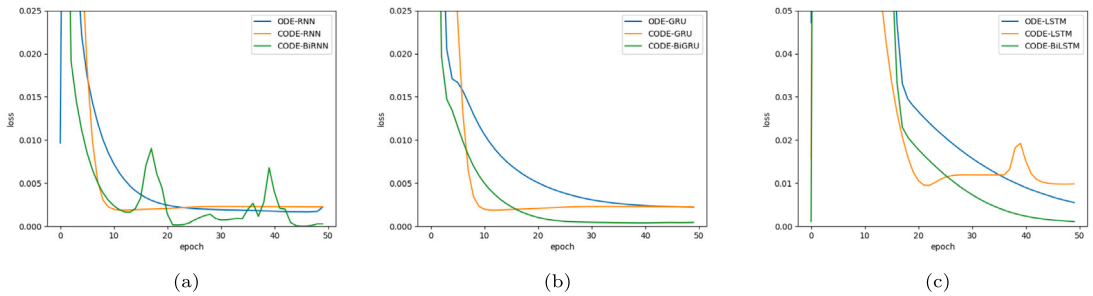


Fig. 14. Training loss through the epochs for the future extrapolation task for the DC dataset, for 7/7. (a) ODE-RNN, CODE-RNN, CODE-BiRNN; (b) ODE-GRU, CODE-GRU, CODE-BiGRU; (c) ODE-LSTM, CODE-LSTM, CODE-BiLSTM.

outperform their ODE-based counterparts (ODE-RNN, ODE-GRU, and ODE-LSTM) for the 15/15 scenario.

Figs. 14 and 15 show that CODE-BiRNN, CODE-BiGRU, and CODE-BiLSTM generally exhibit the fastest convergence rates and achieve the lowest loss values. As the number of epochs increases, CODE-RNN, CODE-GRU, and CODE-LSTM display a slight upward trend in loss values. This observation suggests that incorporating an early stopping criterion could improve the performance of CODE-RNN, CODE-GRU, and CODE-LSTM.

Backward extrapolation We performed the backward extrapolation task to predict the mean temperature for the past 7 or 15 days after receiving the mean temperature for the next 7 or 15 days, denoted by 7/7 and 15/15. The numerical results are shown in Table 13 for architectures ODE-RNN, CODE-RNN and CODE-BiRNN, Table 14 for architectures ODE-GRU, CODE-GRU, CODE-BiGRU and Table 15 for architectures ODE-LSTM, CODE-LSTM and CODE-BiLSTM.

From Tables 13–15, the performance of the models on the backward extrapolation task are similar to that observed for the forward extrapolation task. Once again, it is worth emphasising that CODE-BiRNN/-BiGRU/-BiLSTM demonstrates the highest performance for 7/7 and 15/15. Among these variants, CODE-BiGRU performs best at 7/7, while CODE-BiRNN excels at 15/15.

Although predicting longer time horizons is more challenging, CODE-BiRNN,/-BiGRU/-BiLSTM achieve lower MSE_{avg} values compared to smaller time horizons.

Hydropower Modelling with Hydrological Data:

This dataset consists of weekly hydrological data from 27 European countries spanning the period 2015 to 2019, totalling 208 data points. The dataset includes hydropower inflows and average river discharges measured at national hydropower plants (De Felice, 2021). This dataset was specifically chosen for its limited amount of data, enabling the evaluation of how sparse training data affects the performance of the

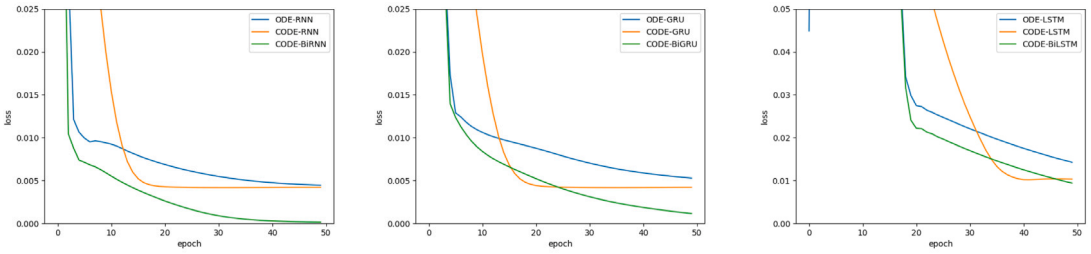


Fig. 15. Training loss through the epochs for future extrapolation task for the DC dataset, for 15/15. (a) ODE-RNN, CODE-RNN, CODE-BiRNN; (b) ODE-GRU, CODE-GRU, CODE-BiGRU; (c) ODE-LSTM, CODE-LSTM, CODE-BiLSTM.

Table 15
Numerical results on the DC dataset at the backward extrapolation task, for ODE-LSTM, CODE-LSTM and CODE-BiLSTM.

Days seen/predict	$[t_f, t_0]$	ODE-LSTM $MSE_{avg} \pm std_{avg}$	CODE-LSTM $MSE_{avg} \pm std_{avg}$	CODE-BiLSTM $MSE_{avg} \pm std_{avg}$
7/7	[1462, 0]	8.18e−2 ± 1.64e−2	4.81e−1 ± 5.08e−2	**2.55e−2 ± 7.6e−3**
15/15	[1462, 0]	5.97e−2 ± 6.26e−3	13.96e−1 ± 1.02e−2	**9.45e−3 ± 1.00e−4**

Table 16
Numerical results on the HM dataset at the missing data imputation task, for ODE-RNN, CODE-RNN and CODE-BiRNN.

Seen/predict features	$[t_0, t_f]$	ODE-RNN $MSE_{avg} \pm std_{avg}$	CODE-RNN $MSE_{avg} \pm std_{avg}$	CODE-BiRNN $MSE_{avg} \pm std_{avg}$
1/1	[0, 208]	9.00e−3 ± 9.83e−5	8.90e−3 ± 7.79e−5	**2.20e−3 ± 3.83e−6**
39/39	[0, 208]	2.56e−2 ± 3.00e−4	2.52e−2 ± 2.00e−4	**1.09e−2 ± 9.80e−5**

Table 17
Numerical results on the HM dataset at the missing data imputation task, for ODE-GRU, CODE-GRU and CODE-BiGRU.

Seen/predict features	$[t_0, t_f]$	ODE-GRU $MSE_{avg} \pm std_{avg}$	CODE-GRU $MSE_{avg} \pm std_{avg}$	CODE-BiGRU $MSE_{avg} \pm std_{avg}$
1/1	[0, 208]	8.20e−3 ± 9.55e−6	8.70e−3 ± 5.95e−5	**2.3e−3 ± 9.68e−6**
39/39	[0, 208]	2.46e−2 ± 4.00e−4	2.49e−2 ± 1.00e−4	**1.03e−2 ± 2.00e−4**

Table 18
Numerical results on the HM dataset at the missing data imputation task, for ODE-LSTM, CODE-LSTM and CODE-BiLSTM.

Seen/predict features	$[t_0, t_f]$	ODE-LSTM $MSE_{avg} \pm std_{avg}$	CODE-LSTM $MSE_{avg} \pm std_{avg}$	CODE-BiLSTM $MSE_{avg} \pm std_{avg}$
1/1	[0, 208]	1.13e−2 ± 3.47e−5	1.12e−2 ± 6.35e−5	**3.20e−3 ± 6.48e−5**
39/39	[0, 208]	2.38e−2 ± 2.00e−4	2.37e−2 ± 1.00e−4	**1.04e−2 ± 2.00e−4**

architectures.

Missing data imputation We performed the missing data imputation task to predict data points with 1 feature (hydropower inflow) and 39 features (39 average river discharges), denoted by 1/1 and 39/39 respectively. The numerical results are shown in Table 16 for architectures ODE-RNN, CODE-RNN and CODE-BiRNN, Table 17 for architectures ODE-GRU, CODE-GRU and CODE-BiGRU, and Table 18 for architectures ODE-LSTM, CODE-LSTM and CODE-BiLSTM. The evolution of the loss during training for the predictions with 1/1 and 39/39 features are depicted in Figs. 16 and 17, respectively.

As shown in Tables 16–18, the CODE-BiRNN, CODE-BiGRU, and CODE-BiLSTM models stand out by delivering the best performance compared to the other approaches. Among these three variants, their performances are similar. When comparing ODE-RNN, ODE-GRU, and ODE-LSTM with CODE-RNN, CODE-GRU, and CODE-LSTM, the performance differences are minimal. However, as the number of features increases, all architectures show an increase in MSE_{avg} values.

Figs. 16 and 17 show that CODE-RNN, CODE-GRU, and CODE-LSTM achieve the lowest MSE_{avg} values among all architectures. CODE-BiRNN, CODE-BiGRU, and CODE-BiLSTM show faster convergence, although ODE-RNN, ODE-GRU, and ODE-LSTM, along with CODE-RNN, CODE-GRU, and CODE-LSTM, reach lower MSE_{avg} values. When comparing the training loss values for ODE-RNN, ODE-GRU, ODE-LSTM, and CODE-RNN, CODE-GRU, and CODE-LSTM, we observe similar behaviour, which is consistent with the test results presented in (Tables 7–9). Although CODE-BiRNN, CODE-BiGRU, and CODE-BiLSTM exhibit higher training loss values, their test results outperform the other architectures. This suggests that, despite higher training losses, these models generalise better. In contrast, the ODE-RNN and CODE-RNN models, along with their variants, show poorer generalisation, indicating a tendency toward overfitting.

Future extrapolation We performed the future extrapolation task to predict the hydropower inflow for the next 7 or 15 weeks after receiving the hydropower inflow for past 7 or 15 weeks, denoted by 7/7 and 15/15. The numerical results are shown in Table 19 for architectures ODE-RNN, CODE-RNN and CODE-BiRNN, Table 20 for architectures ODE-GRU, CODE-GRU, CODE-BiGRU and Table 21 for architectures ODE-LSTM, CODE-LSTM and CODE-BiLSTM. The evolution of the losses during training for predictions with 7/7 and 15/15 are depicted in Figs. 18 and 19, respectively.

The results in Tables 19–21 indicate that CODE-BiRNN, CODE-BiGRU, and CODE-BiLSTM consistently outperform the other models, achieving the best predictive performance for both 7/7 and 15/15. Among these variants, CODE-BiRNN yields the best performance. When comparing ODE-RNN, ODE-GRU, and ODE-LSTM with their CODE counterparts (CODE-RNN, CODE-GRU, CODE-LSTM), the architectures

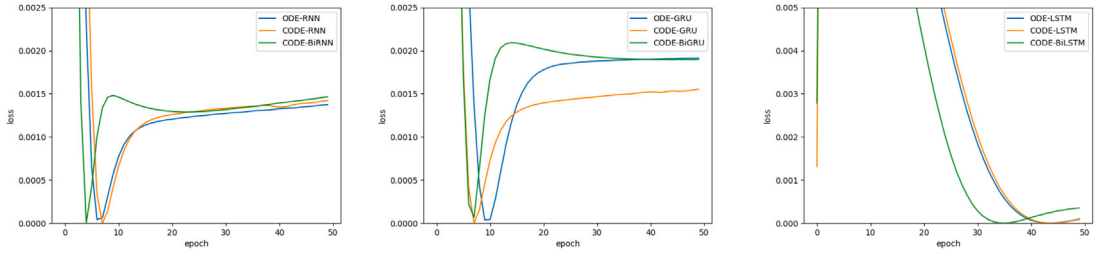


Fig. 16. Training loss through the epochs for the missing data imputation task for the HM dataset, for 1/1. (a) ODE-RNN, CODE-RNN, CODE-BiRNN; (b) ODE-GRU, CODE-GRU, CODE-BiGRU; (c) ODE-LSTM, CODE-LSTM, CODE-BiLSTM.

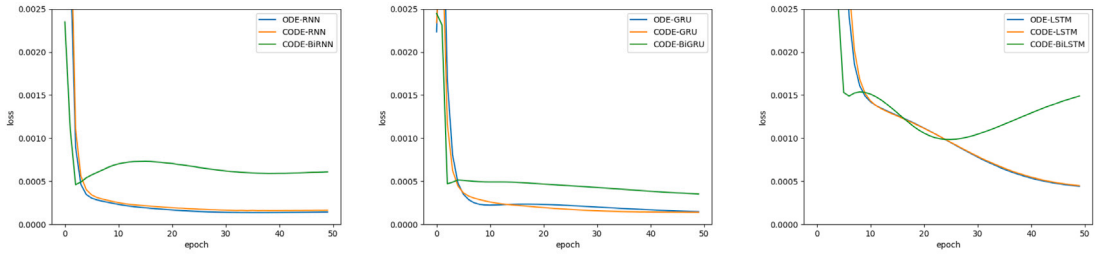


Fig. 17. Training loss through the epochs for the missing data imputation task for the HM dataset, for 39/39. (a) ODE-RNN, CODE-RNN, CODE-BiRNN; (b) ODE-GRU, CODE-GRU, CODE-BiGRU; (c) ODE-LSTM, CODE-LSTM, CODE-BiLSTM.

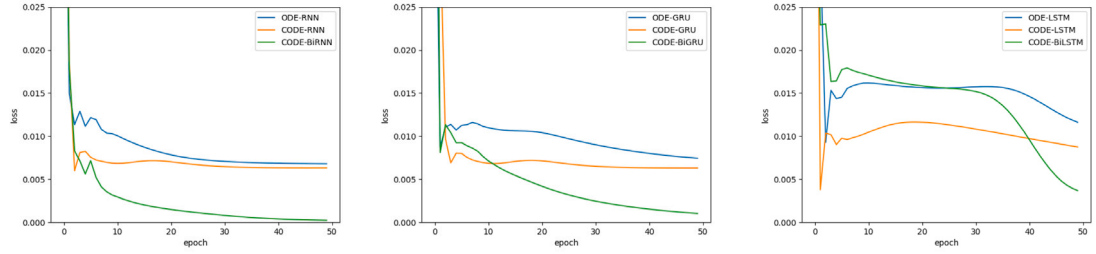


Fig. 18. Training loss through the epochs for future extrapolation task for the DC dataset, for 7/7. (a) ODE-RNN, CODE-RNN, CODE-BiRNN; (b) ODE-GRU, CODE-GRU, CODE-BiGRU; (c) ODE-LSTM, CODE-LSTM, CODE-BiLSTM.

Table 19
Numerical results on the HM dataset at the future extrapolation task, for ODE-RNN, CODE-RNN and CODE-BiRNN.

Weeks seen/predict	$[t_0, t_f]$	ODE-RNN $MSE_{avg} \pm std_{avg}$	CODE-RNN $MSE_{avg} \pm std_{avg}$	CODE-BiRNN $MSE_{avg} \pm std_{avg}$
7/7	[0, 208]	7.33e−2 ± 2.00e−4	8.53e−2 ± 2.00e−4	**1.00e−3 ± 6.00e−5**
15/15	[0, 208]	1.40e−1 ± 5.00e−4	1.05e−1 ± 1.20e−3	**1.61e−2 ± 2.30e−3**

Table 20
Numerical results on the HM dataset at the future extrapolation task, for ODE-GRU, CODE-GRU and CODE-BiGRU.

Weeks seen/predict	$[t_0, t_f]$	ODE-GRU $MSE_{avg} \pm std_{avg}$	CODE-GRU $MSE_{avg} \pm std_{avg}$	CODE-BiGRU $MSE_{avg} \pm std_{avg}$
7/7	[0, 208]	8.35e−2 ± 9.00e−4	8.50e−2 ± 1.00e−4	**4.00e−3 ± 2.00e−4**
15/15	[0, 208]	1.76e−1 ± 1.01e−2	1.10e−1 ± 9.00e−5	**3.87e−2 ± 2.00e−3**

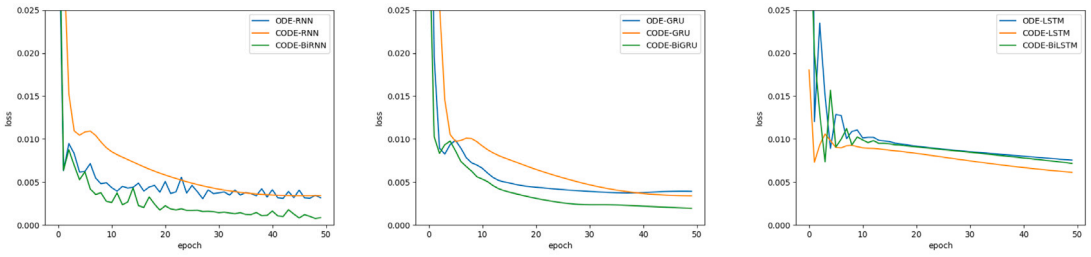
exhibit similar performance for both 7/7 and 15/15.

Figs. 18 and 19 show CODE-BiRNN/-BiGRU/-BiLSTM, in general, exhibit the fastest convergence rates and the lowest loss values. ODE-RNN/-GRU/-LSTM and CODE-RNN/-GRU/-LSTM present achieve similar MSE_{avg} values at 50 epochs although, in general, CODE-RNN/-GRU/-LSTM achieve lower MSE_{avg} values faster.

Backward extrapolation We performed the backward extrapolation task to predict the hydropower inflow for the past 7 or 15 weeks after receiving the hydropower inflow for the next 7 or 15 weeks, denoted by 7/7 and 15/15. The numerical results are shown in Table 22 for architectures ODE-RNN, CODE-RNN and CODE-BiRNN, Table 23 for architectures ODE-GRU, CODE-GRU, CODE-BiGRU and Table 24 for

Table 21
Numerical results on the HM dataset at the future extrapolation task, for ODE-LSTM, CODE-LSTM and CODE-BiLSTM.

Weeks seen/predict	$[t_0, t_f]$	ODE-LSTM $MSE_{avg} \pm std_{avg}$	CODE-LSTM $MSE_{avg} \pm std_{avg}$	CODE-BiLSTM $MSE_{avg} \pm std_{avg}$
7/7	[0, 208]	1.03e−1 ± 8.00e−4	9.63e−2 ± 9.00e−5	**4.18e−2 ± 1.30e−3**
15/15	[0, 208]	1.40e−1 ± 3.00e−4	1.41e−1 ± 3.00e−4	**1.10e−1 ± 7.00e−4**



(a) (b) (c)

Fig. 19. Training loss through the epochs for future extrapolation task for the DC dataset, for 15/15. (a) ODE-RNN, CODE-RNN, CODE-BiRNN; (b) ODE-GRU, CODE-GRU, CODE-BiGRU; (c) ODE-LSTM, CODE-LSTM, CODE-BiLSTM.

Table 22
Numerical results on the HM dataset at the backward extrapolation task, for ODE-RNN, CODE-RNN and CODE-BiRNN.

Weeks seen/predict	$[t_f, t_0]$	ODE-RNN $MSE_{avg} \pm std_{avg}$	CODE-RNN $MSE_{avg} \pm std_{avg}$	CODE-BiRNN $MSE_{avg} \pm std_{avg}$
7/7	[208, 0]	5.24e−2 ± 3.00e−4	4.26e−2 ± 2.00e−5	**2.25e−2 ± 1.00e−4**
15/15	[208, 0]	1.41e−1 ± 0.0019	1.04e−1 ± 1.80e−3	**4.25e−2 ± 1.20e−3**

Table 23
Numerical results on the HM dataset at the backward extrapolation task, for ODE-GRU, CODE-GRU and CODE-BiGRU.

Weeks seen/predict	$[t_f, t_0]$	ODE-GRU $MSE_{avg} \pm std_{avg}$	CODE-GRU $MSE_{avg} \pm std_{avg}$	CODE-BiGRU $MSE_{avg} \pm std_{avg}$
7/7	[208, 0]	5.56e−2 ± 2.00e−3	4.26e−2 ± 7.00e−5	**2.58e−2 ± 2.00e−4**
15/15	[208, 0]	1.86e−1 ± 1.27e−2	1.12e−1 ± 8.00e−5	**4.34e−2 ± 7.00e−4**

Table 24
Numerical results on the HM dataset at the backward extrapolation task, for ODE-LSTM, CODE-LSTM and CODE-BiLSTM.[3]

Weeks seen/predict	$[t_f, t_0]$	ODE-LSTM $MSE_{avg} \pm std_{avg}$	CODE-LSTM $MSE_{avg} \pm std_{avg}$	CODE-LSTM $MSE_{avg} \pm std_{avg}$
7/7	[208, 0]	1.18e−1 ± 5.70e−3	9.67e−2 ± 6.80e−3	**3.36e−2 ± 5.00e−4**
15/15	[208, 0]	1.38e−1 ± 2.00e−4	1.37e−1 ± 2.00e−4	**1.09e−1 ± 8.00e−4**

architectures ODE-LSTM, CODE-LSTM and CODE-BiLSTM.

From Tables 22–24, the performance of the models on the backward extrapolation task are similar to that observed for the forward extrapolation task. Once again, it is worth emphasising that CODE-BiRNN/-BiGRU/-BiLSTM shows the best performance for 7/7 and 15/15. Among these variants, CODE-BiRNN performs the best. As expected, when predicting longer time horizons, all architectures show an increase in the MSE_{avg} values compared to smaller time horizons.

DJIA 30 Stock time-series:

The architectures were tested on an irregularly sampled dataset called DJIA 30 Stock time-series. The dataset consisted of 3019 data points and included 6 daily features: the stock's price at open and close, the highest and lowest price, the number of shares traded, and the stock's ticker name. The data was collected between January 3, 2006, and December 29, 2017, for 29 DJIA companies (szrlee, 2018). However, for this particular study, data from a single company was utilised.

Missing data imputation We performed the missing data imputation task to predict data points with 1 feature (stock's price at close) and 4 features (stock's price at open and close, and the highest and lowest price), denoted by 1/1 and 39/39 respectively. The numerical results are shown in Table 25 for architectures ODE-RNN, CODE-RNN and CODE-BiRNN, Table 26 for architectures ODE-GRU, CODE-GRU and CODE-BiGRU, and Table 27 for architectures ODE-LSTM, CODE-LSTM and CODE-BiLSTM. The evolution of the loss during training for the predictions with 1/1 and 39/39 features are depicted in Figs. 20 and 21, respectively.

From Tables 25–27, it is evident that the performance of CODE-BiRNN, CODE-BiGRU, and CODE-BiLSTM models outperforms the others, with CODE-BiRNN and CODE-BiGRU offering the best results. When comparing ODE-RNN, ODE-GRU, and ODE-LSTM with CODE-RNN, CODE-GRU, and CODE-LSTM, the performances are similar. Additionally, when the number of features is increased, the MSE_{avg} values do not show a significant rise.

Figs. 20 and 21 show that CODE-BiRNN/-BiGRU/-BiLSTM achieve faster convergence and lower MSE_{avg} values, out of all architectures. When comparing ODE-RNN/-GRU/-LSTM and CODE-RNN/-GRU/-LSTM they present similar loss evolution being ODE-GRU able to achieve slightly lower MSE_{avg} values than CODE-GRU.

Future extrapolation We performed the future extrapolation task to predict the stock's price at close for the next 7 or 15 days after receiving the stock's price at close for past 7 or 15 days, denoted by 7/7 and

[3] For doing predictions backward in time using the $ODESolve$ in ODE-LSTM receives the time interval in the reverse order.

Table 25
Numerical results on the DJIA dataset at the missing data imputation task, for ODE-RNN, CODE-RNN and CODE-BiRNN.

Seen/predict features	$[t_0, t_f]$	ODE-RNN $MSE_{avg} \pm std_{avg}$	CODE-RNN $MSE_{avg} \pm std_{avg}$	CODE-BiRNN $MSE_{avg} \pm std_{avg}$
1/1	[0, 3019]	4.00e−4 ± 2.84e−5	4.00e−4 ± 2.91e−5	**9.19e−5 ± 2.28e−6**
4/4	[0, 3019]	3.00e−4 ± 7.80e−7	3.00e−4 ± 1.68e−6	**2.00e−4 ± 1.43e−5**

Table 26
Numerical results on the DJIA dataset at the missing data imputation task, for ODE-GRU, CODE-GRU and CODE-BiGRU.

Seen/predict features	$[t_0, t_f]$	ODE-GRU $MSE_{avg} \pm std_{avg}$	CODE-GRU $MSE_{avg} \pm std_{avg}$	CODE-BiGRU $MSE_{avg} \pm std_{avg}$
1/1	[0, 3019]	4.00e−4 ± 2.18e−5	3.00e−4 ± 2.17e−5	**8.96e−5 ± 4.68e−6**
4/4	[0, 3019]	3.00e−4 ± 1.31e−6	3.00e−4 ± 7.50e−6	**1.00e−4 ± 8.62e−6**

Table 27
Numerical results on the DJIA dataset at the missing data imputation task, for ODE-LSTM, CODE-LSTM and CODE-BiLSTM.

Seen/predict features	$[t_0, t_f]$	ODE-LSTM $MSE_{avg} \pm std_{avg}$	CODE-LSTM $MSE_{avg} \pm std_{avg}$	CODE-BiLSTM $MSE_{avg} \pm std_{avg}$
1/1	[0, 3019]	2.00e−4 ± 1.97e−6	2.00e−4 ± 1.11e−6	2.00e−2 ± 9.58e−6
4/4	[0, 3019]	3.00e−4 ± 3.68e−6	4.00e−4 ± 4.56e−6	**2.00e−4 ± 9.03e−6**

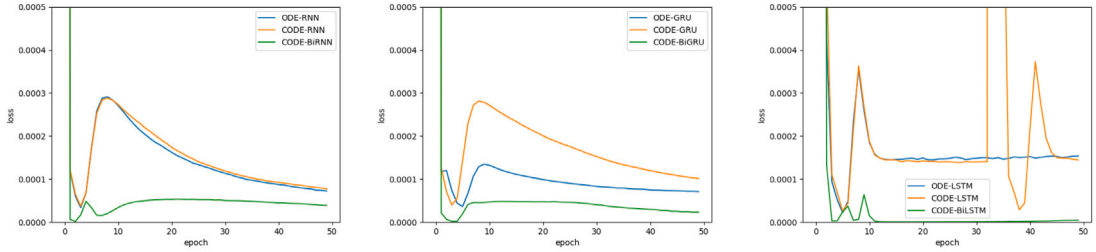


Fig. 20. Training loss through the epochs for the missing data imputation task for the HM dataset, for 1/1. (a) ODE-RNN, CODE-RNN, CODE-BiRNN; (b) ODE-GRU, CODE-GRU, CODE-BiGRU; (c) ODE-LSTM, CODE-LSTM, CODE-BiLSTM.

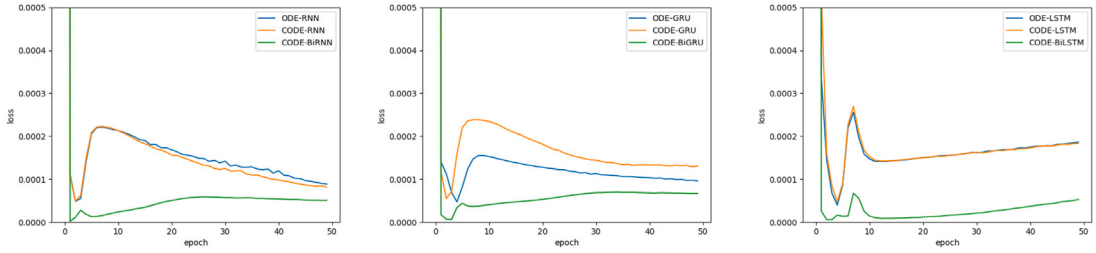


Fig. 21. Training loss through the epochs for the missing data imputation task for the HM dataset, for 4/4. (a) ODE-RNN, CODE-RNN, CODE-BiRNN; (b) ODE-GRU, CODE-GRU, CODE-BiGRU; (c) ODE-LSTM, CODE-LSTM, CODE-BiLSTM.

15/15. The numerical results are shown in Table 28 for architectures ODE-RNN, CODE-RNN and CODE-BiRNN, Table 29 for architectures ODE-GRU, CODE-GRU, CODE-BiGRU and Table 30 for architectures ODE-LSTM, CODE-LSTM and CODE-BiLSTM. The evolution of the losses during training for predictions with 7/7 and 15/15 are depicted in Figs. 22 and 23, respectively.

The results in Tables 28–30 show that, CODE-BiRNN/-BiGRU/-BiLSTM stand out from the others, consistently achieving the best predictive performance, for 7/7 and 15/15. Among these variants, CODE-BiGRU offers the best performance for 7/7 while CODE-BiRNN presents the lowest MSE_{avg} for 15/15. CODE-LSTM presents the lowest performance out of all networks, for 7/7 and 15/15 and, in general ODE-RNN/-GRU/-LSTM have lower or similar MSE_{avg} values when compared to CODE-RNN/-GRU/-LSTM. This was expected since the architecture of these proposed networks do not consider the information given by the input to update the hidden state passed onto the next iteration.

From Figs. 22 and 23 CODE-BiRNN/-BiGRU/-BiLSTM exhibit the lowest loss values. In general, reveal unstable loss evolution for all architectures, for 7/7, being CODE-BiRNN/-BiGRU/BiLSTM the most stable and providing the lowest MSE_{avg} values throughout the epochs. Furthermore, ODE-RNN/-GRU/-LSTM shows lower MSE_{avg} values than CODE-RNN/-GRU/-LSTM, corroborating the test results in Tables 28–30. For 15/15, CODE-RNN/-GRU/-LSTM show fastest convergence. The loss value of all models increases around epoch 40, suggesting the implementation of an early stopping criterion could be

Table 28
Numerical results on the DJIA dataset at the future extrapolation task, for ODE-RNN, CODE-RNN and CODE-BiRNN.

Days seen/predict	$[t_0, t_f]$	ODE-RNN $MSE_{avg} \pm std_{avg}$	CODE-RNN $MSE_{avg} \pm std_{avg}$	CODE-BiRNN $MSE_{avg} \pm std_{avg}$
7/7	[0, 3019]	1.60e−3 ± 8.00e−4	3.12e−2 ± 1.00e−3	**1.10e−3 ± 2.00e−4**
15/15	[0, 3019]	1.60e−2 ± 6.00e−4	1.41e−2 ± 8.10e−3	**8.00e−4 ± 5.00e−4**

Table 29
Numerical results on the DJIA dataset at the future extrapolation task, for ODE-GRU, CODE-GRU and CODE-BiGRU.

Days seen/predict	$[t_0, t_f]$	ODE-GRU $MSE_{avg} \pm std_{avg}$	CODE-GRU $MSE_{avg} \pm std_{avg}$	CODE-BiGRU $MSE_{avg} \pm std_{avg}$
7/7	[0, 3019]	1.70e−2 ± 1.30e−3	3.10e−2 ± 3.00e−4	**3.00e−4 ± 5.00e−5**
15/15	[0, 3019]	3.00e−3 ± 1.40e−3	1.81e−2 ± 1.93e−2	**1.80e−3 ± 1.00e−3**

Table 30
Numerical results on the DJIA dataset at the future extrapolation task, for ODE-LSTM, CODE-LSTM and CODE-BiLSTM.

Days seen/predict	$[t_0, t_f]$	ODE-LSTM $MSE_{avg} \pm std_{avg}$	CODE-LSTM $MSE_{avg} \pm std_{avg}$	CODE-BiLSTM $MSE_{avg} \pm std_{avg}$
7/7	[0, 3019]	4.96e−2 ± 8.00e−4	3.19e−1 ± 1.34e−2	**1.13e−2 ± 3.00e−4**
15/15	[0, 3019]	1.69e−2 ± 4.50e−3	3.06e−1 ± 1.21e−1	**3.20e−3 ± 5.00e−4**

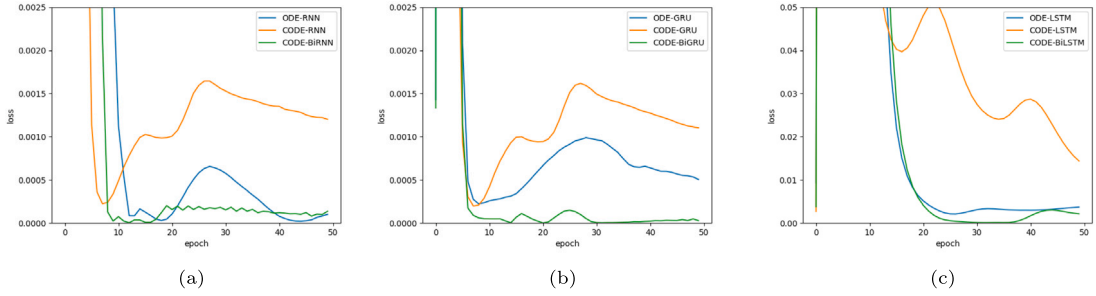


Fig. 22. Training loss through the epochs for future extrapolation task for the DC dataset, for 7/7. (a) ODE-RNN, CODE-RNN, CODE-BiRNN; (b) ODE-GRU, CODE-GRU, CODE-BiGRU; (c) ODE-LSTM, CODE-LSTM, CODE-BiLSTM.

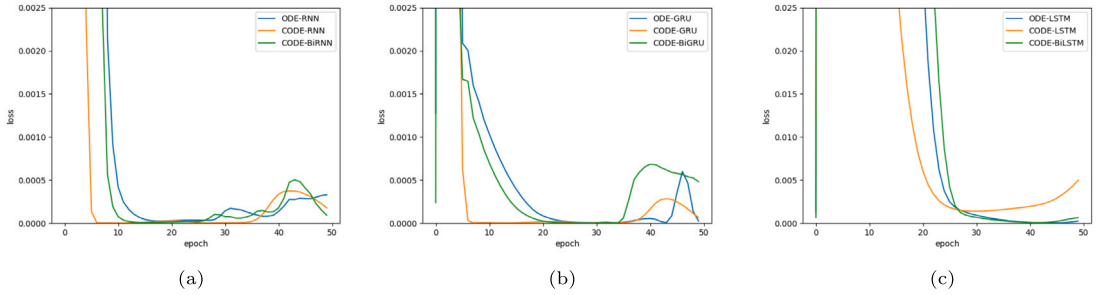


Fig. 23. Training loss through the epochs for future extrapolation task for the DC dataset, for 15/15. (a) ODE-RNN, CODE-RNN, CODE-BiRNN; (b) ODE-GRU, CODE-GRU, CODE-BiGRU; (c) ODE-LSTM, CODE-LSTM, CODE-BiLSTM.

Table 31
Numerical results on the DJIA dataset at the backward extrapolation task, for ODE-RNN, CODE-RNN and CODE-BiRNN.

Days seen/predict	$[t_f, t_0]$	ODE-RNN $MSE_{avg} \pm std_{avg}$	CODE-RNN $MSE_{avg} \pm std_{avg}$	CODE-BiRNN $MSE_{avg} \pm std_{avg}$
7/7	[3019, 0]	1.80e−3 ± 9.00e−4	3.19e−2 ± 1.00e−3	**1.40e−3 ± 2.00e−4**
15/15	[3019, 0]	1.40e−2 ± 6.00e−4	1.36e−2 ± 7.00e−3	**1.00e−3 ± 5.00e−4**

beneficial.

Backward extrapolation We performed the backward extrapolation task to predict the stock's price at close for the past 7 or 15 days after receiving the stock's price at close for the next 7 or 15 days, denoted by 7/7 and 15/15. The numerical results are shown in Table 31 for architectures ODE-RNN, CODE-RNN and CODE-BiRNN, Table 32 for architectures ODE-GRU, CODE-GRU, CODE-BiGRU and Table 33 for architectures ODE-LSTM, CODE-LSTM and CODE-BiLSTM.

From Tables 31–33, the performance of the models on the backward extrapolation task are similar to that observed for the forward extrapolation task, with CODE-BiRNN/-BiGRU/-BiLSTM demonstrating the highest performance for 7/7 and 15/15. Among these variants, CODE-BiGRU performs best. As expected, when predicting longer time horizons, all architectures show an increase in the MSE_{avg} values compared to smaller time horizons. Furthermore, in general, CODE-RNN/-GRU/-LSTM perform less effectively than ODE-RNN/-GRU/-LSTM.

Table 32
Numerical results on the DJIA dataset at the backward extrapolation task, for ODE-GRU, CODE-GRU and CODE-BiGRU.

Days seen/predict	$[t_f, t_0]$	ODE-GRU $MSE_{avg} \pm std_{avg}$	CODE-GRU $MSE_{avg} \pm std_{avg}$	CODE-BiGRU $MSE_{avg} \pm std_{avg}$
7/7	[3019, 0]	1.76e−2 ± 1.40e−3	3.17e−2 ± 3.00e−4	**6.00e−4 ± 7.00e−5**
15/15	[3019, 0]	2.20e−3 ± 9.00e−4	1.78e−2 ± 1.78e−2	**2.10e−3 ± 1.00e−3**

Table 33
Numerical results on the DJIA dataset at the backward extrapolation task, for ODE-LSTM, CODE-LSTM and CODE-BiLSTM.[4]

Days seen/predict	$[t_f, t_0]$	ODE-LSTM $MSE_{avg} \pm std_{avg}$	CODE-LSTM $MSE_{avg} \pm std_{avg}$	CODE-LSTM $MSE_{avg} \pm std_{avg}$
7/7	[3019, 0]	1.01e−1 ± 2.10e−3	1.69e0 ± 1.06e−1	**1.29e−2 ± 3.00e−4**
15/15	[3019, 0]	4.24e−2 ± 1.47e−2	7.32e−1 ± 2.24e−1	**3.80e−3 ± 6.00e−4**

4. Conclusion

In this work, we have introduced Neural CODE, a novel NN architecture that adjusts an ODE dynamics to model data both forward and backward in time. To achieve this, Neural CODE minimises a loss function that considers both forward predictions, obtained by solving an initial value problem, and backward predictions, obtained by solving a final value problem, over time. In doing so, Neural CODE effectively captures the dependencies between data points by accounting for how both past and future values influence the current value, thereby improving the interdependencies within the data.

Furthermore, we have proposed two innovative recurrent architectures, CODE-RNN and CODE-BiRNN, which use Neural CODE to model the states between observations. Additionally, GRU and LSTM update cell variants were introduced in these architectures originating CODE-GRU/-LSTM and CODE-BiGRU/-BiLSTM architectures.

Our experimental evaluation clearly demonstrates that Neural CODE surpasses Neural ODE in terms of learning the dynamics of a spiral ODE when making predictions both forward and backward in time. This highlights the significant impact of considering data from both directions on the performance of the trained models. Furthermore, when applied to modelling real-life time-series using the recurrent architectures ODE-RNN/-GRU/-LSTM, CODE-RNN/-GRU/-LSTM and CODE-BiRNN/-BiGRU/-BiLSTM, our results indicate that CODE-BiRNN/-BiGRU/-BiLSTM stands out as the architecture with the highest accuracy, faster convergence, and the ability to reach the lowest training loss values. These experimental findings provide strong evidence that Neural CODE-based architectures effectively double the available information for fitting the ODE, enhancing the optimisation algorithm's capabilities and resulting in faster convergence and superior performance. Learning the context from both past and future observations enables the capture of patterns and trends, leading to improved generalisation and increased resilience against possible existing data errors.

Overall, leveraging both the forward and backward solutions of an ODE proves to be a powerful approach for data fitting, offering superior performance, generalisation, and robustness compared to solely relying on forward solutions. However, it is crucial to consider the trade-off between these benefits and the associated computational resources and complexity.

While this work highlights the advantages of Neural CODE, several directions for future research remain:

- Many real-world systems are not perfectly time-reversible due to factors such as energy dissipation, irreversible processes, or information loss. Further investigation is needed to evaluate the applicability of Neural CODE and recurrent Neural CODE architectures to these systems;
- CODE-RNN, CODE-GRU, CODE-LSTM, CODE-BiRNN, CODE-BiGRU, and CODE-BiLSTM architectures employ merging operations to integrate information from both directions. The effect of different merging operations on the performance of recurrent Neural CODE architectures remains an open question;
- With the rising popularity and performance improvements offered by attention mechanisms, it is worth exploring the integration of Neural CODE with attention mechanisms to evaluate its potential for enhancing long-term dependency modelling in time-series forecasting;
- The architectures proposed in this work can be applied to problems in various domains being important to study and explore their applicability and performance gains;
- A comprehensive study on the effect of the hyperparameter choice in their performance should be carried out;
- With the growing emphasis on the interpretability of NNs, it is essential to use/develop methods for analysing internal dynamic changes, assessing feature importance, and understanding the decision-making processes of the proposed architectures. Such investigations would offer deeper insights into their mechanisms and could also drive further enhancements in the architectures and broaden their range of applications.

[3] For doing predictions backward in time using the $ODESolve$ in ODE-LSTM receives the time interval in the reverse order.

CRediT authorship contribution statement

C. Coelho: Conceptualization, Methodology, Software, Data curation, Writing – original draft, Visualization, Investigation, Validation, Formal analysis, Writing – review & editing. **M. Fernanda P. Costa:** Validation, Formal analysis, Writing – review & editing, Supervision, Writing – original draft. **L.L. Ferrás:** Validation, Formal analysis, Writing – review & editing, Supervision, Writing – original draft.

Declaration of competing interest

The authors declare the following financial interests/personal relationships which may be considered as potential competing interests: C. Coelho reports financial support was provided by Foundation for Science and Technology. C. Coelho reports financial support was provided by LaCaixa Foundation. M. Fernanda P. Costa reports financial support was provided by Foundation for Science and Technology. L.L. Ferras reports financial support was provided by Foundation for Science and Technology. M. Fernanda P. Costa reports financial support was provided by LaCaixa Foundation. L.L. Ferras reports financial support was provided by LaCaixa Foundation. If there are other authors, they declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgements

The authors acknowledge the funding by Fundação para a Ciência e Tecnologia (Portuguese Foundation for Science and Technology) - FCT

through CMAT projects UIDB/00013/2020 and UIDP/00013/2020, the funding by FCT and Google Cloud partnership through projects CPCA-IAC/AV/589164/2023 and CPCA-IAC/AF/589140/2023, and the support of the High Performance Computing Center at the University of Évora funded by FCT I.P. under the project "OptXAI: Constrained Optimisation in NNs for Explainable, Ethical and Greener AI", reference 2024.00191.CPCA.A1, platform Vision.

C. Coelho would like to thank FCT the funding through the scholarship with reference 2021.05201.BD.

This work is also financially supported by national funds through the FCT/MCTES (PIDDAC), under the project 2022.06672.PTDC - iMAD - Improving the Modelling of Anomalous Diffusion and Viscoelasticity: solutions to industrial problems, with DOI 10.54499/2022.06672.PTDC (https://doi.org/10.54499/2022.06672.PTDC); and by Fundação "la Caixa"|BPI and FCT through project PL24-00057: "Inteligência Artificial na Otimização da Rega para Olivais Resilientes às Alterações Climáticas".

Additional funding was provided by FCT and the Centro de Estudos de Fenómenos de Transporte through the projects UIDB/00532/2020 and UIDP/00532/2020. Further support was granted by FCT and ALiCE through the project LA/P/0045/2020.

Appendix. Background

This appendix aims to provide some background information on the developments discussed in Section 2. We will introduce a brief description of Neural ODE, RNN, ODE-RNN, and BiRNN.

A.1. Neural ODE

In traditional NNs, a sequential arrangement of hidden layers $\boldsymbol{h}_i$ is employed to transform an input vector $\boldsymbol{x}_i \in \mathbb{R}^d$ into the desired predicted output $\hat{\boldsymbol{y}}_i \in \mathbb{R}^{d^*}$ (Chen et al., 2018). This transformation is achieved by propagating the output of each layer to the next one, following Eq. (A.1) until reaching the output layer and obtaining the desired predicted output as indicated in Eq. (A.2),

$$\boldsymbol{h}_i = \sigma(\boldsymbol{W}^{[i]}\boldsymbol{h}_{i-1} + \boldsymbol{b}^{[i]}), \tag{A.1}$$

$$\hat{\boldsymbol{y}}_i = \sigma(\boldsymbol{W}^{[out]}\boldsymbol{h}_i + \boldsymbol{b}^{[out]}). \tag{A.2}$$

Here, $\boldsymbol{h}_0$ is the input layer, σ an activation function, $\boldsymbol{W}^{[i]} \in \mathbb{R}^{n\times d}$ the weights matrix of layer i with n neurons, $\boldsymbol{b}^{[i]} \in \mathbb{R}^n$ the biases of layer i, $\boldsymbol{W}^{[out]} \in \mathbb{R}^{d^*\times n}$ the weights matrix of the output layer and $\boldsymbol{b}^{[out]} \in \mathbb{R}^{d^*}$ the biases of the output layer.

In Chen et al. (2018), the authors introduced Neural ODE, a NN architecture that models hidden states using a continuous-time function. Unlike traditional NNs that rely on discrete layers, Neural ODE employs an infinite number of hidden layers, approaching a continuum. The output of these layers is defined as the solution of an IVP,

$$\frac{d\boldsymbol{h}(t)}{dt} = \boldsymbol{f}_{\boldsymbol{\theta}}(\boldsymbol{h}(t), t) \text{ with } \boldsymbol{h}(t_0) = \boldsymbol{h}_0 \tag{A.3}$$

where $\boldsymbol{f}_{\boldsymbol{\theta}}$ is given by a NN, parametrised by a set of weights and biases, $\boldsymbol{\theta}$, t is the time step of the solution and $(\boldsymbol{h}_0, t_0)$ is the initial condition of the IVP.

One significant advantage of training a Neural ODE is that it produces a continuous-time function. Consequently, predictions can be made at any desired time point by discretising the function using a numerical ODE solver:

$$\boldsymbol{h}(t) = ODESolve(\boldsymbol{f}_{\boldsymbol{\theta}}, \boldsymbol{h}_0, [t_0, t_f]),$$

with $\boldsymbol{h}(t)$ the solutions computed by solving the IVP (A.3) in the time interval $[t_0, t_f]$.

Taking $\boldsymbol{f}_{\boldsymbol{\theta}}(\boldsymbol{h}(t), t)$ to be a NN, which builds an ODE dynamics with learnable parameters $\boldsymbol{\theta}$ and the input vector $(\boldsymbol{x}, t) \in \mathbb{R}^{d+1}$ that corresponds to the first time step t_0 of the time-series, the input layer $\boldsymbol{h}_0$ is given by (A.4):

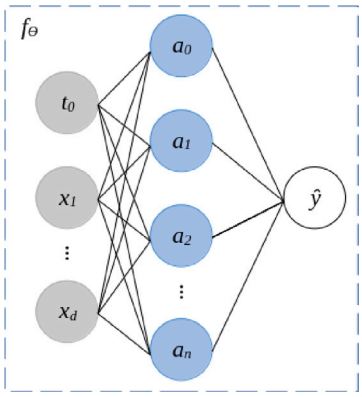


Fig. A.24. The adjusted ODE dynamics, $\boldsymbol{f}_{\boldsymbol{\theta}}$, is given by a NN with an input layer with the dimension $d+1$ to accommodate the input vector, $\boldsymbol{x} \in \mathbb{R}^d$ with d features, and its time step, $t \in \mathbb{R}$.

$$\boldsymbol{h}(t_0) = \boldsymbol{h}_0 = \sigma\left(\boldsymbol{W}^{[in]}(\boldsymbol{x}, t) + \boldsymbol{b}^{[in]}\right), \tag{A.4}$$

where $\boldsymbol{W}^{[in]} \in \mathbb{R}^{n\times d+1}$ and $\boldsymbol{b}^{[in]} \in \mathbb{R}^n$ are the weight matrix and the bias vector of the input layer, respectively.

Consider a single hidden layer, $\boldsymbol{h} \in \mathbb{R}^n$, with n neurons, a_n, a batch size of 1 given by the Neural ODE, $\boldsymbol{f}_{\boldsymbol{\theta}}$ (see Fig. A.24).

The dynamics built by such a NN has the form:

$$\boldsymbol{f}_{\boldsymbol{\theta}}(\boldsymbol{h}(t), t) = \sigma\left(\boldsymbol{W}^{[out]}(\sigma(\boldsymbol{W}^{[1]}\boldsymbol{h}(t_0) + \boldsymbol{b}^{[1]})) + \boldsymbol{b}^{[out]}\right) \tag{A.5}$$

where σ represents an activation function, $\boldsymbol{h}(t_0) \in \mathbb{R}^n$ is the output of the input layer, (A.4), $\boldsymbol{W}^{[1]} \in \mathbb{R}^{n\times n}$, $\boldsymbol{W}^{[out]} \in \mathbb{R}^{d^*\times n}$ are the matrix of the weights of the hidden and output layers and $\boldsymbol{b}^{[1]} \in \mathbb{R}^n$, $\boldsymbol{b}^{[out]} \in \mathbb{R}^{d^*}$ are the bias of the hidden and output layers, respectively.

Upon examining (A.4) and (A.5), it becomes apparent that a NN is a composition of linear transformations with the application of various activation functions σ. These activation functions can either be linear (such as identity or binary step functions) or non-linear (including sigmoid, hyperbolic tangent (tanh), rectified linear unit (ReLU), softmax, etc.).

When all activation functions in the NN that constructs $\boldsymbol{f}_{\boldsymbol{\theta}}$ are linear, it results in a linear ODE. However, linear activation functions are not commonly used as they cannot effectively capture complex relationships within the data. Consequently, they limit the representational power of the NN and hinder its performance in the given task (Goodfellow et al., 2016). On the other hand, by incorporating one or more non-linear activation functions in the dynamics builder NN, it becomes a non-linear ODE.

In addition to the NN that models the function dynamics $\boldsymbol{f}_{\boldsymbol{\theta}}$ and by optimising its parameters $\boldsymbol{\theta}$, a Neural ODE also incorporates an ODE solver. The ultimate outcome of training a Neural ODE is the learned function $\boldsymbol{f}_{\boldsymbol{\theta}}$. To make predictions $\hat{\boldsymbol{y}}_i$, $i \in 1, \ldots, N$ at specific time points t_i within the prediction time interval $[t_1, t_N]$, the IVP is solved. The solutions obtained from solving the IVP, $\boldsymbol{h}(t_i) \equiv \boldsymbol{h}_i$, represent the predictions at each respective time step t_i, $i \in 1, \ldots, N$.

Remark. In some cases the solutions $\boldsymbol{h}(t_i)$ obtained from the Neural ODE do not directly correspond to the desired predictions $\hat{\boldsymbol{y}}_i$. In such cases, an additional NN can be employed to process the intermediate solutions $\boldsymbol{h}(t_i)$ and produce the final predictions $\hat{\boldsymbol{y}}_i$.

A.2. RNNs

RNNs (Elman, 1990) are a type of artificial NN that are specifically designed to process sequential data. Unlike feed-forward NNs, which

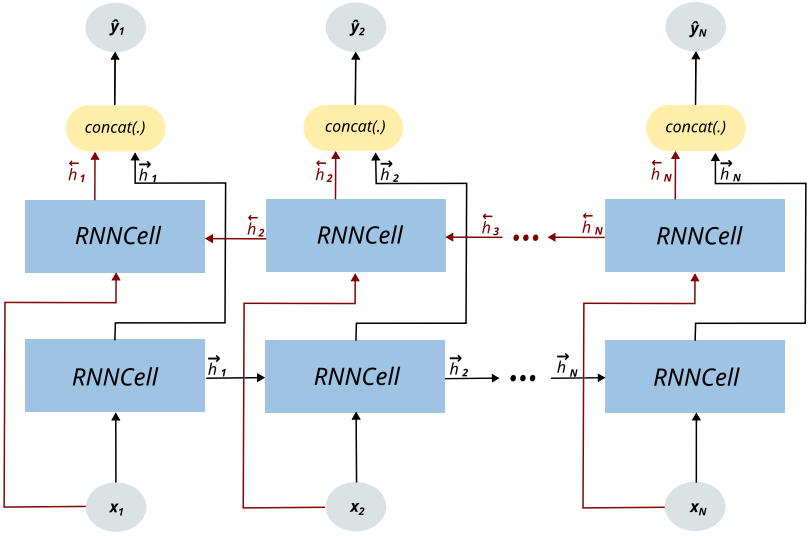


Fig. A.25. Scheme of a BiRNN unfolded through time.

process inputs in a single pass and do not have memory of previous inputs, RNNs have a form of memory that allows them to retain and utilise information from previous steps or time points in the sequence.

The key characteristic of an RNN is its recurrent connection, which forms a loop that allows information to be passed from one step to the next (from one hidden state at time step t_{i-1}, $\boldsymbol{h}_{i-1}$, into the next one, $\boldsymbol{h}_i$). This loop enables the network to maintain an internal state or hidden state that captures the context and dependencies of the previous steps. The hidden state serves as a memory that encodes information about the sequence up to the current step, allowing the network to make decisions or predictions based on the entire previous observations rather than just the current input. Mathematically, a RNN cell which forms the layers of an RNN, can be defined for $i = 1, \dots, N$:

$$\boldsymbol{h}_i = \sigma(\boldsymbol{W}^{[\text{feedback}]}\boldsymbol{h}_{i-1} + \boldsymbol{W}^{[\text{in}]}\boldsymbol{x}_i + \boldsymbol{b}^{[i]})$$

where $\boldsymbol{h}_i \in \mathbb{R}^n$ is the initial hidden state vector $\boldsymbol{h}_0$ is initialised at the start of the recurrent computation, $\boldsymbol{W}^{[\text{feedback}]} \in \mathbb{R}^{n\times n}$ is the recurrent matrix that contains the weights that link the two hidden state vectors at time step t_{i-1} and t_i, $\boldsymbol{W}^{[\text{in}]} \in \mathbb{R}^{n\times d}$ is the matrix of the weights between the input and all hidden state vectors, $\boldsymbol{x}_i \in \mathbb{R}^d$, $\boldsymbol{b}^{[i]} \in \mathbb{R}^n$ is the bias vector of the current hidden state and σ is an arbitrary activation function.

At each step t_i in the sequence, an RNN takes an input, processes it along with the current hidden state, and produces an output and an updated hidden state. The input can be any form of sequential data, such as a time-series, a sentence, or a sequence of images. The output can be a prediction, a classification, or another sequence of values.

The update of an RNN cell at a time step t_i is usually represented concisely as:

$$\boldsymbol{h}_i = RNNCell(\boldsymbol{h}_{i-1}, \boldsymbol{x}_i). \tag{A.6}$$

The predicted output at time step t_i, $\hat{\boldsymbol{y}}_i$, is computed by applying an output layer to the hidden state $\boldsymbol{h}_i$ (Elman, 1990):

$$\hat{\boldsymbol{y}}_i = \sigma(\boldsymbol{W}^{[\text{out}]}\boldsymbol{h}_i + \boldsymbol{b}^{[\text{out}]})$$

where $\boldsymbol{W}^{[\text{out}]} \in \mathbb{R}^{d^*\times n}$ is the matrix of the weights between the hidden state and the output vectors, $\boldsymbol{b}^{[\text{out}]} \in \mathbb{R}^{d^*}$ is the bias vector of the output layer and $\hat{\boldsymbol{y}}_i \in \mathbb{R}^{d^*}$ is the prediction at time step i.

A.3. ODE-RNN

RNNs adjust a discrete-time dynamics with states defined at each time step, denoted as t_i. However, the states between data observations, occurring in the interval $[t_i, t_{i+1}]$, remain undefined. Moreover, RNNs face challenges when handling irregularly sampled data due to the inability to perceive the sampling intervals between observations. Consequently, preprocessing of such data becomes essential to achieve regular sampling intervals, although this process introduces potential errors (Rubanova et al., 2019).

To address these limitations, Rubanova et al. (2019) introduced the ODE-RNN architecture, which defines the states between observations using an ODE.

Instead of using the previous hidden state $\boldsymbol{h}_{i-1}$, as in (A.6), for the RNN update, it employs an intermediate hidden state $\boldsymbol{h}'_i$ instead,

$$\boldsymbol{h}_i = RNNCell(\boldsymbol{h}'_i, \boldsymbol{x}_i). \tag{A.7}$$

The intermediate hidden states are obtained as the solution of an IVP:

$$\frac{d\boldsymbol{h}'(t)}{dt} = \boldsymbol{f}_\theta(\boldsymbol{h}(t), t) \text{ with } \boldsymbol{h}(t_{i-1}) = \boldsymbol{h}_{i-1}$$

Thus, the solution is given by:

$$\boldsymbol{h}'_i = ODESolve(\boldsymbol{f}_\theta, \boldsymbol{h}_{i-1}, [t_{i-1}, t_i])$$

where $\boldsymbol{h}'_i \in \mathbb{R}^n$, $\boldsymbol{f}_\theta$ is the ODE dynamics given by the NN with parameters θ, $\boldsymbol{h}_{i-1}$ is the previous hidden state, and t_{i-1} and t_i are the time steps of the previous and current hidden state, respectively.

ODE-RNNs are well-suited for handling irregularly sampled data due to their ability to leverage the sampling intervals and adapt to time-dependent dynamics. This advantage eliminates the necessity for preprocessing steps, which are typically required when using traditional RNNs.

A.4. BiRNNs

RNNs have been widely used, within encoder–decoder architectures, in various applications such as machine translation (Cho et al., 2014), NLP (Sutskever et al., 2014), and time-series prediction (Hewamalage et al., 2021).

As previously mentioned, RNNs enable the prediction of the output at a specific time step t_i, $\hat{\boldsymbol{y}}_i$, by using past information stored in the hidden state from the previous time step, $\boldsymbol{h}_{i-1}$. This captures the dependency between the current value and the preceding one (Elman,

1990). However, certain tasks, such as NLP and machine translation, require predictions that depend not only on the past but also on the future. Therefore, incorporating future information can lead to improved predictive performance.

To address this, Schuster and Paliwal (1997) introduced BiRNNs, an architecture composed of two stacked RNNs. The first RNN receives the input sequence in its original order, from $\boldsymbol{x}_1$ to $\boldsymbol{x}_N$, while the second RNN receives the same sequence but in reverse order, from $\boldsymbol{x}_N$ to $\boldsymbol{x}_1$. Consequently, one row of the unfolded RNNs captures the dependence between the previous value and the current one through forward hidden states, denoted as $\overrightarrow{\boldsymbol{h}}_i$. Simultaneously, the other row captures the dependence between the next value and the current one through backward hidden states, denoted as $\overleftarrow{\boldsymbol{h}}_i$. This is represented in the two following equations for $i = 1, \ldots, N$:

$$\overrightarrow{\boldsymbol{h}}_i = \sigma(\overrightarrow{\boldsymbol{W}}^{[\text{feedback}]}\overrightarrow{\boldsymbol{h}}_{i-1} + \boldsymbol{W}^{[f]}\boldsymbol{x}_i + \boldsymbol{b}^{[f]}),$$

$$\overleftarrow{\boldsymbol{h}}_i = \sigma(\overleftarrow{\boldsymbol{W}}^{[\text{feedback}]}\overleftarrow{\boldsymbol{h}}_{i-1} + \boldsymbol{W}^{[b]}\boldsymbol{x}_i + \boldsymbol{b}^{[b]}),$$

where $\boldsymbol{W}^{[f]} \in \mathbb{R}^{n\times d}$ is the weight matrix between the input and forward hidden state vectors, $\boldsymbol{b}^{[f]} \in \mathbb{R}^n$ is the bias vector of the forward hidden states, $\boldsymbol{W}^{[b]} \in \mathbb{R}^{n\times d}$ is the weight matrix between the input and backward hidden states and $\boldsymbol{b}^{[b]} \in \mathbb{R}^n$ the bias vector of the backward hidden states.

To get the predicted output at a given time step, $\hat{\boldsymbol{y}}_i$, the full hidden state, $\boldsymbol{h}_i$, is computed by performing an aggregation/merging operation (concatenate, sum, mean, etc.) of the forward, $\overrightarrow{\boldsymbol{h}}_i$, and backward, $\overleftarrow{\boldsymbol{h}}_i$, hidden states of the same time step i (Schuster & Paliwal, 1997) as follows, for $i = 1, \ldots, N$,

$$\hat{\boldsymbol{y}}_i = \sigma(\boldsymbol{W}^{[\text{out}]}\boldsymbol{h}_i + \boldsymbol{b}^{[\text{out}]}) \text{ with } \boldsymbol{h}_i = \overrightarrow{\boldsymbol{h}}_i \oplus \overleftarrow{\boldsymbol{h}}_i,$$

where $\oplus$ denotes the aggregation/merging operation used. In our study and experiments, we chose concatenation. Fig. A.25 depicts the architecture of a BiRNN.

If multiple layers of BiRNNs are used, the complete hidden state, $\boldsymbol{h}_i$, is the input for the subsequent BiRNN (Schuster & Paliwal, 1997).

Data availability

Data will be made available on request.

References

Bureau, U. C. (2020). Population time series data. https://www.kaggle.com/datasets/census/population-time-series-data.

Chen, R. T. Q. (2018). Torchdiffeq. https://github.com/rtqichen/torchdiffeq.

Chen, R. T. Q., Rubanova, Y., Bettencourt, J., & Duvenaud, D. K. (2018). Neural ordinary differential equations. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, & R. Garnett (Eds.), *vol. 31*, *Advances in neural information processing systems*. Curran Associates, Inc., URL: https://proceedings.neurips.cc/paper_files/paper/2018/file/69386f6bb1dfed68692a24c8686939b9-Paper.pdf.

Cho, K., van Merriënboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., & Bengio, Y. (2014). Learning phrase representations using RNN encoder–decoder for statistical machine translation. In A. Moschitti, B. Pang, & W. Daelemans (Eds.), *Proceedings of the 2014 conference on empirical methods in natural language processing* EMNLP, (pp. 1724–1734). Doha, Qatar: Association for Computational Linguistics, http://dx.doi.org/10.3115/v1/D14-1179, URL: https://aclanthology.org/D14-1179/.

De Felice, M. (2021). Hydropower modelling with hydrological data. https://www.kaggle.com/datasets/matteodefelice/hydropower-modelling-with-hydrological-data.

Elman, J. L. (1990). Finding structure in time. *Cognitive Science*, *14*(2), 179–211. http://dx.doi.org/10.1207/s15516709cog1402_1.

Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep Learning*. MIT Press, http://www.deeplearningBook.org.

Gouk, H., Frank, E., Pfahringer, B., & Cree, M. J. (2021). Regularisation of neural networks by enforcing lipschitz continuity. *Machine Learning*, *110*, 393–416.

Hewamalage, H., Bergmeir, C., & Bandara, K. (2021). Recurrent neural networks for time series forecasting: currentstatus and future directions. *International Journal of Forecasting*, [ISSN: 01692070] *37*(1), 388–427. http://dx.doi.org/10.1016/j.ijforecast.2020.06.008.

Lechner, M., & Hasani, R. (2020). Learning long-term dependencies in irregularly-sampled time series. arXiv:2006.04418.

Mahadevaswamy, U., & Swathi, P. (2023). Sentiment analysis using bidirectional LSTM network. *Procedia Computer Science*, [ISSN: 1877-0509] *218*, 45–56. http://dx.doi.org/10.1016/j.procs.2022.12.400, URL: https://www.sciencedirect.com/science/article/pii/S1877050922024930 International Conference on Machine Learning and Data Engineering.

Pontryagin, L. (2018). *Mathematical Theory of Optimal Processes* (First). Routledge, ISBN: 978-0-203-74931-9, http://dx.doi.org/10.1201/9780203749319.

Rubanova, Y., Chen, R. T. Q., & Duvenaud, D. K. (2019). Latent ordinary differential equations for Irregularly-Sampled Time Series. *vol. 32*, In *Advances in neural information processing systems*. Curran Associates, Inc..

Schuster, M., & Paliwal, K. (1997). Bidirectional recurrent neural networks. *IEEE Transactions on Signal Processing*, [ISSN: 1053587X] *45*(11), 2673–2681. http://dx.doi.org/10.1109/78.650093.

Senthilkumar, N., Karpakam, S., Devi, M. G., Balakumaresan, R., & Dhilipkumar, P. (2022). Speech emotion recognition based on bi-directional LSTM architecture and deep belief networks. *Materials Today: Proceedings*, [ISSN: 2214-7853] *57*, 2180–2184. http://dx.doi.org/10.1016/j.matpr.2021.12.246, URL: https://www.sciencedirect.com/science/article/pii/S2214785321079761 International Conference on Innovation and Application in Science and Technology.

Sutskever, I., Vinyals, O., & Le, Q. V. (2014). Sequence to sequence learning with neural networks. In *Proceedings of the 28th international conference on neural information processing systems - volume 2* (pp. 3104–3112). Cambridge, MA, USA: MIT Press.

szrlee (2018). DJIA 30 stock time series. https://www.kaggle.com/datasets/szrlee/stock-time-series-20050101-to-20171231.

V. Rao, S. (2020). Daily Climate time series data. https://www.kaggle.com/datasets/sumanthvrao/daily-climate-time-series-data.

Wang, S., He, J., Wang, C., & Li, X. (2018). The definition and numerical method of final value problem and arbitrary value problem. http://dx.doi.org/10.32604/csse.2018.33.379, URL: http://www.techscience.com/csse/v33n5/39990.

Yang, B., Qin, L., Liu, J., & Liu, X. (2022). IRCNN: An irregular-time-distanced recurrent convolutional neural network for change detection in satellite time series. *IEEE Geoscience and Remote Sensing Letters*, *19*, 1–5. http://dx.doi.org/10.1109/LGRS.2022.3154894.

Yeafi, A. (2022). Air passenger data for time series analysis. https://www.kaggle.com/datasets/ashfakyeafi/air-passenger-data-for-time-series-analysis/data,