Pay Attention to Evolution: Time Series Forecasting With Deep Graph-Evolution Learning

Gabriel Spadon, Shenda Hong, Bruno Brandoli, Stan Matwin, Jose F. Rodrigues-Jr, and Jimeng Sun

Abstract—Time-series forecasting is one of the most active research topics in artificial intelligence. It has the power to bring light to problems in several areas of knowledge, such as epidemiological studies, healthcare inference, and climate change analysis. Applications in real-world time series should consider two factors for achieving reliable predictions: modeling dynamic dependencies among multiple variables and adjusting the model's intrinsic hyperparameters. An open gap in the literature is that statistical and ensemble learning approaches systematically present lower predictive performance than deep learning methods. The existing applications consistently disregard the data sequence aspect entangled with multivariate data represented in more than one time series. Conversely, this work presents a novel neural network architecture for time-series forecasting that combines the power of graph evolution with deep recurrent learning on distinct data distributions, named after Recurrent Graph Evolution Neural Network (REGENN). The idea is to infer multiple multivariate relationships between co-occurring time-series by assuming that the temporal data depends not only on inner variables and intra-temporal relationships (i.e., observations from itself) but also on outer variables and inter-temporal relationships (i.e., observations from other-selves). An extensive set of experiments was conducted comparing REGENN with tens of ensemble methods and classical statistical ones. The results outperformed both statistical and ensemble-learning approaches, showing an improvement of 64.87 percent over the competing algorithms on the SARS-CoV-2 dataset of the renowned John Hopkins University for 188 countries simultaneously. For further validation, we tested our architecture in two other public datasets of different domains, the PhysioNet Computing in Cardiology Challenge 2012 and Brazilian Weather datasets. We also analyzed the *Evolution Weights* arising from the hidden layers of REGENN to describe how the variables of the dataset interact with each other; and, as a result of looking at inter and intra-temporal relationships simultaneously, we concluded that time-series forecasting is majorly improved if paying attention to how multiple multivariate data synchronously evolve.

Index Terms—Time series, graph evolution, representation learning

1 Introduction

Time series refers to the persistent recording of a phenomenon along time, a continuous and intermittent unfolding of chronological events subdivided into past, present, and future. In the last decades, time series analysis has been vital to predict dynamic phenomena on a wide range of applications, such as climate change [1], [2], [3], [4], financial market [5], [6], [7], land-use monitoring [8], [9], [10], anomaly detection [11], [12], [13], energy consumption, and price forecasting [14], [15], [16], apart from epidemiology and healthcare-related studies [17], [18], [19], [20], [21], [22]. On such applications, an effective data-driven decision requires precise forecasting based on time series [23]. A prime example is the SARS-CoV-2, COVID-19, or Coronavirus Pandemic [24], which is known to be highly contagious and cause increased pressure on healthcare systems worldwide [25]. In this case, time-series analysis plays a vital role in planning a safe retake of fundamental activities by preventing economic systems' collapse.

Time series can be regarded as univariate or multivariate describing, respectively, single and multiple variables varying over time [26]. Recent techniques in time series have roots in the use of Artificial Neural Networks [27], which contain a non-linear functioning that enables it to outperform classical algorithms [28]. Such techniques evolved into deep learning models for time-series forecasting, such as Haoyi *et al*. [29] that used an informer component to enhance long-sequence time-series predictions but disregarded the inter-dependencies existing within different multivariate time-series, besides others from the spatiotemporal forecasting field. For example, Seo *et al*. [30] proposed the Graph

- *Gabriel Spadon is with the Institute of Mathematics and Computer Sciences, University of Sao Paulo, São Carlos - SP 13566-590, Brazil, and also with the College of Computing, Georgia Institute of Technology, Atlanta, GA 30332-0365 USA. E-mail: spadon@usp.br, gabriel@spadon.com.br.*
- *Shenda Hong is with the National Institute of Health Data Science and Institute of Medical Technology, Peking University, Beijing 100191, China. E-mail: hongshenda@pku.edu.cn.*
- *Bruno Brandoli is with the Institute for Big Data Analytics, Dalhousie University, Halifax, NS B3H 1W5, Canada. E-mail: brunobrandoli@dal.ca.*
- *Stan Matwin is with the Institute for Big Data Analytics, Dalhousie University, Halifax, NS B3H 1W5, Canada, and also with the Institute of Computer Science, Polish Academy of Sciences, 525-000-94-01 Warsaw, Poland. E-mail: stan@cs.dal.ca.*
- *Jose F. Rodrigues-Jr is with the Institute of Mathematics and Computer Sciences, University of Sao Paulo, São Carlos - SP 13566-590, Brazil, and also with the Faculty of Science, Université Grenoble Alpes, 38400 Saint-Martin-d'Hères, France. E-mail: junio@icmc.usp.br.*
- *Jimeng Sun is with the College of Computing, Georgia Institute of Technology, Atlanta, GA 30332-0365 USA, and also with the Department of Computer Science, University of Illinois Urbana-Champaign, Champaign, IL 61801-2302 USA. E-mail: jsun@cc.gatech.edu.*

Manuscript received 28 Aug. 2020; revised 18 Apr. 2021; accepted 23 Apr. 2021. Date of publication 27 Apr. 2021; date of current version 4 Aug. 2022. (Corresponding author: Jimeng Sun.) Recommended for acceptance by M. Moens. Digital Object Identifier no. 10.1109/TPAMI.2021.3076155

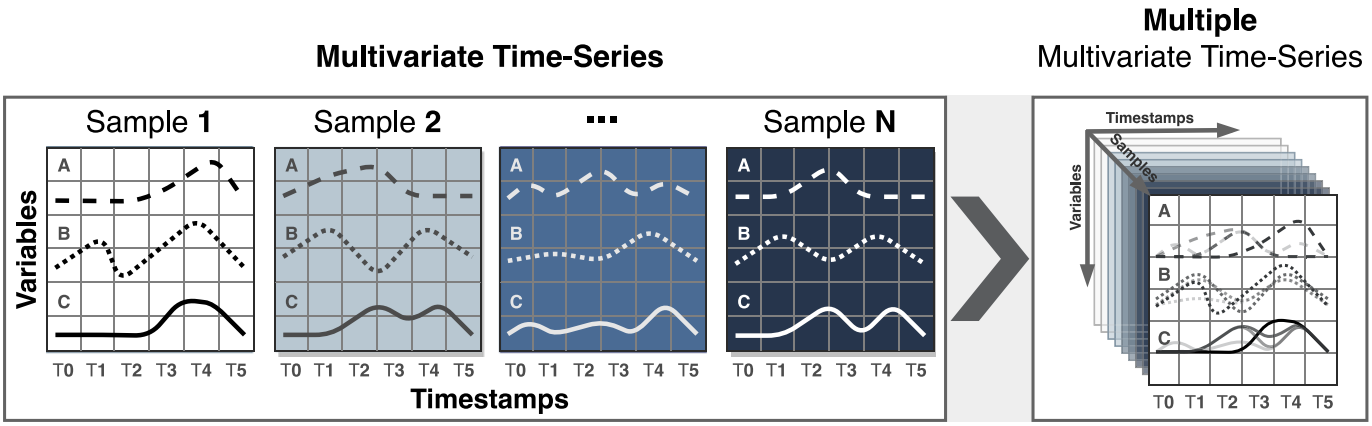


Fig. 1. A multiple multivariate time-series forecasting problem, where each multivariate time-series (i.e., sample) shares the same domain, timestream, and variables. When stacking the time-series together, we assemble a tridimensional tensor with the axes describing samples, timestamps, and variables. The multiple samples have equal variables recorded during the same timestamps, meaning that samples are unique but all observed in the same way. By tackling the problem altogether, we leverage inner and outer variables besides intra- and inter-temporal relationships to improve forecasting.

Convolutional Recurrent Network (GCRN) from graph-structured and time-varying data by combining a Convolutional Neural Network (CNN) [31] that identifies spatial structures and a Recurrent Neural Network (RNN) [32] that learns dynamic patterns; Li *et al.* [33] introduced a spatiotemporal model for traffic forecasting with Diffusion over Convolutional Recurrent Neural Network (DCRNN); and, Zhang *et al.* [34] presented a Gated Attention Networks (GaAN) for forecasting traffic speed using a Graph Gated Recurrent Unit (GGRU) with a CNN controlling the attention head's importance. Further contributions on the spatiotemporal field [35], [36], [37] employ traditional recurrent units, attention mechanisms, and even Graph Convolution Networks (GCN) [38]. However, due to being designed to deal with spatial data, those models often fail to frame temporal dependencies as they aim to achieve state-of-the-art generalization across space and time at once.

Moreover, the LSTNet [39] encodes short-term data into low dimensional vectors by using a CNN for later decoding through an RNN; leverages from a recurrent-skip Gated Recurrent Unit for capturing long-term dependencies within the temporal data; and, incorporates an Autoregressive (AR) component in parallel to the non-linear neural network for preserving the scale of the output. Similarly, the DSANet [40] integrates an AR component with a dual self-attention network [41] with parallel convolutional components, a versatile idea for modeling global and local temporal patterns. More recently, the MLCNN [42] proposed the use of short and long-term prediction strategies for modeling temporal behavior through a multi-layer CNN together with Long Short-Term Memory (LSTM) [43] recurrent units. However, although LSTNet, DSANet, and MLCNN are cutting-edge multivariate time-series forecasting algorithms, they do not explicitly address per-variable and inter-time-series dependencies, which weakens their forecasting ability in the face of higher-dimensional data. Therefore, the state-of-the-art in time-series forecasting is bounded to a bidimensional space in which we understand the forecasting process by a non-linear function between time and variables.

Differently, we hypothesize that *time-series are dependent on their inner variables, which are observations from themselves, and from outer variables provided by different time series that share the same timestream*. For instance, the evolution of a biological species is not solely related to observations from itself but also from other species that share the same habitat, as they are all part of the same food chain. The time series gains an increased dimensionality by considering the variables and the dependency aspect during the analysis. Consequently, a previously considered bidimensional problem, in which a model's forecasting ability comes from observing relationships of variables over time, now becomes tridimensional, where forecasting means understanding the entanglement between variables of different time-series that co-occur in time. Accordingly, time-series define an event that is not a consequence of a single chain of observations but a set of synchronous observations of many time-series.

For example, during the Coronavirus Pandemic, it is paramount to understand the disease's time-aware behavior in every country. Despite progressing in different moments and locations, the pandemic's underlying mechanisms are supposed to follow similar (and probably interconnected) patterns. Along these lines, looking individually at the development of the pandemic in each country, one can describe the problem in terms of multiple variables, like the number of confirmed cases, recovered people, and deaths. However, when looking at all countries at once, the problem yields an additional data dimension, and each country becomes a multivariate sample of a broader problem, such as depicted in Fig. 1. In linguistic terms, we refer to such a problem as multiple multivariate time-series forecasting.

Along with these premises, in this study, we contribute with an unpreceded neural network that emerges from a graph-based time-aware auto-encoder with linear and non-linear components working in parallel to forecast multiple multivariate time-series simultaneously, named after Recurrent Graph Evolution Neural Network (ReGENN). *We refer to evolution as the natural progression of a process where the neural network iteratively optimizes a graph representing observations from the past until it reaches an evolved version of itself that generalizes on future data still to be observed*. Accordingly, the underlying network structure of ReGENN is powered by two Graph Soft Evolution (GSE) layers, a further contribution of this study. The GSE stands for a graph-based learning-representation layer that enhances the encoding and

decoding processes by learning a shared graph across different time-series and timestamps.

The results we present are based on an extensive set of experiments, in which REGENNN surpassed a set of 49 competing algorithms from the fields of deep learning, machine learning, and time-series; among of which are single-target, multi-output, and multi-task regression algorithms in addition to univariate and multivariate time-series forecasting algorithms. Aside from surpassing the state-of-the-art, REGENNN remained effective after three rounds of 30 ablation tests through distinct hyperparameters. All experiments were carried out over the SARS-CoV-2, Brazilian Weather, and PhysioNet datasets. In the task of epidemiology modeling on the SARS-CoV-2 dataset, we had improvements of at least 64.87 percent. We outperformed the task of climate forecasting on the Brazilian Weather dataset by at least 11.96 percent and patient monitoring on intensive care units on the PhysioNet dataset by 7.33 percent. Furthermore, we analyzed the results using the *Evolution Weights* from the GSE layers, which are the intermediate hidden adjacency matrices that arise from the graph-evolution process after going through the cosine similarity activation, showing that graphs shed new light on the understanding of non-linear black-box models. Since multiple multivariate time-series is an ascending research topic, we understand REGENNN has implications in multiple domains, like economics, social sciences, and biology, in which different time-series share the same timestream and co-occur in time, mutually influencing one another.

In order to present our contributions, this paper is further divided into four sections. We begin by proposing a layer and neural network architecture, besides detailing the methods used along with this study. Subsequently, we display the experimental results compared to previous literature. Next, we provide an overall discussion on our proposal and the achieved results. Finally, we present the conclusions and final remarks. Alongside, the Supplementary Material presents extended methods and additional results, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TPAMI.2021.3076155.

2 METHOD

2.1 Preliminaries

Hereinafter, we use bold uppercase letters to denote multi-dimensional matrices (e.g., $\mathbf{X}$), bold lowercase letters to vectors (e.g., $\mathbf{x}$), and calligraphic letters to sets (e.g., $\mathcal{X}$). Matrices, vectors, and sets can be used with subscripts. For example, the element in the ith row and jth columns of a matrix is $\mathbf{X}_{ij}$, the ith element of a vector is $\mathbf{x}_i$, and the jth element of a set is $\mathcal{X}_j$. The transposed matrix of $\mathbf{X} \in \mathbb{R}^{m \times n}$ is $\mathbf{X}^{\mathrm{T}} \in \mathbb{R}^{n \times m}$, and the transposed vector of $\mathbf{x} \in \mathbb{R}^{m \times 1}$ is $\mathbf{x}^{\mathrm{T}} \in \mathbb{R}^{1 \times m}$, where m and n are arbitrary dimensions. We display in Table 1 a summary of all context-specific notations.

2.2 Graph Soft Evolution

Graph Soft Evolution (GSE) stands for a representation-learning layer that, given a training dataset, builds a graph in the form of an adjacency matrix, as in Fig. 2. The GSE layer receives no graph as input but a set of multiple multivariate time-series. The graph is built by tracking pairs of co-occurring variables, one sample at a time, and merging the results into a single co-occurrence graph shared among samples and timestamps. We define co-occurring variables as two variables, from a multivariate time-series, with a non-zero value in the same timestamp – in that case, we say one variable influences

TABLE 1
Summary of Context-Specific Notations

Notation	Definition
$\omega \in \mathbb{N}^+$	Sliding window size
$w, z \in \mathbb{N}^+$	Number of training and testing (*i.e.*, stride) timestamps
$s, t, v \in \mathbb{N}^+$	Number of samples, timestamps, and variables
$\mathbf{T} \in \mathbb{R}^{s \times t \times v}$	Tensor of multiple multivariate time-series
$\mathbf{Y} \in \mathbb{R}^{s \times \omega \times v}$	Batched input of the first GSE and the Autoregression layers
$\mathbf{Y}_\alpha \in \mathbb{R}^{s \times \omega \times v}$	Output of the first GSE and input of the encoder layers
$\mathbf{Y}_\varepsilon \in \mathbb{R}^{s \times \omega \times v}$	Output of the encoder and input of the decoder layers
$\tilde{\mathbf{Y}}_\varepsilon \in \mathbb{R}^{s \times z \times v}$	Output from the first recurrent unit and input to the second one
$\tilde{\mathbf{Y}} \in \mathbb{R}^{s \times z \times v}$	Output of the second recurrent unit and input of the second GSE layer
$\mathbf{Y}_\psi \in \mathbb{R}^{s \times z \times v}$	Non-linear output yielded by the second GSE layer
$\mathbf{Y}_\lambda \in \mathbb{R}^{s \times z \times v}$	Linear output provided by the Autoregression layer
$\widehat{\mathbf{Y}} \in \mathbb{R}^{s \times z \times v}$	The final result from the merging of the linear and non-linear outputs
$\widehat{\mathbf{T}} \in \mathbb{R}^{s \times z \times v}$	The ground truth expected from merging the linear and non-linear outputs
$\mathcal{G} = \langle \mathcal{V}, \mathcal{E} \rangle$	Graph in which $\mathcal{V}$ is the set of nodes and $\mathcal{E}$ the set of edges
$\mathbf{A} \in \mathbb{R}^{v \times v}$	Adjacency matrix of co-occurring variables
$\mathbf{A}_\mu \in \mathbb{R}^{v \times v}$	Neighbor-smoothed per-variable embedding shared between GSE layers
$\mathbf{A}_\phi \in \mathbb{R}^{v \times v}$	Evolved adjacency matrix produced by the second GSE layer
$\mathbf{U} \circ \mathbf{V}$	Batch-wise Hadamard product between matrices $\mathbf{U}$ and $\mathbf{V}$
$\mathbf{U} \cdot \mathbf{V}$	Batch-wise scalar product between matrices $\mathbf{U}$ and $\mathbf{V}$
$\lVert \cdot \rVert_{\mathrm{F}}$	The Frobenius norm of a given vector or matrix
$\varphi(\cdot)$	Dropout regularization function
$\sigma_g(\cdot)$	Sigmoid activation function
$\sigma_h(\cdot)$	Hyperbolic tangent activation function
$\cos_\theta(\cdot)$	Cosine matrix-similarity activation function
RELU$(\cdot)$	Rectified exponential linear unit activation function
SOFTMAX$(\cdot)$	Normalized exponential function activation function

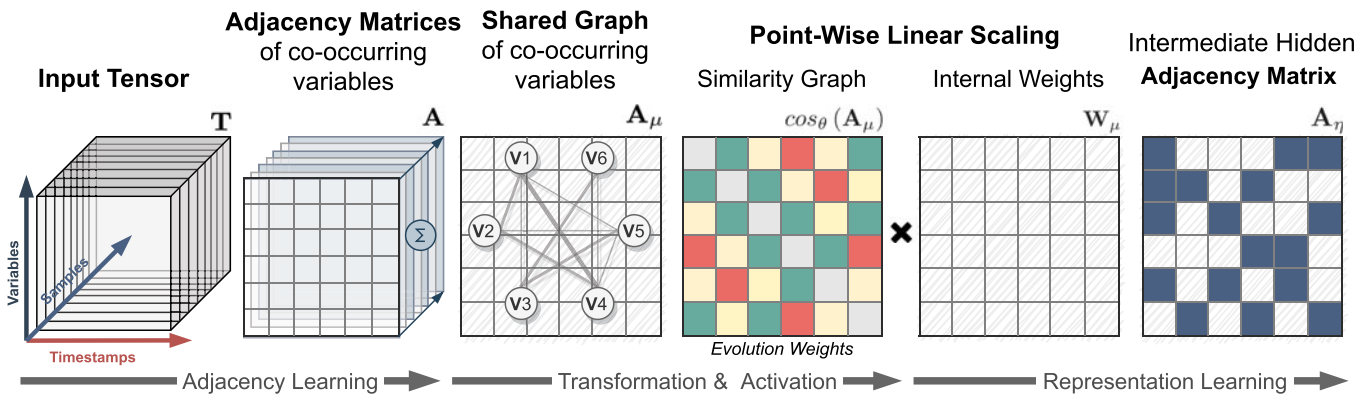


Fig. 2. Graph Soft Evolution representation-learning, in which the set of multiple multivariate time-series is mapped into adjacency matrices of co-occurring variables. The matrices are element-wise summed to generate a shared graph among samples, which, after a linear transformation, goes through a similarity activation function and is scaled by an element-wise multiplication to produce an intermediate hidden adjacency-matrix with similarity properties inherent to the shared graph.

another and is influenced back. The co-occurrence graph is the projection of a tridimensional tensor, $\mathbf{T}$, $\mathbf{T} \in \mathbb{R}^{s \times w \times v}$, into a bidimensional one, $\mathbf{A}$, $\mathbf{A} \in \mathbb{R}^{v \times v}$, describing variables' pairwise time-invariant relationships.

The co-occurrence graph $\mathcal{G} = < \mathcal{V}, \mathcal{E} >$ is symmetric and weighted. It is composed of a set $\mathcal{V}$ of $|\mathcal{V}|$ nodes equal to the number of variables and another set $\mathcal{E}$ of $|\mathcal{E}|$ non-directed edges equal to the number of co-occurring variables. A node $v \in \mathcal{V}$ corresponds to a variable from the time-series multivariate domain, and an edge $e \in \mathcal{E}$ is an ordered pair $< u, v > \equiv < v, u >$ of co-occurring variables $u, v \in \mathcal{V}$. The edges' weight f corresponds to the summation of the values of the variables $u, v \in \mathcal{V}$ whenever they co-occur in time, such that $f(u, v) = \sum_{i=0}^{s-1} \sum_{j=0}^{w-1} \mathbf{T}_{i,j,u} + \mathbf{T}_{i,j,v}$. We use summation as the graph-merging operator when building $\mathbf{A}$ from $\mathcal{G}$ because, in the face of a zero-one input, a multiplicative operator would provide values close to zero, division would make those values overgrow toward infinity, subtraction would turn some of them into negative, while summation provides consistently positive values upper bounded to $2 \times (s \times w)$, helping to sustain the magnitude of variables' values. This way, the whole graph is bounded to w, which is the number of timestamps existing in the training portion of the input tensor, and if a pair of variables never co-occur in the training data, no edge will be assigned to the graph, meaning that $< u, v > \notin \mathcal{E}$, and $f(u, v) = 0$.

The GSE layer for an arbitrary graph is formulated as

$$\mathbf{A}_\mu = \mathbf{W}_\mu \cdot \mathbf{A} + \mathbf{b}_\mu \tag{1.1}$$

$$\mathbf{A}_\eta = \mathbf{W}_\eta \circ cos_\theta(\mathbf{A}_\mu) + \mathbf{b}_\eta \tag{1.2}$$

$$\mathbf{Y}_\alpha = \mathbf{W}_\alpha \cdot \varphi\big(\mathbf{Y} \cdot \mathbf{A}_\eta\big) + \mathbf{b}_\alpha, \tag{1.3}$$

where $\mathbf{W}_\alpha$, $\mathbf{W}_\eta$, $\mathbf{W}_\mu \in \mathbb{R}^{v \times v}$ are symmetrically-unconstrained weights and $\mathbf{b}_\alpha$, $\mathbf{b}_\eta$, $\mathbf{b}_\mu \in \mathbb{R}^v$ the bias. In Eq. (1.1), the layer starts by employing a linear transformation to the shared adjacency matrix $\mathbf{A}$, providing a per-variable embedding $\mathbf{A}_\mu$ after smoothing across neighbors. Subsequently, in Eq. (1.2), it uses the cosine similarity (see the Supplementary Material, available online) on the output of Eq. (1.1), i.e., $cos_\theta(\mathbf{A}_\mu)$, which is an intermediate activation function that provides the *Evolution Weights* a symmetric per-variable similarity adjacency-matrix. Under the same

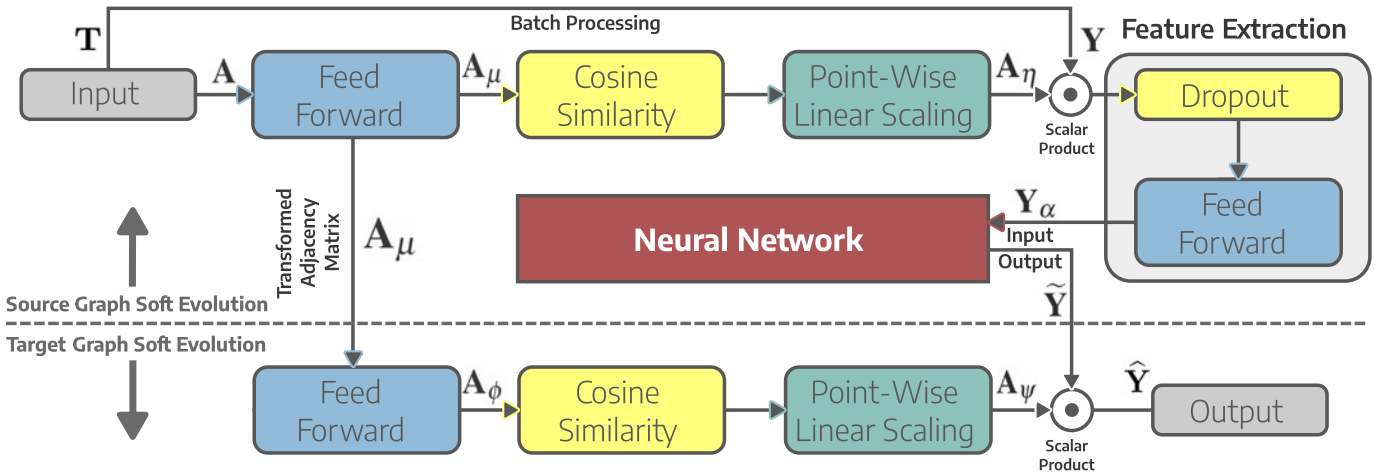


Fig. 3. Graph Soft Evolution layers assembled for evolution-based learning. In such a case, the first GSE layer's output (i.e., source) will feed further layers of the neural network, whose result goes through the second GSE layer (i.e., target). The GSE, as the last layer, does not use regularizers nor linear transformations before the output. Contrarily, it outputs the result from the scalar product between the learned representation and the data propagated throughout the network.

equation, the *Evolution Weights* goes through a point-wise operator with a symmetrically-unconstrained weight matrix that enables the network to filter meaningful similarities from spurious similarities by learning that two variables influence one another on different scales and resulting in $\mathbf{A}_\eta$. Next, in Eq. (1.3), it performs a batch-wise matrix-by-matrix multiplication between the adjacency matrix from Eq. (1.2) and the batched input tensor $\mathbf{Y}$ to combine the information from the graph, which generalizes samples and timestamps, with the time-series. The result will be followed by a dropout regularizer [44] and batch-wise matrix-by-matrix multiplication, where the final features from joining both tensors will be extracted and forwarded to the next layer of the network.

The evolution concept comes from the cooperation between two GSE layers, one at the beginning (i.e., right after the input) and the other at the end (i.e., right before the output) of a neural network, such as in the example shown in Fig. 3. As evolution arises from sharing hidden weights between a pair of non-sequential layers, we named this process after *Soft Evolution*. Accordingly, the first layer (i.e., source) aims to learn the weights to scale the matrix and produce $\mathbf{A}_\mu$. Such a result is the input of the second GSE layer (i.e., target), and it will be used for learning the evolved version of the adjacency matrix, referred to as $\mathbf{A}_\phi$ and produced as in Eq. (1.1). Notice that in Fig. 3, the source layer is different from the target one because we disregard the regularizer φ, trainable weights $\mathbf{W}_\alpha$, and bias $\mathbf{b}_\alpha$ from Eq. (1.3). They aim to enhance the feature-learning processes when multiple layers are stacked together and, as the last layer, GSE provides the output from already learned features through one last scalar product between the data propagated throughout the network, i.e., $\widetilde{\mathbf{Y}}$, and the intermediate hidden adjacency-matrix, i.e., $\mathbf{A}_\psi$.

One can see that the source GSE layer has two constant inputs: the graph and input tensor. The target GSE layer has two dynamic inputs, the shared graph from the source GSE layer and input propagated throughout the network. In this work scope, we use an auto-encoder between GSE layers to learn data codings from the source layer's output, which will be decoded into a representation closest to the expected output and later re-scaled by the target layer. In this sense, while the first layer learns a graph from the training data (i.e., past data) working as a pre-encoding feature-extraction layer, the second one re-learns (i.e., evolve) a graph at the end of the forecasting process based on future data, working as a post-decoding output-scaling layer. When joining the GSE layers with the auto-encoder, we assemble the Recurrent Graph Evolution Neural Network (REGENN).

2.3 Recurrent Graph Evolution Neural Network

REGENN is a graph-based time-aware auto-encoder with linear and non-linear components on parallel data-flows working together to provide future predictions based

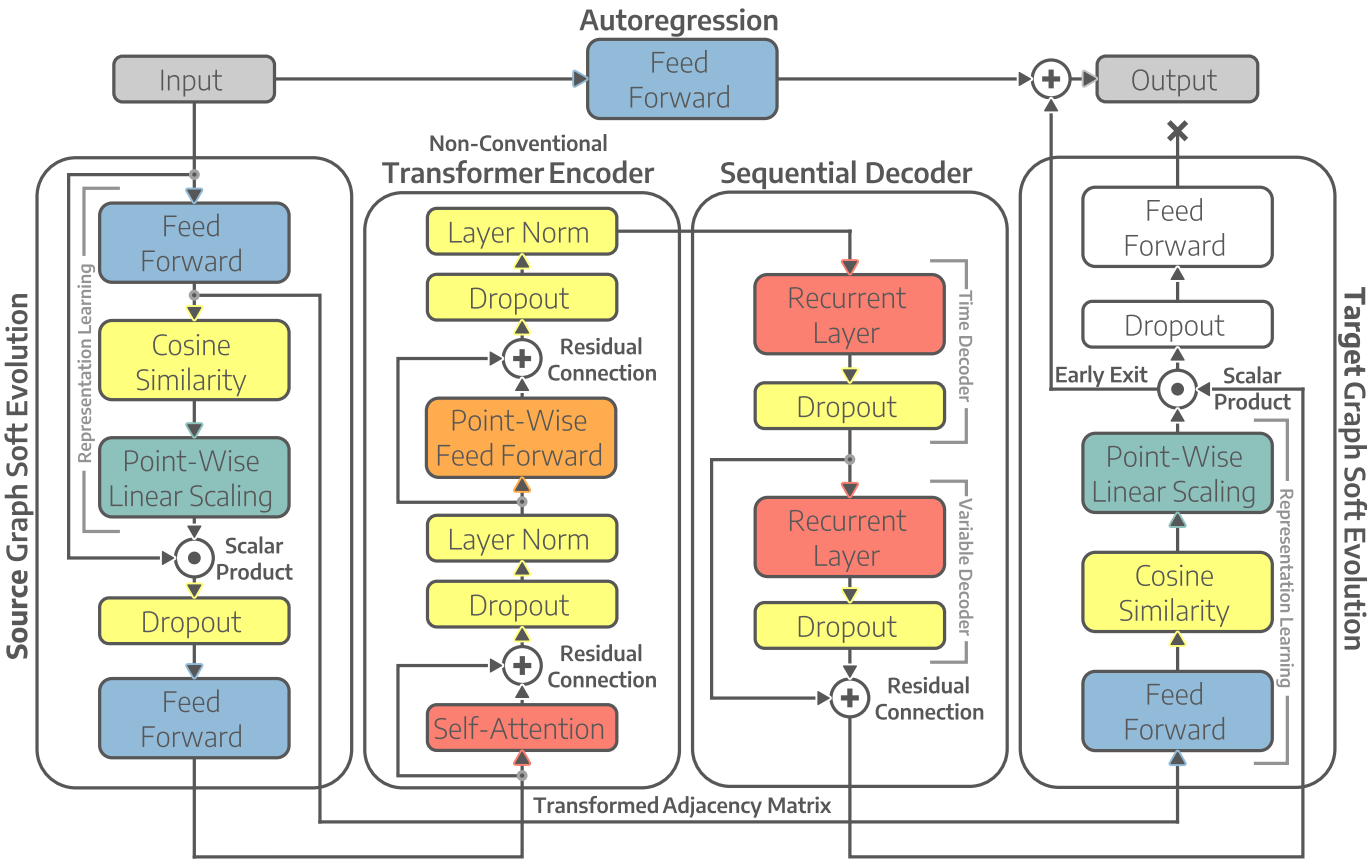


Fig. 4. Data diagram of the Recurrent Graph Evolution Neural Network (REGENN), which has a linear component parallel to a non-linear one. The linear component has a feed-forward layer, and the non-linear one has an auto-encoder and two GSE layers. Although equal to the first, the last GSE layer yields an early output as it is not stacked with another layer.

on past observations. The linear component is the autoregression implemented as a feed-forward layer, and the non-linear component is made of an encoder and a decoder module powered by a pair of GSE layers. Fig. 4 shows how these components communicate from the input to the output, and, in the following, we detail their operation.

2.3.1 Autoregressive Component

The non-periodical changes and constant progressions of the series across time usually decrease the performance of the network. That is because the output scale loses significance compared to the input, which comes from the complexity and non-linear nature of neural networks in time-series forecasting tasks [39]. Following a systematic strategy to deal with such a problem [45], [46], ReGENN leverages from an Autoregressive (AR) layer working as a linear feed-forward shortcut between the input and output, which for a tridimensional input, is algebraically defined as

$$\mathbf{Y}_\lambda = \mathbf{W} \cdot \mathbf{Y} + \mathbf{b} \equiv \left(\sum_{i=1}^{\omega} \mathbf{W}_{i,z} \times \mathbf{Y}_{s,i,v} \right) + \mathbf{b}, \tag{2}$$

where $\mathbf{W} \in \mathbb{R}^{\omega \times z}$ are the weights and $\mathbf{b} \in \mathbb{R}^{z}$ the bias to be learned. The output of the linear component, i.e., $\mathbf{Y}_\lambda \in \mathbb{R}^{s \times z \times v}$ as in Eq. (2), is element-wise added to the non-linear component's output, i.e., $\mathbf{Y}_\psi \in \mathbb{R}^{s \times z \times v}$, so to produce the final predictions of the network $\widehat{\mathbf{Y}} \in \mathbb{R}^{s \times z \times v}$, formally given as $\widehat{\mathbf{Y}} = \mathbf{Y}_\lambda + \mathbf{Y}_\psi$. Subsequently, we describe the auto-encoder that produces the non-linear output of ReGENN.

2.3.2 Encoder

We use a non-conventional Transformer Encoder [41] that employs self-attention to learn an encoding from the features forwarded by the GSE layer. It consists of multiple encoders joined through the scaled dot-product attention into a single set of encodings through multi-head attention. The number of expected features by the Transformer Encoder must be a multiple of the number of heads in the multi-head attention. Our encoder's non-conventionality comes from the fact that the first GSE layer's output goes through a single scaled dot-product attention on a single-head attention task. That is because the number of features produced by the encoder is equal to the length of the sliding window, and through single-head attention, the window can assume any length. The encoder module is defined as follows:

$$\mathbf{Y}_\varepsilon = \textsc{Self-Attention}(\mathrm{Q} : \mathbf{Y}_\alpha, \mathrm{K} : \mathbf{Y}_\alpha, \mathrm{V} : \mathbf{Y}_\alpha) \tag{3a}$$

$$\mathbf{Y}_\varepsilon = \textsc{Layer-Norm}_{\gamma,\beta}(\mathbf{Y}_\varepsilon + \varphi(\mathbf{Y}_\varepsilon)) \tag{3b}$$

$$\mathbf{Y}_\varepsilon = \mathbf{W}_\varepsilon \cdot \varphi(\textsc{ReLU}(\mathbf{W}_\iota \cdot \mathbf{Y}_\varepsilon + \mathbf{b}_\iota)) + \mathbf{b}_\varepsilon \tag{3c}$$

$$\mathbf{Y}_\varepsilon = \textsc{Layer-Norm}_{\gamma,\beta}(\mathbf{Y}_\varepsilon + \varphi(\mathbf{Y}_\varepsilon)), \tag{3d}$$

where self-attention in Eq. (3a) is a particular case of the multi-head attention, in which the input *query* Q, *key* K, and *value* V of the scaled dot-product attention, i.e., $\textsc{softmax}\left(\mathrm{Q} \cdot \mathrm{K}^{\mathrm{T}} \div \sqrt{d_k}\right) \cdot \mathrm{V}$, are equal; and d_k is the dimension of the keys. The attention results are followed by a dropout regularization [44], a residual connection [47], and a layer normalization [48] as in Eq. (3b) to ensure generalization. The first two layers work to avoid overfitting and gradient vanishing, while the last one normalizes the output such that the samples among the input have zero mean and unit variance $\gamma(\Delta(\mathbf{Y}_\varepsilon + \varphi(\mathbf{Y}_\varepsilon))) + \beta$, where Δ is the normalization function, and γ and β are parameters to be learned. After, in Eq. (3c), the intermediate encoding goes through a double linear layer, a point-wise feed-forward layer, which, in this case, consists of two linear transformations in sequence with a ReLU activation in between, having the weights $\mathbf{W}_\varepsilon, \mathbf{W}_\iota$ and bias $\mathbf{b}_\varepsilon, \mathbf{b}_\iota$ as optimizable parameters. Finally, the transformed encoding goes through one last set of generalizing operations, as shown in Eq. (3d). The resulting encoding $\mathbf{Y}_\varepsilon \in \mathbb{R}^{s \times \omega \times v}$ is a tensor with the time-axis length matching the size of the sliding window ω.

2.3.3 Decoder

The previous encoding will be decoded by two sequence-to-sequence layers, which are Long Short Term Memory (LSTM) [43] units. The decoder operates in two of the tridimensional axes of the encoding, the time-axis and variable-axis, once at a time. Under the sample and time-axis, the time-axis decoder tracks temporal dependencies, looks for a set of weights that generalizes across variables, and translates the window-sized input into a stride-sized output

$$\begin{aligned}
\mathbf{f}^{(t)}(v) &= \sigma_g\Big(\mathbf{W}_\mathbf{f}^{(t)} \cdot \Big[\mathbf{h}^{(t)}(v-1), \mathbf{Y}_\varepsilon^{(t)}(v)\Big] + \mathbf{b}_\mathbf{f}^{(t)}\Big) \\
\mathbf{i}^{(t)}(v) &= \sigma_g\Big(\mathbf{W}_\mathbf{i}^{(t)} \cdot \Big[\mathbf{h}^{(t)}(v-1), \mathbf{Y}_\varepsilon^{(t)}(v)\Big] + \mathbf{b}_\mathbf{i}^{(t)}\Big) \\
\mathbf{o}^{(t)}(v) &= \sigma_g\Big(\mathbf{W}_\mathbf{o}^{(t)} \cdot \Big[\mathbf{h}^{(t)}(v-1), \mathbf{Y}_\varepsilon^{(t)}(v)\Big] + \mathbf{b}_\mathbf{o}^{(t)}\Big) \\
\widetilde{\mathbf{C}}^{(t)}(v) &= \sigma_h\Big(\mathbf{W}_\mathbf{C}^{(t)} \cdot \Big[\mathbf{h}^{(t)}(v-1), \mathbf{Y}_\varepsilon^{(t)}(v)\Big] + \mathbf{b}_\mathbf{C}^{(t)}\Big) \\
\mathbf{C}^{(t)}(v) &= \mathbf{f}^{(t)}(v) \circ \mathbf{C}^{(t)}(v-1) + \mathbf{i}^{(t)}(v) \circ \widetilde{\mathbf{C}}^{(t)}(v-1) \\
\mathbf{h}^{(t)}(v) &= \varphi\Big(\mathbf{o}^{(t)}(v) \circ \sigma_h\Big(\mathbf{C}^{(t)}(v)\Big)\Big),
\end{aligned} \tag{4}$$

where v is the vth variable of the tth time-series group, and the weights $\mathbf{W}_\mathbf{f}^{(t)}, \mathbf{W}_\mathbf{i}^{(t)}, \mathbf{W}_\mathbf{C}^{(t)}, \mathbf{W}_\mathbf{o}^{(t)} \in \mathbb{R}^{\omega \times z}$ and bias $\mathbf{b}_\mathbf{f}^{(t)}, \mathbf{b}_\mathbf{i}^{(t)}, \mathbf{b}_\mathbf{C}^{(t)}, \mathbf{b}_\mathbf{o}^{(t)} \in \mathbb{R}^{z}$ are parameters to be learned. Along with Eq. (4), we refer to $\mathbf{f}$ as the forget gate's activation vector, $\mathbf{i}$ as the input and update gate's activation vector, $\mathbf{o}$ as the output gate's activation vector, $\widetilde{\mathbf{C}}$ as the cell input activation vector, $\mathbf{C}$ as the cell state vector, and $\mathbf{h}$ as the hidden state vector. The last hidden state vector goes through a dropout regularization φ before the next LSTM in the sequence.

Under the time and variable-axis, the next recurrent unit decodes the variable-axis from the partially-decoded encoding, searches for a set of weights that generalizes across time, and translates patterns arising from different variables on the same timestamp. The set of variables within the time-series does not necessarily imply a sequence, which does not interfere in the decoding process as long as the variables are always kept in the same order; a common approach used with boosting trees, such as the XGBoost [49] algorithm. The

second LSTM, in which t is the tth timestamp of the vth variable group, is algebraically defined as follows

$$\begin{aligned}
\mathbf{f}^{(v)}(t) &= \sigma_g\Big(\mathbf{W}_{\mathbf{f}}^{(v)} \cdot \Big[\mathbf{h}^{(v)}(t-1), \widetilde{\mathbf{Y}}_{\varepsilon}^{(v)}(t)\Big] + \mathbf{b}_{\mathbf{f}}^{(v)}\Big) \\
\mathbf{i}^{(v)}(t) &= \sigma_g\Big(\mathbf{W}_{\mathbf{i}}^{(v)} \cdot \Big[\mathbf{h}^{(v)}(t-1), \widetilde{\mathbf{Y}}_{\varepsilon}^{(v)}(t)\Big] + \mathbf{b}_{\mathbf{i}}^{(v)}\Big) \\
\mathbf{o}^{(v)}(t) &= \sigma_g\Big(\mathbf{W}_{\mathbf{o}}^{(v)} \cdot \Big[\mathbf{h}^{(v)}(t-1), \widetilde{\mathbf{Y}}_{\varepsilon}^{(v)}(t)\Big] + \mathbf{b}_{\mathbf{o}}^{(v)}\Big) \\
\widetilde{\mathbf{C}}^{(v)}(t) &= \sigma_h\Big(\mathbf{W}_{\mathbf{C}}^{(v)} \cdot \Big[\mathbf{h}^{(v)}(t-1), \widetilde{\mathbf{Y}}_{\varepsilon}^{(v)}(t)\Big] + \mathbf{b}_{\mathbf{C}}^{(v)}\Big) \\
\mathbf{C}^{(v)}(t) &= \mathbf{f}^{(v)}(t) \circ \mathbf{C}^{(v)}(t-1) + \mathbf{i}^{(v)}(t) \circ \widetilde{\mathbf{C}}^{(v)}(t-1) \\
\mathbf{h}^{(v)}(t) &= \widetilde{\mathbf{Y}}_{\varepsilon} + \varphi\Big(\mathbf{o}^{(v)}(t) \circ \sigma_h\Big(\mathbf{C}^{(v)}(t)\Big)\Big),
\end{aligned} \tag{5}$$

where $\widetilde{\mathbf{Y}}_{\varepsilon}$ is the partially-decoded encoding, and the weights $\mathbf{W}_{\mathbf{f}}^{(v)}, \mathbf{W}_{\mathbf{i}}^{(v)}, \mathbf{W}_{\mathbf{C}}^{(v)}, \mathbf{W}_{\mathbf{o}}^{(v)} \in \mathbb{R}^{z \times z}$ and bias $\mathbf{b}_{\mathbf{f}}^{(v)}, \mathbf{b}_{\mathbf{i}}^{(v)}, \mathbf{b}_{\mathbf{C}}^{(v)}, \mathbf{b}_{\mathbf{o}}^{(v)} \in \mathbb{R}^{z}$ are internal parameters. The description of the notations within Eq. (4) holds for Eq. (5). Their difference is the residual connection with the partially-decoded encoding $\widetilde{\mathbf{Y}}_{\varepsilon}$ at the last hidden state vector after the dropout regularization φ. After the decoding is complete, the last recurrent unit output $\widetilde{\mathbf{Y}}$ goes through the second GSE layer, producing the non-linear output $\mathbf{Y}_{\psi}$ of ReGENN.

2.4 Optimization Strategy

ReGENN operates on a tridimensional space shared between samples, timestamps, and variables. In such a space, it carries out a time-based optimization strategy. The training process iterates over the time-axis of the dataset, showing to the network how the variables within a subset of time-series behave as time goes by and later repeating the process through subsets of different samples in distinct batches of preset window-size. However, the shared graph $\mathbf{A}$ is built during the first epoch of the training phase. Such a process happens in parallel to the optimization process, not impacting training stability nor convergence. After that point, $\mathbf{A}$ will remain as it is during the training and testing.

The network's weights are shared among the entire dataset and optimized towards best generalization simultaneously across samples, timestamps, and variables. We used Adam [50], a gradient descent-based algorithm, to optimize the model and, as the optimization criterion, the Mean Absolute Error (MAE), which is a generalization of the Support Vector Regression [51] with soft-margin criterion where Ω is the set of internal parameters of ReGENN, $\widehat{\mathbf{Y}}$ is the network's output, and $\widehat{\mathbf{T}}$ the ground truth

$$\underset{\Omega}{\textit{minimize}} \sum_{i=1}^{s} \sum_{j=1}^{w} \Big| \widehat{\mathbf{Y}}_{i,j} - \widehat{\mathbf{T}}_{i,j} \Big|. \tag{6}$$

Due to the SARS-CoV-2 data behave as a streaming dataset, we adopted a transfer learning approach to train the network on that dataset. Transfer learning shares knowledge across different domains by using pre-trained weights of another neural network. The approach we adopted, although different, resembles Online Deep Learning [52]. The main idea is to train the network on incremental slices of the time-axis, such that the pre-trained weights of a previous slice are used to initialize the weights of the network in the next slice. This technique aims not only to achieve better forecasting performance but also to show that ReGENN is superior to other algorithms throughout the pandemic.

3 Experiments

3.1 Experimental Setup

3.1.1 Datasets

The results are based on three datasets, all of which are multi-sample, multivariate, and vary over time. The first dataset describes the Coronavirus Pandemic, referred to as SARS-CoV-2, made available by John Hopkins University [53]. It describes 3 variables through 120 days for 188 countries and varies from the first day of the pandemic to the day it completed four months of duration. The second one is the Brazilian Weather dataset collected from 253 sensors during 1,095 days regarding 4 variables. The third dataset is from the 2012 PhysioNet Computing in Cardiology Challenge [54], from which we are using 9 variables across 48 hours recorded from 11,988 ICU patients.

The variables within the datasets are:

- *SARS-CoV-2:* Number of Recovered, Number of Infected, and Number of Deaths;
- *Brazilian Weather:* Minimum Temperature, Maximum Temperature, Solar Radiation, and Rainfall; and,
- *PhysioNet:* Non-Invasive Diastolic Arterial Blood Pressure (mmHg), Non-Invasive Systolic Arterial Blood Pressure (mmHg), Invasive Diastolic Arterial Blood Pressure (mmHg), Non-Invasive Mean Arterial Blood Pressure (mmHg), Invasive Systolic Arterial Blood Pressure (mmHg), Invasive Mean Arterial Blood Pressure (mmHg), Urine Output (mL), Heart Rate (bpm), and Weight (Kg).

The number of datasets that can be used for multiple multivariate time-series forecasting is still limited. Although the ones we experimented with have a small number of variables, there is no theoretical upper bound for the number of samples, timestamps, and variables ReGENN can handle. However, as ReGENN deals with dense tridimensional tensors, increasing any of the axes $(samples \times time \times variables)$ in size will make the tensor grow at an almost-cubic rate, quickly overflowing the GPU Memory. Accordingly, ReGENN ends up suffering from a trade-off between the dataset's size and the available hardware.

3.1.2 Data Pre-Processing

The datasets were individually min-max normalized into a zero-one scale on the variable-axis to avoid gradient spikes and speed up training. We used the training data to define the min-max parameters required for normalization. Such parameters were applied for normalizing the test data as well. As a consequence of normalizing the entire dataset, the network's output will follow a similar scale. Therefore, the output was inversely transformed to what it was before the normalization for evaluating the results.

A simplistic yet effective approach to train time-series algorithms is through the Sliding Window technique [55], also referred to as Rolling Window. The window size is well known to be a highly sensitive hyperparameter [56], [57]. Consequently, we followed a non-tunable approach, in which we set the window size before the experiments, just taking into consideration the context and domain of the datasets. These values were used across all window-based

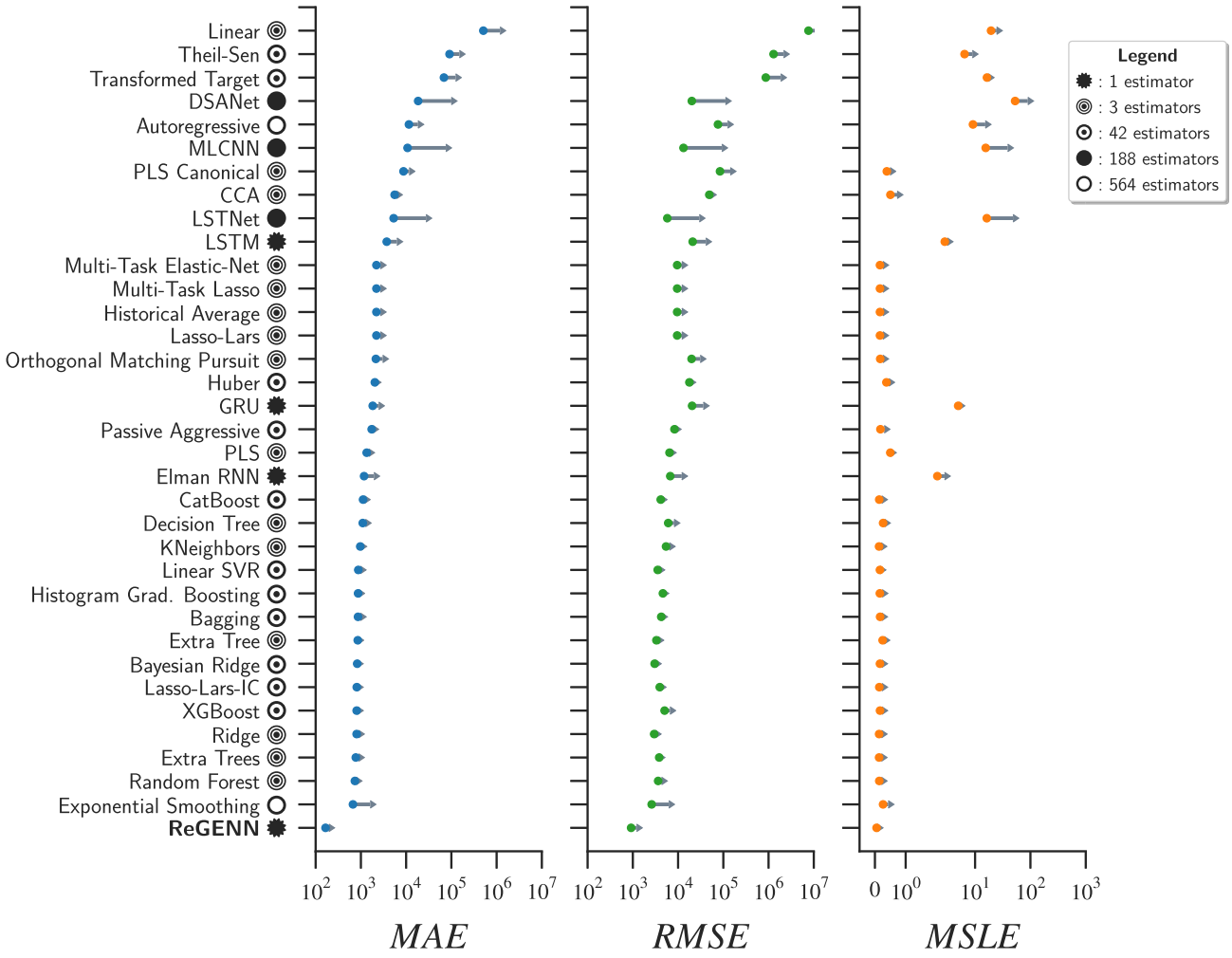


Fig. 5. Baseline results for the SARS-CoV-2 dataset over the Mean Absolute Error (MAE), Root Mean Square Error (RMSE), and Mean Squared Logarithmic Error (MSLE). The results are presented in descending order of MAE (i.e., worst performance at the top). The results confirmed ReGENN's superior performance as it is the algorithm with the lowest error and standard deviation – the improvement in the experiment was no lower than 64.87 percent. In the image, the algorithms are symbol-encoded based on their type and number of estimators; we use gray arrows to report the standard deviation of the results. The negative deviation, which is equal to the positive one, was suppressed for improved readability.

experiments, including the baselines and ablation tests. It is noteworthy that most of the machine-learning algorithms are not meant to handle time-variant data, such that no sliding window was used in those cases. Conversely, we considered training timestamps as features and those reserved for testing as multi-task regression labels.

On the deep learning algorithms, we used a window size of 7 days for training and reserved 7 days for validation (between the training and test sets) to predict the last 14 days of the SARS-CoV-2 dataset. The 7-7-14 split idea comes from the disease incubation period, which is of 14 days. On the other hand, we used a window size of 84 days and reserved 28 days for validation to predict the last 56 days in the Brazilian Weather dataset. The 84-28-56 split is based on the seasonality of the weather data, such that we will look to the previous 3 months (a weather-season window) to predict the last 2 months of the upcoming season. Finally, we used a window size of 12 hours for training and 6 hours for validation to predict the last 6 hours of the PhysioNet dataset. The 12-6-6 split comes from the fact that patients in ICUs are in a critical state, such that predictions within 24 hours are more useful than long-term predictions.

3.1.3 Algorithms & Ensembles

Many existing algorithms are limited because they neither support multi-task nor multi-output regression, making these algorithms even more limited to tasks when data is tridimensional. The most straightforward yet effective approach we followed to compare them to ReGENN was to create a chain of ensembles.[1] In such a case, each estimator makes predictions on order specified by the chain using all of the available features provided to the model plus the predictions of earlier models in the chain. The number of estimators in each experiment varies according to the type of the ensemble and the type of the algorithms, and the final performance is the average of each estimator's performance. For simplicity sake, we grouped the algorithms as follows:

- ✹ Corresponds to tridimensional compliant algorithms of single estimators;
- ● Describes multivariate algorithms – s estimators, one estimator for each sample;

1. See *Regressor Chain* at https://bit.ly/3hBfxTA.

TABLE 2
Ablation Results for the SARS-CoV-2 Dataset Using ReGENN's Data-Flow but no GSE Layer

Recurrent Unit (RU)	Elman RNN			GRU			LSTM		
Network Architecture	MAE	RMSE	MSLE	MAE	RMSE	MSLE	MAE	RMSE	MSLE
${}_{\mathrm{B}}$RU	424.32	2819.11	0.07	3130.47	28063.07	7.78	1800.23	20264.27	3.95
$\mathrm{E} \rightarrow {}_{\mathrm{B}}\mathrm{RU}$	4161.35	28755.33	11.08	596.77	3446.67	0.17	704.80	4088.05	0.20
$(\mathrm{E} \rightarrow {}_{\mathrm{B}}\mathrm{RU} + {}_{\mathrm{B}}\mathrm{RU}) + \mathrm{AR}$	1624.44	17245.24	3.84	388.93	1888.70	0.09	1598.03	19056.28	3.89
$(\mathrm{E} \rightarrow {}_{\mathrm{B}}\mathrm{RU} + {}_{\mathrm{U}}\mathrm{RU}) + \mathrm{AR}$	**257.15**	**1438.72**	**0.09**	366.06	2158.32	0.10	4064.52	31196.10	11.30
$(\mathrm{E} \rightarrow {}_{\mathrm{B}}\mathrm{RU}) + \mathrm{AR}$	1471.49	17050.08	3.74	1502.20	16799.96	3.67	**211.93**	**1181.31**	**0.08**
$\mathrm{E} \rightarrow {}_{\mathrm{U}}\mathrm{RU}$	1949.65	16925.69	3.84	968.83	5982.35	0.22	2778.55	12830.03	0.21
$(\mathrm{E} \rightarrow {}_{\mathrm{U}}\mathrm{RU} + {}_{\mathrm{B}}\mathrm{RU}) + \mathrm{AR}$	372.80	2155.36	0.11	**208.88**	**1141.56**	**0.08**	1380.99	16370.41	3.66
$(\mathrm{E} \rightarrow {}_{\mathrm{U}}\mathrm{RU} + {}_{\mathrm{U}}\mathrm{RU}) + \mathrm{AR}$	1472.05	16491.81	3.72	4052.71	31384.80	11.30	1713.72	18531.68	3.81
$(\mathrm{E} \rightarrow {}_{\mathrm{U}}\mathrm{RU}) + \mathrm{AR}$	1350.35	16673.11	3.69	2852.28	25869.18	7.60	260.66	1513.92	0.10
${}_{\mathrm{U}}$RU	1175.99	6716.42	2.08	1825.66	20400.04	4.95	3723.88	21022.60	2.83
Recurrent Graph Evolution Neural Network (ReGENN)							**165.41**	**915.92**	**0.05**
ReGENN without Variable Decoder							197.51	1141.04	0.15
ReGENN without Autoregression							8208.53	50089.84	19.19
Autoregression Only							11529.79	76118.62	9.15
Overall Improvement							20.81%	19.77%	35.72%

The experiment varies between Recurrent Unit (RU) and their directional flag, differing between Bidirectional (B) and Unidirectional (U). We use E to designate the non-conventional Transformer Encoder and AR the Autoregression component along with the table.

- ◉ Consists of multi-output and multi-task algorithms – v estimators, one estimator per variable;
- ⊙ Indicates single-target algorithms – $v \times z$ estimators, one estimator per variable and stride; and,
- ○ Represents univariate algorithms – $s \times v$ estimators, one estimator for each sample and variable.

3.1.4 Evaluation Metrics

As time-series forecasting works as a time-aware regression problem, our goal remains in predicting values that resemble the ground truth the most. As such, we used three evaluation metrics, the Mean Squared Logarithmic Error (MSLE), Mean Absolute Error (MAE), and Root Mean Squared Error (RMSE), that, despite different, have a complementary meaning. The MSLE uses the logarithmic scale to assess how well the model can predict values close to zero. Contrarily, the MAE uses the absolute operator to explain how the model fared among the median values. In another perspective, the RMSE uses the square root to assess the model's ability to predict the larger values, which might be outliers. The metrics' formal definition is as follows:

$$MSLE = \frac{1}{n} \sum_{i=1}^{n} log\left(\frac{\widehat{\mathbf{Y}}_i + 1}{\widehat{\mathbf{T}}_i + 1}\right)^2. \tag{7}$$

$$MAE = \frac{1}{n} \sum_{i=1}^{n} \left|\widehat{\mathbf{Y}}_i - \widehat{\mathbf{T}}_i\right| \tag{8}$$

$$RMSE = \sqrt{\frac{1}{n} \sum_{i=1}^{n} \left(\widehat{\mathbf{Y}}_i - \widehat{\mathbf{T}}_i\right)^2} \tag{9}$$

3.1.5 Hyperparameter Tuning

ReGENN has two hyperparameters able to change the dimension of the weights' tensors, which are the window size (i.e., input size) and the stride size (i.e., output size). As already discussed, both were set before the experiments, and none of them were tuned towards any performance improvement. The trade-off of having fewer hyperparameters is to spend more energy on training the network towards a better performance. We are focusing on the network optimizer, gradient clipping, learning rate scheduler, and dropout regularization when we refer to tunable hyperparameters. Along these lines, we followed a controlled and limited exploratory approach similar to a random grid-search, starting with PyTorch's defaults. The tuning process was on the validation set, intentionally reserved for measuring the network improvement. The tuning process follows by updating the hyperparameters whenever observing better results on the validation set, leading us to a set of optimized but no optimum hyperparameters. We used the set of optimized hyperparameters to evaluate ReGENN on the test set and the default values for all the other algorithms [49], [58], [59] unless explicitly required for working with a particular dataset, as was the case of LSTNet [39], DSANet [40], and MLCNN [42]. The complete list of hyperparameters is in the Supplementary Material, available online.

3.1.6 Computer Environment

The experiments related to machine-learning and time-series algorithms were carried out on a Linux-based system with 64 CPUs and 750 GB of RAM. The experiments related to deep-learning on the SARS-CoV-2 dataset were carried out on a Linux-based system with 56 CPUs, 256 GB of RAM, and 8 GPUs (Titan X – Pascal). The Brazilian Weather and PhysioNet datasets were tested on a different system with 32 CPUs, 512 GB of RAM, and 8 GPUs (Titan X – Maxwell). While CPU-based experiments are even across all CPU architectures, the same does not hold for GPUs, such that the GPU model and architecture must match to guarantee reproducibility. Aiming at complete reproducibility, we disclose not only the source code of ReGENN on GitHub,[2] but also the scripts, preprocessed data, and snapshots of all trained networks on a public folder at Google Drive.[3]

2. Available at https://bit.ly/2YDBrOo.
3. Available at https://bit.ly/30csLiJ.

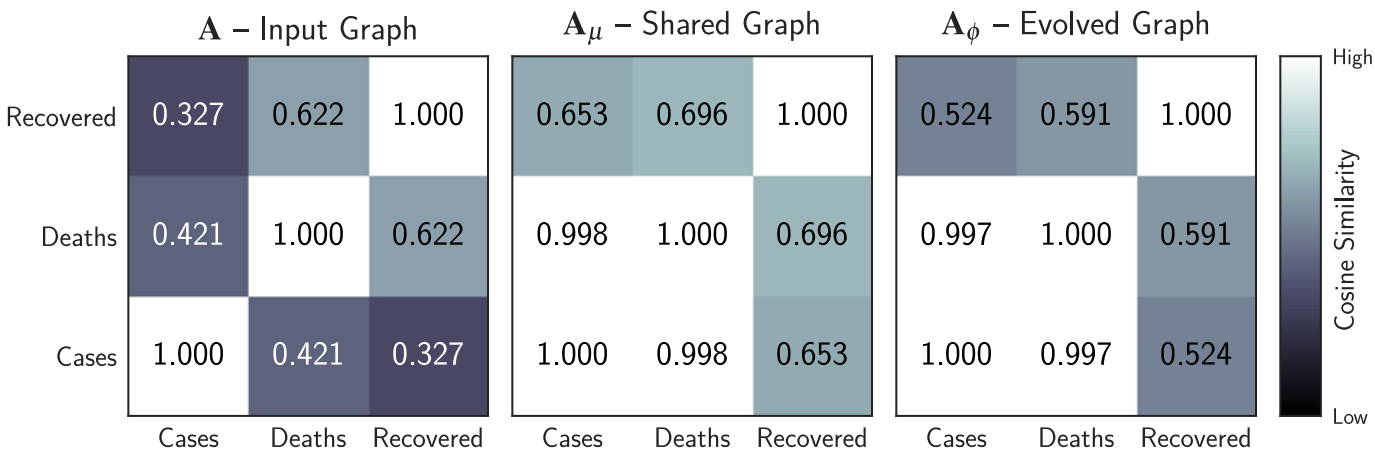


Fig. 6. Set of *Evolution Weights*, i.e., cosine-similarity activated hidden weights extracted from REGENN at the end of the network training. The images compare the cosine similarity between the set of variables within the SARS-CoV-2 dataset.

3.2 Results

Subsequently, we go through the experimental results in each one of the benchmark datasets. Due to the fact that not all the algorithms perform evenly across them, we display the 34 most prominent ones out of the 49 tested algorithms; for the extended results, please refer to the Supplementary Material, available online. We also discuss the ablation experiments, which were carried out with REGENN's hyperparameters; in the Supplementary Material, available online, we provide two other rounds of this same experiment using as hyperparameters PYTORCH's defaults[4] and other settings recurrently employed in the literature. Additionally, we draw explanations about the *Evolution Weights*, i.e., intermediate adjacency matrices from the GSE layers, by using the cosine similarity on the adjacency matrices of co-occurring variables (see Section 2.2).

3.2.1 Experiment on SARS-CoV-2 Dataset

The SARS-CoV-2 has being updated daily since the beginning of the Coronavirus Pandemic. We used a self-to-self transfer-learning approach to train the network in slices of time due to such a dataset's streaming nature. In short, the network was re-trained every 15 days with new incoming data, using as starting weights, the pre-trained weights from the network trained in the past 15 days so to evaluate REGENN's performance over the pandemic's progression.

In such a case, when the pandemic completed 45 days, the first time-slice in which REGENN was trained, it outperformed the second-placed algorithm, the Orthogonal Matching Pursuit, by 27.27 percent on the Mean Absolute Error (MAE), 16.50 percent on the Root Mean Square Error (RMSE), and 38.87 percent on the Mean Squared Logarithmic Error (MSLE). For all subsequent time-slices (i.e., 60, 75, 90, 105, and 120 days), the second-placed algorithm was the Exponential Smoothing, which was outperformed with improvement no lower than 47.40 percent on the MAE, 17.19 percent on the RMSE, and 37.39 percent on the MSLE. We further detail the results on the time-slices mentioned above in the Supplementary Material, available online. However, in Fig. 5, we detail the results on the complete dataset of 120 days, in which REGENN surpassed the Exponential Smoothing, the second-best algorithm, by 75.21 percent on the MAE, 64.87 percent on the RMSE, and 79.61 percent on the MSLE.

As a result of the analysis of the dataset in time-slices, we noticed that, as time goes by and more information is available on the SARS-CoV-2 dataset, the problem becomes more challenging to solve by looking individually at each country and more natural when looking at all of them together. Although countries have their particularities, which make the disease spread in different ways, the main goal is to decrease the spreading, such that similarities between the historical data of different countries provide for finer predictions. Furthermore, we observed that not all the estimators within an ensemble perform in the same way in the face of different countries. Due to the REGENN capability of observing inter- and intra-relationships between time-series, it performs better on highly uncertain cases like this one.

Subsequently, we present the ablation results, in which we utilized the same data-flow as REGENN but no GSE layer while systematically changing the decoder architecture. We provide results using different Recurrent Units (RU), including the Elman RNN [32], LSTM [43], and GRU [60]. We also varied the recurrent unit's directional flag between Unidirectional (U) and Bidirectional (B). That because a unidirectional recurrent unit tracks only forward dependencies while a bidirectional one tracks both forward and backward dependencies. A summarized tag describes each test's network architecture; for example, $(\mathrm{E} \rightarrow {}_{\mathrm{U}}\mathrm{RU} + {}_{\mathrm{B}}\mathrm{RU}) + \mathrm{AR}$ means the model has a Transformer Encoder (E) as the encoder, a Unidirectional Recurrent Unit as the time-axis decoder, and a Bidirectional Recurrent Unit as the variable-axis decoder. Besides that, the output of the decoder is element-wise added to the Autoregression (AR) output. The table shows results with and without the encoder and AR component, besides using a single recurrent unit only for time-axis decoding.

According to the ablation results in Table 2, the improvement of REGENN is slightly smaller than previously reported. That is because its performance not only comes from the GSE layer but also from how the network handles the multiple multivariate time-series data. Consequently, the ablation experiments reveal that some models without GSE layers are

4. Available at https://bit.ly/2QuWRsD.

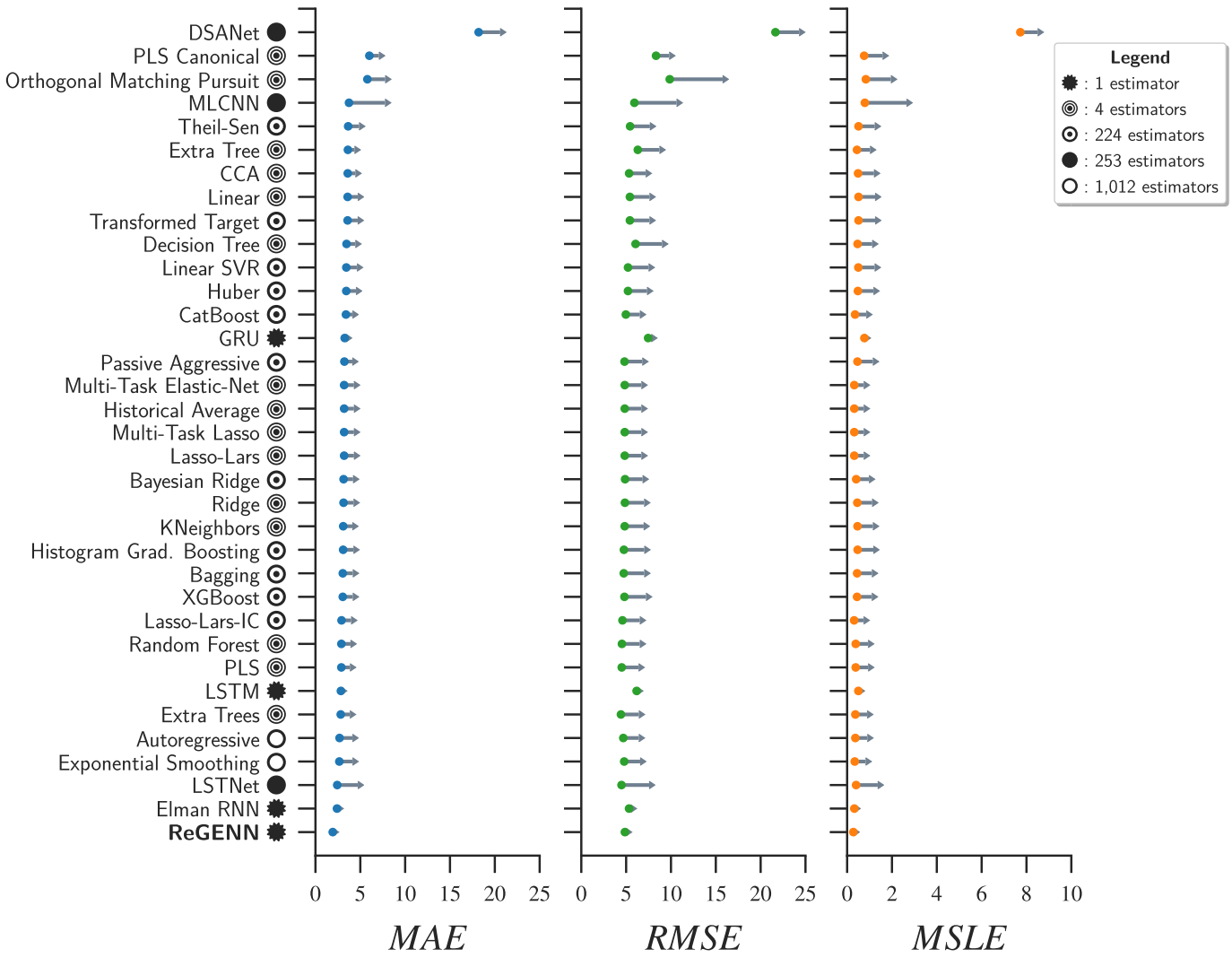


Fig. 7. Baseline results for the Brazilian Weather dataset presented in descending order of MAE. In this experiment, ReGENN once more outperformed all the competing algorithms, demonstrating versatility by performing well even on a highly-seasonal dataset with improvement no lower than 11.95 percent. In the face of seasonality, the Elman RNN surpassed the Exponential Smoothing, the previously second-best algorithm. In the image, the algorithms are symbol-encoded based on their type and number of estimators; we use gray arrows to report the results' standard deviation. The negative deviation, which is equal to the positive one, was suppressed for improved readability.

enough to surpass all the competing algorithms. However, when using ReGENN, we can improve them further and achieve 20.81 percent of additional reduction on the MAE, 19.77 percent on the RMSE, and 35.72 percent on the MSLE.

Fig. 6 shows the *Evolution Weights* originated from applying the cosine similarity on the hidden adjacency matrices of ReGENN. When comparing the input and evolved graphs, the number of cases and deaths have a mild similarity. That might come from the fact that diagnosing infected people was already a broad concern at the beginning of the pandemic. The problem did not go away, but more infected people were discovered as more tests were made, and also because the disease spread worldwide. A similar scenario can be drawn from the number of recovered people and the number of cases, as infected people with mild or no symptoms were unaware of being infected. Contrarily, we can see that the similarity between recovered and deaths decreases over time, which comes from the fact that, as more tests are made, the mortality rate drops to a stable threshold due to the increased number of recovered people.

3.2.2 Experiment on Brazilian Weather Dataset

The Brazilian Weather dataset is a highly seasonal dataset with a higher number of samples, variables, and timestamps than the previous one. For simplicity's sake, in this experiment, ReGENN was trained on the whole training set at once. The results are in Fig. 7, in which ReGENN was the first-placed algorithm, followed by the Elman RNN in second. ReGENN overcame the Elman RNN by 11.95 percent on the MAE, 11.96 percent on the RMSE, and 25.84 percent on the MSLE.

We noticed that all the algorithms perform quite similarly for this dataset. The major downside for most algorithms comes from predicting small values close to zero, as noted by the MSLE results. In such a case, the ensembles showed a high variance when compared to ReGENN. We believe this is why the Elman RNN shows performance closer to ReGENN rather than to Exponential Smoothing, the third-placed algorithm, as ReGENN has a single estimator, while the Exponential Smoothing is an ensemble of estimators. Another understanding of why some algorithms

TABLE 3
Ablation Results From Experimenting Over the Brazilian Weather Dataset That was Conducted

Recurrent Unit (RU)	Elman RNN			GRU			LSTM		
Network Architecture	MAE	RMSE	MSLE	MAE	RMSE	MSLE	MAE	RMSE	MSLE
${}_{\mathrm{B}}$RU	2.56	5.90	0.45	2.19	5.00	0.28	**2.20**	**5.04**	**0.28**
$\mathrm{E} \rightarrow {}_{\mathrm{B}}$RU	2.70	5.66	0.37	2.33	5.19	0.28	2.57	5.42	0.29
$(\mathrm{E} \rightarrow {}_{\mathrm{B}}\mathrm{RU} + {}_{\mathrm{B}}\mathrm{RU}) + \mathrm{AR}$	**2.18**	**5.52**	**0.37**	2.38	5.72	0.41	2.44	5.75	0.41
$(\mathrm{E} \rightarrow {}_{\mathrm{B}}\mathrm{RU} + {}_{\mathrm{U}}\mathrm{RU}) + \mathrm{AR}$	2.84	6.47	0.55	2.24	5.05	0.28	3.17	7.42	0.80
$(\mathrm{E} \rightarrow {}_{\mathrm{B}}\mathrm{RU}) + \mathrm{AR}$	3.58	8.13	0.98	3.31	7.57	0.82	3.61	8.19	0.96
$\mathrm{E} \rightarrow {}_{\mathrm{U}}$RU	3.12	5.86	0.32	2.81	5.69	0.34	2.96	5.54	0.30
$(\mathrm{E} \rightarrow {}_{\mathrm{U}}\mathrm{RU} + {}_{\mathrm{B}}\mathrm{RU}) + \mathrm{AR}$	2.36	5.69	0.41	2.23	5.11	0.28	2.47	5.77	0.41
$(\mathrm{E} \rightarrow {}_{\mathrm{U}}\mathrm{RU} + {}_{\mathrm{U}}\mathrm{RU}) + \mathrm{AR}$	2.52	5.83	0.42	**2.10**	**4.97**	**0.28**	4.88	10.24	1.61
$(\mathrm{E} \rightarrow {}_{\mathrm{U}}\mathrm{RU}) + \mathrm{AR}$	4.30	9.38	1.34	3.25	7.55	0.81	5.25	10.69	1.75
${}_{\mathrm{U}}$RU	2.40	5.32	0.32	3.26	7.45	0.77	2.83	6.17	0.50
Recurrent Graph Evolution Neural Network (ReGENN)							**1.92**	**4.86**	**0.28**
ReGENN without Variable Decoder							1.95	4.85	0.27
ReGENN without Autoregression							2.00	4.95	0.27
Autoregression Only							2.69	4.68	0.37
Overall Improvement							8.42%	2.16%	0.00%

We use E to designate the non-conventional Transformer Encoder and AR the feed-forward Autoregressive linear component. In the experiment, we vary the Recurrent Unit (RU) and its Directional Flag Between Bidirectional (B) and Unidirectional (U).

underperform on the MSLE is related to their difficulty to track temporal dependencies, such as weather seasonality.

The ablation results are in Table 3, in which we observed again that the network without the GSE layers already surpasses the baselines. When decommissioning the GSE layers of ReGENN and using GRU instead of LSTM on the decoder, we observed a 3.86 percent improvement on the MAE, 10.02 percent on the RMSE, and 25.34 percent on the MSLE when compared to the Elman RNN results. Using ReGENN instead, we achieve a further performance gain of 8.42 percent on the MAE and 2.16 percent on the RMSE over the ablation experiment.

Fig. 8 depicts the *Evolution Weights* for the current dataset, in which we can observe a consistent similarity between pairs of variables in the input graph, which does not repeat in the evolved graph, implying different relationships. We observe that the similarity between all pairs of variables increased on the evolved graph. The pairs *Solar Radiation* and *Rain*, *Maximum Temperature* and *Rain*, and *Solar Radiation* and *Minimum Temperature* stood out. Those pairs are mutually related, which comes from solar radiation interfering in maximum and minimum temperature and in the precipitation factors, where the opposite relation holds. What can be extracted from the *Evolution Weights*, in this case, is the notion of importance between pairs of variables so that the pairs that stood out are more relevant and provide better information during the forecasting process.

3.2.3 Experiment on PhysioNet Dataset

The PhysioNet dataset presents a large number of samples and an increased number of variables, but little information on the time-axis, a setting in which ensembles still struggle to perform accurate predictions, as depicted in Fig. 9. Once again, ReGENN keeps steady as the first-placed algorithm in performance, showing solid improvement over the Linear SVR, the second-placed algorithm. The improvement was 7.33 percent on the MAE and 35.13 percent on the

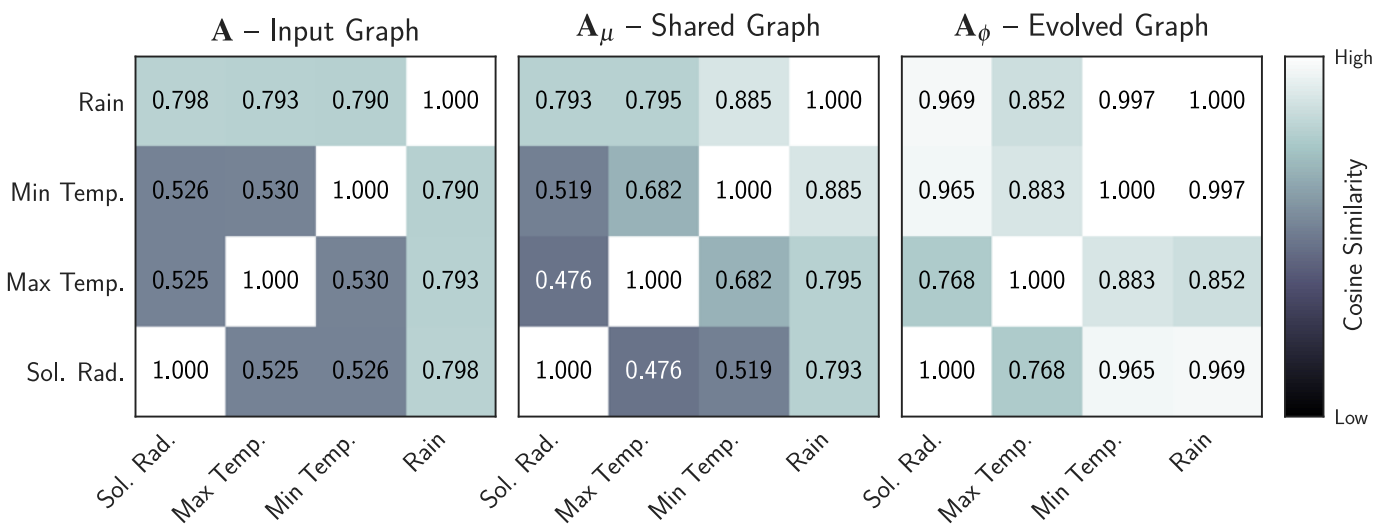


Fig. 8. The *Evolution Weights* from ReGENN for the Brazilian Weather dataset, in which we use the cosine similarity activation function on the network's hidden weights to compare the relationship between pairs of variables. The image uses "Sol. Rad." as a shortening for Solar Radiation, "Temp" for Temperature, "Max" for Maximum, and "Min" for Minimum.

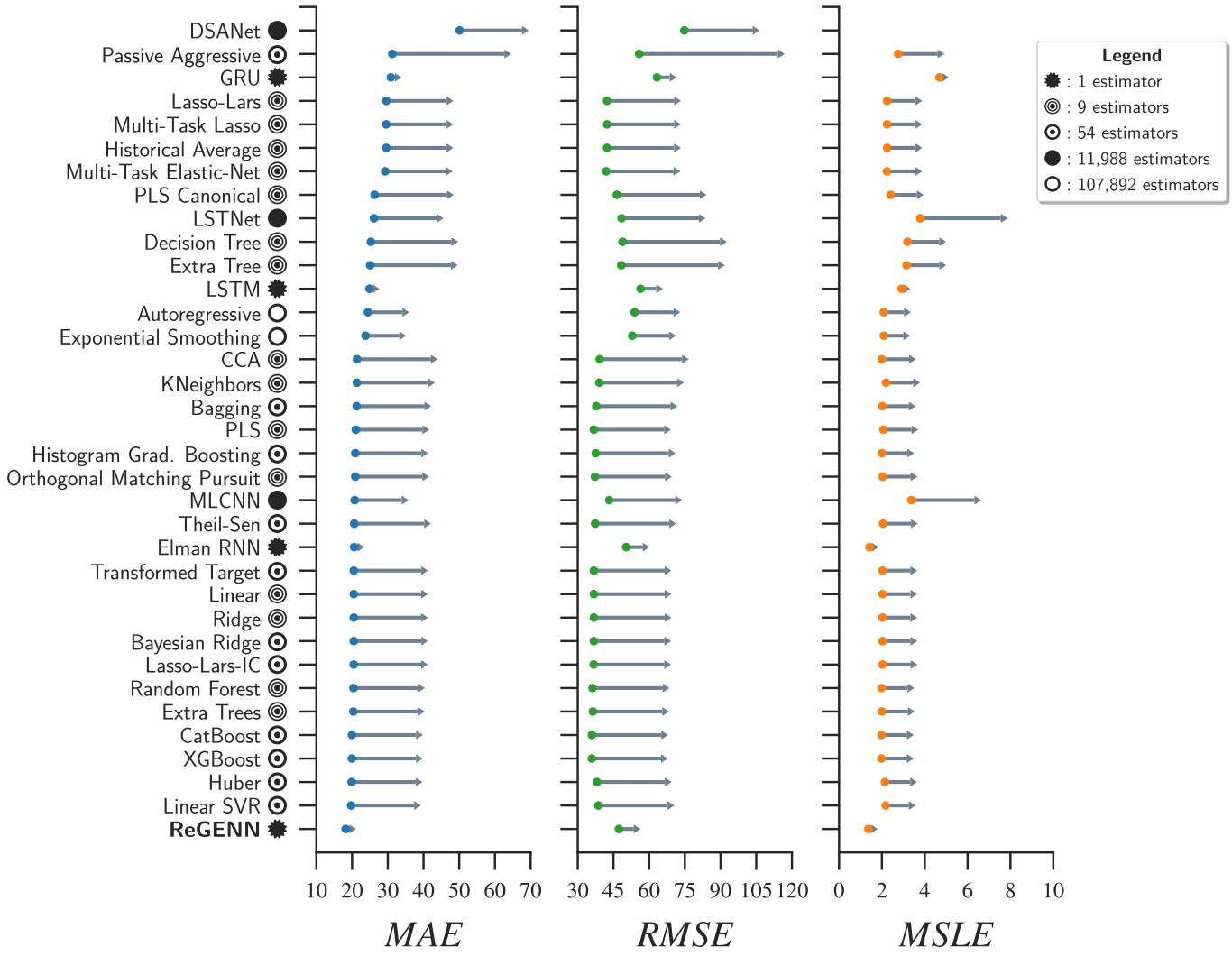


Fig. 9. Baseline results for the PhysioNet Computing in Cardiology Challenge 2012 dataset presented in descending order of MAE, in which ReGENN was the algorithm with the best performance followed by the Linear SVR. The comparative improvement was no lower than 7.33 percent, but, in this case, ReGENN yielded an RMSE compatible with the Linear SVR. In the image, the algorithms are symbol-encoded based on their type and number of estimators; we use gray arrows to report standard deviation. The negative deviation, which is equal to the positive one, was suppressed for improved readability.

TABLE 4
Ablation Results Over the PhysioNet Computing in Cardiology Challenge 2012 Dataset

Recurrent Unit (RU)	Elman RNN			GRU			LSTM		
Network Architecture	MAE	RMSE	MSLE	MAE	RMSE	MSLE	MAE	RMSE	MSLE
${}_{\mathrm{B}}$RU	19.95	50.47	1.38	24.80	56.42	3.01	19.49	50.46	1.40
$\mathrm{E} \rightarrow {}_{\mathrm{B}}\mathrm{RU}$	19.11	49.16	1.41	24.17	54.81	3.00	18.88	48.55	1.40
$(\mathrm{E} \rightarrow {}_{\mathrm{B}}\mathrm{RU} + {}_{\mathrm{B}}\mathrm{RU}) + \mathrm{AR}$	**18.72**	**48.60**	**1.37**	18.55	47.86	1.39	**18.42**	**47.78**	**1.37**
$(\mathrm{E} \rightarrow {}_{\mathrm{B}}\mathrm{RU} + {}_{\mathrm{U}}\mathrm{RU}) + \mathrm{AR}$	18.97	49.59	1.38	**18.54**	**48.90**	**1.37**	18.46	48.33	1.36
$(\mathrm{E} \rightarrow {}_{\mathrm{B}}\mathrm{RU}) + \mathrm{AR}$	18.81	50.20	1.38	18.67	48.59	1.39	18.54	47.35	1.38
$\mathrm{E} \rightarrow {}_{\mathrm{U}}\mathrm{RU}$	19.46	50.24	1.44	19.42	51.03	1.44	19.78	51.29	1.44
$(\mathrm{E} \rightarrow {}_{\mathrm{U}}\mathrm{RU} + {}_{\mathrm{B}}\mathrm{RU}) + \mathrm{AR}$	18.76	48.81	1.39	18.65	48.64	1.38	18.55	48.28	1.38
$(\mathrm{E} \rightarrow {}_{\mathrm{U}}\mathrm{RU} + {}_{\mathrm{U}}\mathrm{RU}) + \mathrm{AR}$	19.02	49.75	1.43	18.64	48.81	1.38	18.63	48.93	1.37
$(\mathrm{E} \rightarrow {}_{\mathrm{U}}\mathrm{RU}) + \mathrm{AR}$	18.98	48.67	1.45	18.82	49.98	1.41	18.88	50.31	1.39
${}_{\mathrm{U}}$RU	20.58	50.32	1.42	30.85	63.30	4.68	24.85	56.41	2.92
Recurrent Graph Evolution Neural Network (ReGENN)							**18.22**	**47.31**	**1.37**
ReGENN without Variable Decoder							18.23	47.45	1.37
ReGENN without Autoregression							18.48	49.72	1.37
Autoregression Only							24.44	53.92	2.08
Overall Improvement							1.05%	0.98%	0.0%

We use E as a shortening for the Transformer Encoder and AR for the Autoregressive linear component. In the experiment, the Recurrent Unit (RU) is Changed Together With its Directional Flag, Varying Between Bidirectional (B) and Unidirectional (U).

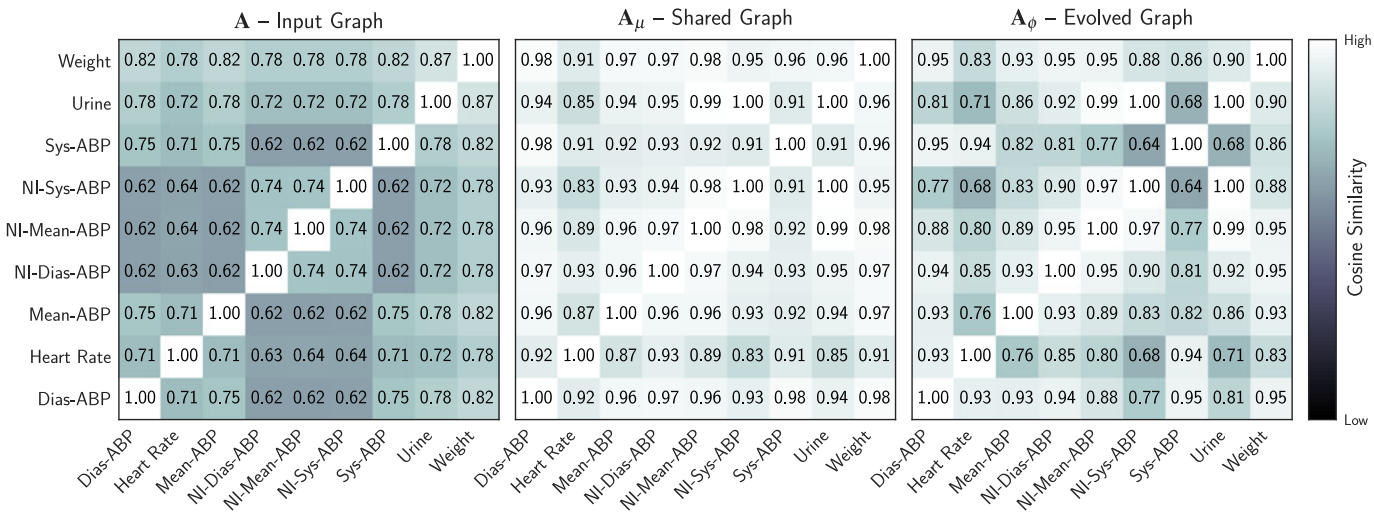


Fig. 10. *Evolution Weights* extracted from REGENN after training on the PhysioNet Computing in Cardiology Challenge 2012 dataset, in which we use the cosine similarity to compare the relationship between pairs of variables. We use "ABP" as a shortening for *Arterial Blood Pressure*, "NI" as *Non-Invasive*, "Dias" as *Diastolic*, and "Sys" as *Systolic*.

MSLE, while the RMSE achieved by REGENN laid within the standard deviation of the Linear SVR, pointing out an equivalent performance between them. The Linear SVR is an ensemble of multiple estimators, while REGENN uses only one, making it better accurate for dealing with the current dataset.

As in Table 4, the ablation results reveal that a neural network architecture without the GSE layers can achieve a better performance than the baseline algorithms. In this specific case, we see that by using a bidirectional LSTM instead of unidirectional on the decoder module of the neural network, we can achieve a performance almost as good as REGENN, but not enough to surpass it, as REGENN still shows an improvement of 1.05 percent on the MAE and 0.98 percent on the RMSE over the experiment with bidirectional LSTM.

In this specific case, REGENN learns by observing multiple ICU patients. However, one cannot say that an ICU patient's state is somehow connected to another patient's state. Contrarily, the idea holds as in the first experiment, where although the samples are different, they have the same domain, variables, and timestream, such that the information from one sample might help enhance future forecasting for another one. That means REGENN learns both from the past of the patient and from the past of other patients. Nevertheless, we must be careful about drawing any understanding about these results, as the reason each patient is in the ICU is different, and while some explanations might be suited for a small set of patients, it tends not to generalize to a significant number of patients. When analyzing the *Evolution Weights* in Fig. 10 aided by a physician, we can say that there is a relationship between the amount of urine excreted by a patient and the arterial blood pressure, and also that there is a relation between the systolic and diastolic blood pressure. However, even aided by the *Evolution Weights*, we cannot further describe these relations once there are variables of the biological domain that are not being taken into consideration.

4 Overall Discussions

We refer to the *Evolution Weights* as the intermediate weights of the representation-learning process optimized throughout the network's training. Such weights are time-invariant and are a requirement for the feature-learning behind the GSE layer. Although time does not flow through the adjacency matrix, the network is optimized as a whole, such that every operation influences the gradients of the backward propagation process. That means the optimizer, influenced by the gradients of both time-variant and invariant data, will optimize the weights towards a better forecasting ability. Such a process depends not only on the network architecture but also on the optimization process's reliability.

That increases uncertainty, which is the downside of REGENN, demanding more time to train the neural network and causing the improvement not to be strictly uprising. Consequently, training might take long sessions, even with consistently reduced learning rates on plateaus or simulated annealing techniques; this is influenced by the fact that the second GSE layer has two dynamic inputs, which arise from the graph-evolution process. However, we observed that throughout the epochs, the *Evolution Weights* reach a stable point with no major updates. As a result, the network demonstrates a remarkable improvement in its final iterations when the remaining weights intensely converge to a near-optimal configuration.

Even though REGENN has a particular drawback, it demonstrates excellent versatility, which comes from its superior performance in the task of epidemiology modeling on the SARS-CoV-2 dataset, climate forecasting on the Brazilian Weather, and patient monitoring on intensive care units on the PhysioNet dataset. Consequently, we see REGENN as a tool to be used in data-driven decision-making tasks, helping prevent, for instance, natural disasters or preparing for an upcoming pandemic. As a precursor in multiple multivariate time-series forecasting, there is still much to be improved. For example, reducing the uncertainty that harms REGENN without decreasing its performance should be the first step, followed

by extending the proposal to handle problems in the spatio-temporal field of great interest to traffic forecasting and environmental monitoring. Another possibility would be to remove the decoder's recurrent layers while tracking the temporal dependencies through multiple graphs, a new temporal-modeling perspective in which one could leverage from Graph Convolution Networks for extracting inter-variable relationships and using those as hidden-features during the time-series forecasting process, to which further hypothesis constraints may apply.

Notwithstanding, in some cases, where extensive generalization is not required, the analysis of singular multivariate time-series may be preferred to multiple multivariate time-series. That because, when focusing on a single series at a time, some but not all samples might yield a lower forecasting error, as the model will be driven to a single multivariate sample. However, both approaches for tackling time-series forecasting can coexist in the state-of-the-art, and, as a consequence, the decision to work on a higher or lower dimensionality must relate to which problem is being solved and how much data is available to solve it.

5 Conclusion

This paper tackles multiple multivariate time-series forecasting tasks by proposing the Recurrent Graph Evolution Neural Network (ReGENN), a graph-based time-aware auto-encoder powered by a pair of Graph Soft Evolution (GSE) layers, a further contribution of this study that stands for a graph-based learning-representation layer.

The literature handles multivariate time-series forecasting with outstanding performance, but up to this point, we lacked a technique with increased generalization over multiple multivariate time-series with sound performance. Previous research might have avoided tackling such a problem as a neural network, to that matter, is challenging to train and usually yields poor results. That because one aims to achieve good generalization on future observations for multivariate time-series that do not necessarily hold the same data distribution.

Because of that, ReGENN is a precursor in multiple multivariate time-series forecasting and, even though this is a challenging problem, ReGENN surpassed all the baselines and remained effective after three rounds of 30 ablation tests through distinct hyperparameters. The experiments were carried out over the SARS-CoV-2, Brazilian Weather, and PhysioNet datasets with improvements, respectively, of at least 64.87, 11.96, and 7.33 percent. As a consequence of the results, ReGENN shows a new range of possibilities in time-series forecasting, starting by demonstrating that ensembles poorly perform if compared to a single model able to learn the entanglement between different variables by looking at how they interact as time goes by and how multiple multivariate time-series synchronously evolve.

Authors Contributions Statement

GS conceived the experiments, GS prepared the data for experimentation, and GS conducted them. SH and BB validated GS's source-code implementation. SH and BB analyzed the preliminary results. SM, JR, and JS validated the final results. GS wrote the original draft. SH, BB, SM, JR, and JS iteratively polished the main ideas and the paper. Shenda Hong and Bruno Brandoli contributed equally to this work. All authors reviewed and approved the results and manuscript.

Acknowledgments

This work was supported in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior – Brazil (CAPES) – Finance Code 001, in part by the Fundação de Amparo à Pesquisa do Estado de São Paulo (FAPESP) under Grants 2014/25337-0, 2016/17078-0, 2017/08376-0, 2018/17620-5, 2019/04461-9, and 2020/07200-9, in part by the Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq) under Grants 167967/2017-7, 305580/2017-5, and 406550/2018-2, in part by the National Science Foundation awards IIS-2014438, PPoSS-2028839, and IIS-1838042, and in part by the National Institute of Health Awards NIH R01 1R01NS107291-01 and R56HL138415. The authors would like to thank Jeffrey Valdez for his aid with Sunlab's computer infrastructure, Lucas Scabora for his careful review of the paper, and Gustavo Merchan, M.D., for his input on the *Evolution Weights* of the PhysioNet dataset.

References

[1] S. L. Lavender *et al.*, "Estimation of the maximum annual number of North Atlantic tropical cyclones using climate models," *Sci. Adv.*, vol. 4, no. 8, Aug. 2018, Art. no. eaat6509.

[2] Y.-C. Kao *et al.*, "Effects of climate and land-use changes on fish catches across lakes at a global scale," *Nature Commun.*, vol. 11, no. 1, pp. 25–26, Dec. 2020.

[3] S. C. Pryor, R. J. Barthelmie, M. S. Bukovsky, L. R. Leung, and K. Sakaguchi, "Climate change impacts on wind power generation," *Nature Rev. Earth Environ.*, vol. 1, no. 12, pp. 627–643, Dec. 2020.

[4] L. Laurent *et al.*, "The impact of climate change and glacier mass loss on the hydrology in the Mont-Blanc massif," *Sci. Rep.*, vol. 10, no. 1, Dec. 2020, Art. no. 10420.

[5] A. Kelotra and P. Pandey, "Stock market prediction using optimized deep-convLSTM model," *Big Data*, vol. 8, no. 1, pp. 5–24, Feb. 2020.

[6] M. Ananthi and K. Vijayakumar, "Stock market analysis using candlestick regression and market trend prediction (CKRM)," *J. Ambient Intell. Human Comput.*, 2020. [Online]. Available: https://doi.org/10.1007/s12652-020-01892-5

[7] S. Jiao, T. Shen, Z. Yu, and H. Ombao, "Change-point detection using spectral PCA for multivariate time series," Jan. 2021, *arXiv:2101.04334*.

[8] D. Roy and L. Yan, "Robust Landsat-based crop time series modelling," *Remote Sens. Environ.*, vol. 238, Mar. 2020, Art. no. 110810.

[9] Z. Zhu *et al.*, "Continuous monitoring of land disturbance based on Landsat time series," *Remote Sens. Environ.*, vol. 238, pp. 111–116, Mar. 2020.

[10] L. Yan and D. P. Roy, "Spatially and temporally complete Landsat reflectance time series modelling: The fill-and-fit approach," *Remote Sens. Environ.*, vol. 241, May 2020, Art. no. 111718.

[11] N. Laptev, S. Amizadeh, and I. Flint, "Generic and scalable framework for automated time-series anomaly detection," in *Proc. ACM SIGKDD Int. Conf. Knowl. Discov. Data Mining*, Aug. 2015, pp. 1939–1947.

[12] J. Li, W. Pedrycz, and I. Jamal, "Multivariate time series anomaly detection: A framework of hidden Markov models," *Appl. Soft Comput.*, vol. 60, pp. 229–240, Nov. 2017.

[13] J. Hancock and T. M. Khoshgoftaar, "Performance of CatBoost and XGBoost in medicare fraud detection," in *Proc. 19th IEEE Int. Conf. Mach. Learn. Appl.*, Dec. 2020, pp. 572–579.

[14] C. Deb, F. Zhang, J. Yang, S. E. Lee, and K. W. Shah, "A review on time series forecasting techniques for building energy consumption," *Renewable Sustain. Energy Rev.*, vol. 74, pp. 902–924, Jul. 2017.

[15] A. Orlov, J. Sillmann, and I. Vigo, "Better seasonal forecasts for the renewable energy industry," *Nat. Energy*, vol. 5, no. 2, pp. 108–110, Feb. 2020.

[16] S. G. Baratsas *et al.*, "A framework to predict the price of energy for the end-users with applications to monetary and energy policies," *Nature Commun.*, vol. 12, no. 1, Dec 2021, Art. no. 18.

[17] I. Perros, E. E. Papalexakis, R. Vuduc, E. Searles, and J. Sun, "Temporal phenotyping of medically complex children via PARAFAC2 tensor factorization," *J. Biomed. Inform.*, vol. 93, pp. 103–125, May 2019.

[18] J. F. Rodrigues-Jr, G. Spadon, B. Brandoli, and S. Amer-Yahia, "Patient trajectory prediction in the Mimic-III dataset, challenges and pitfalls," 2019, *arXiv:1909.04605*.

[19] J. F. Rodrigues, G. Spadon, B. Brandoli, and S. Amer-Yahia, "Lig-doctor: Real-world clinical prognosis using a bi-directional neural network," in *Proc. IEEE 33rd Int. Symp. Comput.-Based Med. Syst.*, 2020, pp. 569–572.

[20] A. Afshar *et al.*, "TASTE," in *Proc. ACM Conf. Health, Inference, Learn.*, Apr. 2020, pp. 193–203.

[21] L. Yan *et al.*, "An interpretable mortality prediction model for COVID-19 patients," *Nat. Mach. Intell.*, vol. 2, no. 5, pp. 283–288, May 2020.

[22] J. F. Rodrigues-Jr, M. A. Gutierrez, G. Spadon, B. Brandoli, and S. Amer-Yahia, "LIG-Doctor: Efficient patient trajectory prediction using bidirectional minimal gated-recurrent networks," *Inf. Sci.*, vol. 545, pp. 813–827, Feb. 2021.

[23] M. Ienca and E. Vayena, "On the responsible use of digital data to tackle the COVID-19 pandemic," *Nat. Med.*, vol. 26, no. 4, pp. 463–464, Apr. 2020.

[24] T. P. Velavan and C. G. Meyer, "The COVID-19 epidemic," *Trop. Med. Int. Health*, vol. 25, no. 3, pp. 278–280, Mar. 2020.

[25] S. B. Omer, P. Malani, and C. del Rio, "The COVID-19 Pandemic in the US," *JAMA*, vol. 323, no. 18, pp. 1767–1768, Apr. 2020.

[26] A. Silvestrini and D. Veredas, "Temporal aggregation of univariate and multivariate time series models: A survey," *J. Econ. Surv.*, vol. 22, no. 3, pp. 458–497, Jul. 2008.

[27] A. Jain, Jianchang Mao, and K. Mohiuddin, "Artificial neural networks: A tutorial," *Computer*, vol. 29, no. 3, pp. 31–44, Mar. 1996.

[28] G. Zhang, B. Patuwo, and M. Y. Hu, "A simulation study of artificial neural networks for nonlinear time-series forecasting," *Comput. Operations Res.*, vol. 28, no. 4, pp. 381–396, Apr. 2001.

[29] H. Zhou *et al.*, "Informer: Beyond efficient transformer for long sequence time-series forecasting," Dec. 2020, *arXiv:2012.07436*.

[30] Y. Seo, M. Defferrard, P. Vandergheynst, and X. Bresson, "Structured sequence modeling with graph convolutional recurrent networks," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, L. Cheng, A. C. S. Leung, and S. Ozawa, Eds. Berlin, Germany: Springer, 2018, pp. 362–373.

[31] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proc. IEEE*, vol. 86, no. 11, pp. 2278–2324, Nov. 1998.

[32] J. Elman, "Finding structure in time," *Cognitive Sci.*, vol. 14, no. 2, pp. 179–211, Jun. 1990.

[33] Y. Li, R. Yu, C. Shahabi, and Y. Liu, "Diffusion convolutional recurrent neural network: Data-driven traffic forecasting," in *Proc. Int. Conf. Learn. Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*, Jul. 2017. [Online]. Available: https://openreview.net/forum?id=SJiHXGWAZ

[34] J. Zhang, X. Shi, J. Xie, H. Ma, I. King, and D.-Y. Yeung, "GaAN: Gated attention networks for learning on large and spatiotemporal graphs," in *Proc. Conf. Uncertainty Art. Intell.*, Mar. 2018, pp. 339–349.

[35] Y. Liang, S. Ke, J. Zhang, X. Yi, and Y. Zheng, "GeoMAN: Multi-level attention networks for geo-sensory time series prediction," in *Proc. Int. Joint Conf. Artif. Intell.*, Jul 2018, pp. 3428–3434.

[36] B. Yu, H. Yin, and Z. Zhu, "Spatio-temporal graph convolutional networks: A deep learning framework for traffic forecasting," in *Proc. Int. Joint Conf. Artif. Intell.*, Sep. 2017, pp. 3634–3640.

[37] L. Zhao *et al.*, "T-GCN: A temporal graph convolutional network for traffic prediction," *IEEE Trans. Intell. Transportation Syst.*, vol. 21, no. 9, pp. 3848–3858, Sep. 2020.

[38] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," in *Proc. 5th Int. Conf. Learn. Representations*, 2017. [Online]. Available: https://openreview.net/forum?id=SJU4ayYgl

[39] G. Lai, W.-C. Chang, Y. Yang, and H. Liu, "Modeling long- and short-term temporal patterns with deep neural networks," in *Proc. 41st Int. ACM SIGIR Conf. Res. Develop. Inf. Retrieval*, Jun. 2018, pp. 95–104.

[40] S. Huang, X. Wu, D. Wang, and A. Tang, "DSANet: Dual self-attention network for multivariate time series forecasting," in *Proc. Int. Conf. Inf. Knowl. Manage.*, Nov. 2019, pp. 2129–2132.

[41] A. Vaswani *et al.*, "Attention is all you need," *Adv. Neural Inf. Process. Syst.*, vol. 30, pp. 5999–6009, Jun. 2017.

[42] J. Cheng, K. Huang, and Z. Zheng, "Towards better forecasting by fusing near and distant future visions," 2019, *arXiv:1912.05122*.

[43] S. Hochreiter and J. Schmidhuber, "Long Short-Term Memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, Nov. 1997.

[44] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *J. Mach. Learn. Res.*, vol. 15, no. 56, pp. 1929–1958, 2014.

[45] J. G. Zilly, R. K. Srivastava, J. Koutnik, and J. Schmidhuber, "Recurrent highway networks," in *Proc. 34th Int. Conf. Mach. Learn.*, Jul. 2017, pp. 6346–6357.

[46] R. K. Srivastava, K. Greff, and J. Schmidhuber, "Highway networks," 2015, *arXiv:1505.00387*.

[47] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Jun. 2016, pp. 770–778.

[48] J. L. Ba, J. R. Kiros, and G. E. Hinton, "Layer normalization," Jul. 2016, *arXiv:1607.06450*.

[49] T. Chen and C. Guestrin, "XGBoost: A scalable tree boosting system," in *Proc. ACM SIGKDD Int. Conf. Knowl. Discov. Data Mining*, Aug. 2016, pp. 785–794.

[50] D. P. Kingma and J. L. Ba, "Adam: A method for stochastic optimization," in *Proc. 3rd Int. Conf. Learn. Representations*, Y. Bengio and Y. LeCun, Eds., 2015. [Online]. Available: https://arxiv.org/abs/1412.6980

[51] V. Vapnik, S. E. Golowich, and A. Smola, "Support vector method for function approximation, regression estimation, and signal processing," in *Advances in Neural Information Processing Systems*, M. C. Mozer, M. I. Jordan, and T. Petsche, Eds. Cambridge, MA, USA: MIT Press, 1997, pp. 281–287.

[52] D. Sahoo, Q. Pham, J. Lu, and S. C. H. Hoi, "Online deep learning: Learning deep neural networks on the fly," in *Proc. Int. Joint Conf. Artif. Intell.*, Jul. 2018, pp. 2660–2666.

[53] E. Dong, H. Du, and L. Gardner, "An interactive web-based dashboard to track COVID-19 in real time," *Lancet Infectious Diseases*, vol. 20, no. 5, pp. 533–534, May 2020.

[54] I. Silva, G. Moody, D. J. Scott, L. A. Celi, and R. G. Mark, "Predicting in-hospital mortality of ICU patients: The PhysioNet/Computing in cardiology challenge 2012," in *Proc. Comput. Cardiology*, 2012, pp. 245–248.

[55] E. Keogh, S. Chu, D. Hart, and M. Pazzani, "Segmenting time series: A survey and novel approach," in *Data Mining in Time Series Databases*. Singapore: World Sci. Publishing Company, Jun. 2004, vol. 57, pp. 1–21.

[56] R. Frank, N. Davey, and S. Hunt, "Input window size and neural network predictors," in *Proc. IEEE-INNS-ENNS Int. Joint Conf. Neural Netw.*, 2000, pp. 237–242.

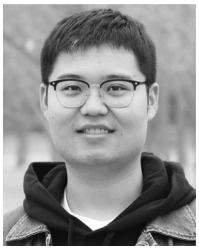
[57] R. J. Frank, N. Davey, and S. P. Hunt, "Time series prediction and neural networks," *J. Intell. Robot. Syst.: Theory Appl.*, vol. 31, no. 1–3, pp. 91–103, 2001.

[58] P. Fabian *et al.*, "Scikit-learn: Machine learning in Python," *J. Mach. Learn. Res.*, vol. 12, no. 85, pp. 2825–2830, 2011.

[59] A. V. Dorogush, V. Ershov, and A. Gulin, "CatBoost: Gradient boosting with categorical features support," in *Proc. Adv. Neural Inf. Processing Syst.*, S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, Eds., Oct. 2018, pp. 6638–6648.

[60] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio, "Empirical evaluation of gated recurrent neural networks on sequence modeling," 2014, *arXiv:1412.3555*.

Gabriel Spadon is currently working toward the PhD degree in computer science at the University of Sao Paulo, Brazil, part of which was carried out with the Georgia Institute of Technology, USA. He has worked intensively on network science and artificial intelligence during the last years. He has authored or coauthored several research articles on knowledge discovery through complex networks and data mining. His current research interests include neural-inspired models, graph-based learning, and complex networks.



Shenda Hong received the PhD degree from the School of Electronics Engineering and Computer Sciences, Peking University, in 2019. He is currently a (Boya) postdoctoral fellow at the National Institute of Health Data Science, Peking University. From 2019 to 2020, he was a postdoctoral fellow with the College of Computing, Georgia Institute of Technology, and a visiting researcher with Harvard Medical School. His research interests include machine learning methods for temporal sequences and time series, especially deep learning methods for electronic health records, and biomedical signals, including ECG and EEG.

Bruno Brandoli received the PhD degree (with honors) in computer science from the University of Sao Paulo, Brazil, in 2016. He is currently a faculty at the Computer Science Department and a researcher with the Institute for Big Data Analytics, Dalhousie University, Canada. Over the past years, he has authored or coauthored AI-related articles in relevant journals and conferences. His main research interests include deep learning-based models, computer vision, and data-driven learning. He is currently leading projects focused on the state-of-the-art AI models dedicated to marine biology.

Stan Matwin is currently the director of the Institute for Big Data Analytics, Dalhousie University, Halifax, Nova Scotia, where he is a professor and Canada Research Chair (Tier 1) in Interpretability for Machine Learning. He is also a distinguished professor (Emeritus) at the University of Ottawa and a full professor with the Institute of Computer Science, Polish Academy of Sciences. His main research interests include big data, text mining, machine learning, and data privacy. He is a member on the Editorial Boards of *IEEE Transactions on Knowledge and Data Engineering* and *Journal of Intelligent Information Systems*. He was the recipient of the Lifetime Achievement Award of the Canadian AI Association (CAIAC).

Jose F. Rodrigues-Jr received the PhD from the University of Sao Paulo, Brazil, part of which was carried out from Carnegie Mellon University, USA, in 2007. He is currently an associate professor at the University of Sao Paulo, Brazil. He is a regular reviewer and author in his research field, which include data science, machine learning, content-based data retrieval, visualization, and the application of such techniques in the medic, agriculture, and e-learning domains, contributing to publications in major journals and conferences.

Jimeng Sun is currently a professor at the Computer Science Department, University of Illinois, Urbana-Champaign. Previously, he was with the College of Computing, Georgia Institute of Technology. His research interest include artificial intelligence for healthcare, deep learning for drug discovery, clinical trial optimization, computational phenotyping, clinical predictive modeling, treatment recommendation, and health monitoring.