The GAIN Method for the Completion of Multidimensional Numerical Series of Meteorological Data

Marina Popolizio, Alberto Amato, Federico Liquori, Tiziano Politi, Alessandro Quarto *Member, IAENG*, Vincenzo Di Lecce

***Abstract*—The missing data imputation is a very significant topic which captures considerable interest, given the importance it has in many applications. This paper analyzes the use of GAIN (Generative Adversarial Imputation Networks) to address the problem of missing data in meteorological data sets. A detailed description of the numerical method is given together with a MATLAB implementation which will be available on request.**

Numerical tests are presented to validate the effectiveness of this method; moreover, a comparison on a real dataset is done with the commonly used ARMA method and GAIN turns out to be more accurate.

***Index Terms*—Artificial Intelligence, Missing Data, Imputation, Neural Network, GAIN**

I. Introduction

The services aimed at public safety, precision agriculture, prevention and forecasting of geophysical risks, as well as all services for monitoring the environment, are increasingly using meteorological data. In turn, the availability of meteorological data is also increasing sharply, thanks to the dense networks of professional and private weather stations, or to the remote sensing carried out by an increasingly dense constellation of satellites, that monitor the earth's surface. These services typically use Decision Support Systems (DSS) or systems that, following decisions made automatically, perform actions in the real way. Undoubtedly, the quality and reliability offered by these services is highly dependent on the quality and quantity of the meteorological data used as input. The presence of *missing data* is however a problem that is almost always present in meteorological data and affects their quality. This issue in the meteorological data could be caused by different phenomena:

- natural factors, where one size prevents the recording of another size (e.g. the presence of clouds in satellite images or the presence of strong wind which prevents the recording of rain);
- malfunctions in the instrumentation;
- incorrect installation of the instrumentation;
- errors in the processing of the data by the entity;
- errors in acquisition systems that collect data from different sources;
- data marked as invalid by the body that provides them.

The missing data problem is present also when one is interested in generating a complete map relating to a geographical area, which reports the value of a meteorological quantity, starting from a map in which only some observations of the magnitude of interest are reported. The operation of attributing to missing data the value they would have assumed is identified with the name of *missing data imputation*.

In this way it is possible to carry out the action of completing the series, for example temporal or spatial. This operation eliminates values not attributable to the event but generated outside the system considered. The data used to replace the missing ones are arbitrary, since no one knows reality, and are generated from models and/or collections of correct data If the essential requirement of this operation is to generate a value as close as possible to the presumed true value of the missing data in order to improve the representation of the event, then an increase in the quality of the operation of these systems will be achieved. Furthermore, defining a requirement related to the quality of this integration, more or less directly, also defines a metric by which to measure the validity of the method.

In this work we propose to use the Generative Adversarial Imputation Networks (GAIN) method for *missing data imputation* in meteorological datasets.

The paper is organized as follows: in Section II we briefly review the state-of-the-art for the treatment of missing values. Section III describes the characterization of missing values and the mechanisms which can regulate their distribution. The Generative Adversarial Network (GAN) is described in Section IV before describing the GAIN method: Section V addresses its mathematical description while Section VI deals with the related algorithm. Then Section VII and Section VIII describe the implementation issues and the parameter settings, respectively. Numerical tests are presented in Section IX: in particular, the performance of the GAIN method is described to complete time series from a real dataset and its results are compared with the performance of the commonly used ARMA method. Conclusions are reported in Section X.

Manuscript received October 31, 2020; revised February 26, 2021.

This work is supported under the project "SeVaRA" which is a research project under the grant PO FESR 2014/2020, proposing subject "Omnitech S.r.l.-Rome"- Project code: 2NQR592.

M. Popolizio is Professor of Applied Mathematics at DEI, Politecnico di Bari, Bari, 70126, Italy (email: marina.popolizio@poliba.it).

A. Amato is scientific collaborator at DEI, Politecnico di Bari, Bari, 70126, Italy (email: alberto.amato@poliba.it).

F. Liquori is scientific collaborator at DEI, Politecnico di Bari, Bari, 70126, Italy (email: federico.liquori@poliba.it).

T. Politi is Professor of Applied Mathematics at DEI, Politecnico di Bari, Bari, 70126, Italy (email: tiziano.politi@poliba.it).

A. Quarto is scientific collaborator at myHermes Srl, Taranto, 74121, Italy (email: alessandro.quarto@myhermessrl.com).

V. Di Lecce is Professor of Computer Science at DEI, Politecnico di Bari, Bari, 70126, Italy (email: vincenzo.dilecce@poliba.it).

II. Some notes on the available literature

The first works related to missing data go back to the 1930s [1] while the first literature review concerning data analysis with missing values dates back to 1966 [2]. Over the years, since the early 1970s, several methods have been proposed for imputing missing data and we just refer to the extensive book by Little and Rubin [3] for a comprehensive treatment and a rich list of references and to the recent review by Silva and Zarate [4]. The simplest solution, often referred to as *missing data ignoring technique*, could be to remove records that contain missing values. It may be satisfactory when missing data are few; however, since the data dimension is clearly reduced, this can represent a serious problem when dealing with scarce data, thus to possibly reduce the statistical power. Secondly, ignoring data can significantly reduce the model's accuracy and bias the results. Another possibility is to replace the missing values: this procedure is called *imputation*. For example, the missing value may be replaced with a plausible one, as for example the mean for the cases that observe the variable. A possible drawback of this strategy is the change in the distribution of that variable, with possible consequences on the variance.

Several methods exist with the purpose of achieving a model that is able to represent, in a generic way, the characteristics of the data. Then, these models can be used as inference mechanism to impute the desired values. Among the approaches belonging to this class, we cite the likelihood algorithms and the neural networks.

Regarding the former, in the case of complete data, quantities such as the mean and linear regression coefficients represent maximum likelihood estimates, based on maximizing the likelihood of the data. To define the likelihood of the observed data is much more difficult when missing data occur. In this case, in 1977 Dempster [5] proposed the Expectation-Maximization (EM) algorithm, an iterative method for the efficient estimation of incomplete data. The EM algorithm represents a milestone in the treatment of missing data. However, an approach of this kind is not able to impute the individual missing values. The *multiple imputation* is an alternative approach which aims to obtain estimates of the missing values; to this purpose, random draws are simulated from the distribution of the missing variables given the observed variables. For a thorough description we refer to [3].

Nowadays, models based on Neural Networks (NNs) are used in a large number of applications. In [6] authors use an artificial NN model trained by a metaheuristic algorithm such as artificial bee colony (ABC) in a bankruptcy prediction problem. In [7] convolutional NNs (CNNs) with faster-RCNNs architecture is proposed to detect SARS-CoV-2 lesions in computer tomography images. Other interesting applications can be found in [8] where NNs are used in speech emotion recognition and in [9] where they are used in hand gesture recognition. The first application of NNs to impute missing values dates back to the beginning of 2000s [10]. Networks are indeed well suited to deal with missing data, due to their robustness and generalization capacity. Many methods have been proposed in this direction, also in combination with other approaches (see [4] and references therein).

Among approaches based on NNs, the Generative Adversarial Nets (GAN) is a well-established method proposed in 2014 [11]. It is a deep-learning-based generative model that consists in training a generative model as a supervised learning problem with two sub-models: the generator model is trained to generate new examples, while the discriminator model tries to classify examples as either real (from the domain) or fake (generated). So it is defined by two NNs to train alternately.

In the context of meteorological applications, real time processing is often required and particularly performing methods are needed. For this reason we consider the use of GAIN (Generative Adversarial Imputation Networks) which is a specialization of the GAN framework. According to studies in the literature, this method is more efficient than traditional imputation methods [12]. We give a detailed description of this method in Section V and Section VI, while in Section IX we show its effectiveness by means of some numerical tests.

III. Characterization of the missing data

The imputation process considerably benefits from a preliminary knowledge of the characteristics of missing values. Moreover, it is particularly useful to characterize from a probabilistic point of view the mechanism by which the missing data are realized, or *the missing data mechanism*. The first to address this topic was Rubin in 1976 [13]. The characterization of the missing data is mainly defined by the pattern with which they occur in the dataset in conjunction with the type of mechanism they are created with. These characterizations are therefore used for the selection of the most appropriate imputation algorithm. The theory relating to the characterization of missing data and their imputation is very extensive, in fact, there are numerous books that deal with this problem from a purely statistical point of view [3], [14].

A. Missing data patterns

The presence of missing data within the dataset can manifest itself with different patterns [3]. These patterns can be more or less carriers of information, therefore, it is useful to observe their structure. Furthermore, the selection of the missing data handling methods could be made in relation to the type of pattern present in the dataset. Let $\mathcal{D}$ be a dataset (a matrix) of dimensions $n \times k$, where n are the rows, i.e. the observations, and k are the variables of the sample (observation) also called features. For the dataset $\mathcal{D}$ with elements d_{ij} it is possible to associate a matrix M of equal size, where the element m_{ij} will have value 0 if the element d_{ij} is missing, while it will have value 1 if the element d_{ij} is observed. We will call M the *mask* matrix. Through the matrix M thus defined, it is possible to observe the pattern with which the missing data are present within $\mathcal{D}$.

B. Missing data mechanisms

We now discuss the mechanisms that regulate the presence of missing data within a dataset, i.e. there is the problem of understanding if the missing value is in some way linked to the current value of that variable for that sample or if it is in relationship with the value of other variables, or

with neither of them. The mechanisms of missing data are often crucial for the choice of the method used to impute them, as some methods are designed by hypothesizing the presence of a certain type of mechanism, or alternatively, are based on hypotheses that are in conflict with the missing mechanism present in the dataset. The missing data mechanism was formalized by Rubin [13], who proposed to treat the missing data indicators as random variables, therefore with an associated probability distribution. According to this analysis, missing data in a dataset $\mathcal{D}$ can be regulated by three types of mechanisms: Missing Completely At Random (MCAR), Missing At Random (MAR), Not MAR (NMAR or even MNAR).

We speak of NMAR (also called non-skippable mechanism, unlike MCAR and MAR, indicated as skippable mechanisms), if the missing mechanism depends on the value that should have appeared in the dataset in place of the missing symbol, in addition to the fact that there may also be dependence on observed data. The adjective "non-skippable" means that it is necessary to model the mechanism that generates the missing data.

We talk about MCAR if the presence of missing data does not depend on the values appearing in $\mathcal{D}$, precisely, they do not depend on the values observed in $\mathcal{D}$, nor on the values that are missing in $\mathcal{D}$. This assumption does not mean that the pattern itself is random, but rather that missingness does not depend on the data values. Formally, this mechanism can be written mathematically as

$$p(M|\mathcal{D};\phi) = p(M|\phi), \text{ for all } \mathcal{D},\ \phi$$

where $p(M|\mathcal{D};\phi)$ is the conditional probability and ϕ represents an unknown parameter. This means that the probability of having a structured configuration such as M is conditioned by the values present in the dataset and by an unknown parameter which precisely parameterizes the probability distribution p. It is not necessary to specify what type of probability distribution is $p(M|\mathcal{D};\phi)$, but it is necessary to understand that M is a random variable that is realized according to $p(M|\mathcal{D};\phi)$. It is precisely $p(M|\mathcal{D};\phi)$ the "mechanism" of the missing data in $\mathcal{D}$.

To give an example, we consider a collection of outdoor temperature data organized by columns (weather stations) and rows (time). The nature of these data is such that we assume both a temporal and a geographic relationship between them. If, for example, a complete row of data were missing, reconstruction of this row with the computed values would be much less accurate than if only some values for an assigned time value t were missing and not all.

MAR is mentioned if the presence of missing data in $\mathcal{D}$ depends on the elements in $\mathcal{D}$. Let $\mathcal{D}_{obs}$ be the set of observed elements of $\mathcal{D}$ and $\mathcal{D}_{miss}$ be the set of missing elements. In the presence of a MAR mechanism, the missing values depend on those observed, formally

$$p(M|\mathcal{D};\phi) = p(M|\mathcal{D}_{obs};\phi) \text{ for all } \mathcal{D}_{miss},\ \phi.$$

For example, with reference to the outdoor temperature example, it is possible to identify and correct values above and below the threshold due i.e. to sensor malfunctioning.

IV. GAN

Before analyzing the GAIN method we start from the description of its precursor GAN in order to highlight the basic concepts and then to stress the differences with respect to GAIN. The GAN method, as originally proposed by Goodfellow [11], is based on an adversarial process which simultaneously trains two models: a generator G that captures the data distribution and a discriminator D that estimates the probability that a sample came from the training data rather than G. In this way an *adversarial* process comes into play: the generator's goal is to accurately impute missing data, while the discriminator's goal is to distinguish between observed and imputed values.

For ease of presentation, we deal with the case of a single sample since we are just interested in showing the main ideas of the approach.

The generator G is a NN that models a transform function; so it takes as input a simple random variable and must return, once trained, a random variable that follows the targeted distribution. Mathematically, it is a function that maps a latent space into the data space. Given $\mathbf{z}$ a sample of the latent space (namely, a noise signal), and $\mathbf{x}$ a sample of the data space, the realization probabilities are denoted respectively by $p_z(\mathbf{z})$ and $p_{data}(\mathbf{x})$. G is a differentiable function with parameters θ_g and implicitly defines a probability distribution p_g as the distribution of the samples $G(\mathbf{z})$ obtained when $z \sim p_z$.

The discriminator D is another NN which maps the data space into a probability value, i.e. $D(\mathbf{x}) \in (0,1)$ indicates the probability that the sample $\mathbf{x}$ given as input to D is a real champion; if $\mathbf{x}$ is real, then D should output a number close to 1, while if $\mathbf{x}$ is synthetic, then D should output a number close to 0. Also D depends on some parameters θ_d.

The discriminator must be able to indicate (through its output) whether the sample submitted is taken from the generative model of G indicated with $p_g(\mathbf{x})$ or from the generative model of the real data $p_{data}(\mathbf{x})$; therefore, during the training D will be trained to recognize (indicating a probability value) if a sample is extracted from $p_{data}(\mathbf{x})$. The generator instead, during the training, will try to make $p_g(\mathbf{x})$ as similar as $p_{data}(\mathbf{x})$.

The training process takes place alternately: initially D is trained so that it learns more or less weakly to discriminate the real samples from the synthetic ones; then it trains G, so as to completely deceive D. At this point, one proceeds by training D again, so that it can again correctly discriminate the real samples from the synthetic ones, considering that now the synthetic ones will be of better quality than the previous ones, since G was trained and was able to make $p_g(\mathbf{x})$ more like to $p_{data}(\mathbf{x})$. This alternate training process allows G and D to improve each other.

Nets G and D (i.e. the two competitive agents) are trained through a minimax game, namely

$$\min_G \max_D V(D,G) \tag{1}$$

with objective function

$$V(D,G) = \mathbb{E}_{\mathbf{x}\sim p_{data}(\mathbf{x})}[\log D(\mathbf{x})] + \mathbb{E}_{\mathbf{z}\sim p_z(\mathbf{z})}[\log(\mathbf{1} - D(G(\mathbf{z})))]$$

where $\mathbf{1}$ is the vector of all 1s and log is the element-wise logarithm.

The minimax game consists in having D with the aim of maximizing the probability of correctly predicting which elements in a sample are imputed and which instead are observed, while G with the aim of minimizing the probability that D makes the correct prediction.

G intends to minimize V, so it will try to maximize $D(G(\mathbf{z}))$ for each synthetic sample $G(\mathbf{z})$ generated; in this way $D(G(\mathbf{z}))$ approaches the value 1, so that the logarithm assumes a very small negative value for each synthetic sample (therefore the expectation will also assume the minimum value, and V will be minimized). In order to do this, G can only act on its weights θ_g.

Agent D, on the other hand, will try to maximize $D(\mathbf{x})$, that is, it will try to output values close to 1 in the presence of real samples as input, so that the expectation at the first addend of V is maximized; therefore, when it receives synthetic samples $G(\mathbf{z})$, the value $D(G(\mathbf{z}))$ will be minimal, and therefore the expectation of the second addend will be maximum (therefore both the addends of V will be maximized). In order to do this, D can only act on its weights θ_d.

The minimax game (1) has a global optimum for $p_g = p_{data}$. The algorithm proposed by Goodfellow [11] is based on a stochastic gradient descent method. So, at each iteration of the training process, the weights of the generative network are updated in order to increase the classification error whereas the weights of the discriminative network are updated to decrease this error.

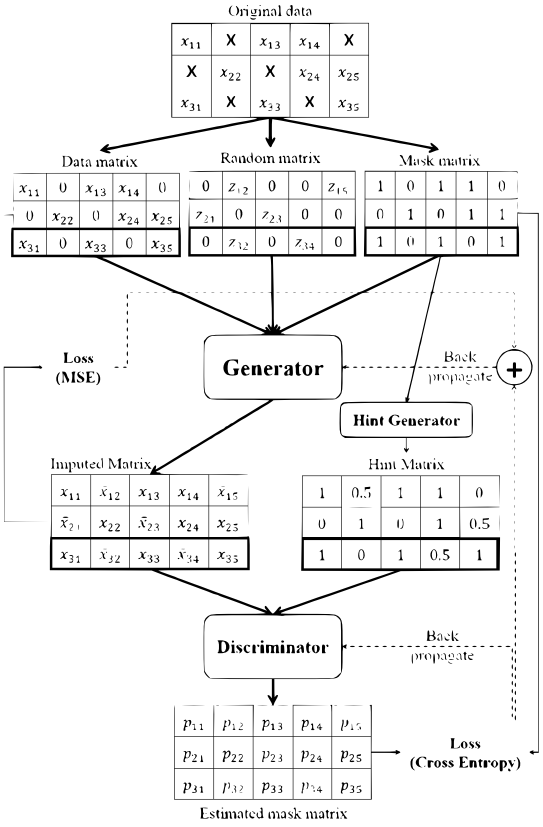

Fig. 1. GAIN's architecture

V. GAIN METHOD: THE MATHEMATICAL DESCRIPTION

Recently, a variant of GAN, called Generative Adversial Imputation Nets (GAIN), has been proposed by Yoon [12]. As in the GAN method, the key players are the generator G and the discriminator D while an additional ingredient, the *hint* H, is added.

In practice, the generator observes a real data vector, with possibly missing data; it imputes these missing values conditioned on what is actually observed and returns a completed vector. The discriminator takes this completed vector and tries to distinguish the observed components from the imputed ones. At this point the hint enters the scene: it reveals to the discriminator some information about the missingness of the original sample. Essentially, it ensures that G imputes data according to the true underlying data distribution.

A positive feature of the GAIN method is that it manages to work effectively even when complete data is unavailable (instead of some generative models that require a complete dataset in the training phase).

Yoon [12] gave theoretical results for the GAIN approach under the hypothesis that the MCAR mechanism regulates the missing data, that is, the missingness occurs entirely at random. However, the effectiveness of this approach was empirically showed also for the MAR and NMAR settings.

Figure 1 gives a graphic description of GAIN's architecture.

Let analyze the mathematical details of the GAIN method in a general framework.

The starting point of our procedure is a data vector, that we denote with $\mathbf{X}$. To be precise, $\mathbf{X} = (X_1, \ldots, X_d)$ is a random variable (continuous or binary) in a d-dimensional space $\mathcal{X} = \mathcal{X}_1 \times \ldots \times \mathcal{X}_d$ and its probability distribution will be denoted as $p_{data}(\mathbf{X}) = P(\mathbf{X})$.

In the following, given a random variable, we denote with lower-case letters its realizations.

We associate to $\mathbf{X}$ the *mask vector* $\mathbf{M}$, that is, a random variable $\mathbf{M} = (M_1, \ldots, M_d)$ with values in $\{0,1\}^d$. In practice, the 1's in $\mathbf{M}$ indicate which components of $\mathbf{X}$ are observed while the 0's correspond to the missing values.

For each $i \in \{1, \ldots, d\}$ a new space is defined $\tilde{\mathcal{X}}_i = \mathcal{X}_i \cup \{*\}$ where the symbol $*$ identifies an unobserved value. In the space $\tilde{\mathcal{X}} = (\tilde{\mathcal{X}}_1, \ldots, \tilde{\mathcal{X}}_d)$ we define a new random variable $\tilde{\mathbf{X}} = (\tilde{X}_1, \ldots, \tilde{X}_d)$ in the following way:

$$\tilde{X}_i = \begin{cases} X_i, & \text{if } M_i = 1, \\ *, & \text{else.} \end{cases} \tag{2}$$

The dataset is $\mathcal{D} \equiv \{(\tilde{\mathbf{x}}^1, \mathbf{m}^1), \ldots, (\tilde{\mathbf{x}}^n, \mathbf{m}^n)\}$ made up of n independent and identically distributed (i.i.d.) copies of $\tilde{\mathbf{X}}$, namely $\tilde{\mathbf{x}}^1, \ldots, \tilde{\mathbf{x}}^n$, with $\mathbf{m}^i$ denoting the realization of $\mathbf{M}$ corresponding to $\tilde{\mathbf{x}}^i$.

The goal of GAIN is to impute the missing values in each $\tilde{\mathbf{x}}^i$ so as to replace all missing data in $\mathcal{D}$; this corresponds to generating samples according to $P(\mathbf{X}|\tilde{\mathbf{X}} = \tilde{\mathbf{x}}^i)$, that is, the conditional distribution of $\mathbf{X}$ given $\tilde{\mathbf{X}} = \tilde{\mathbf{x}}^i$.

A. Generator

The generative network G takes as input the realizations of $\tilde{\mathbf{X}}$, the mask $\mathbf{M}$ and a random variable $\mathbf{Z} = (Z_1, \ldots, Z_d)$ which is a noise (exactly as in the GAN setting). Then it produces as output the vector $\bar{\mathbf{X}}$ of *imputed* values. So,

$$G : \tilde{\mathcal{X}} \times \{0,1\}^d \times [0,1]^d \to \mathcal{X}.$$

Hence, $\bar{\mathbf{X}}$ is the synthetic sample output of G in which the missing data have been imputed, but also the values of the observed variables are generated (therefore they could differ from the observed values of the input sample). To get the *completed data vector* $\hat{\mathbf{X}}$ one simply has to replace the missing data of $\tilde{\mathbf{X}}$ with the corresponding imputed values present in $\bar{\mathbf{X}}$.

Formally,

$$\bar{\mathbf{X}} = G(\tilde{\mathbf{X}}, \mathbf{M}, (\mathbf{1} - \mathbf{M}) \odot \mathbf{Z}) \tag{3}$$

$$\hat{\mathbf{X}} = \mathbf{M} \odot \tilde{\mathbf{X}} + (\mathbf{1} - \mathbf{M}) \odot \bar{\mathbf{X}} \tag{4}$$

where $\odot$ denotes element-wise multiplication.

B. Discriminator

As in the GAN framework, the discriminative network D receives and analyzes the output of G. However, in a standard GAN the discriminator must identify if an entire vector is real or fake since the output of the generator is either completely real or completely fake. In a GAIN framework, instead, the output $\hat{\mathbf{X}}$ has some components that are real and some that are fake. So the discriminator attempts to distinguish which components are real (observed) or fake (imputed).

Therefore, the task of D is to predict the mask $\mathbf{M}$ of the completed sample $\hat{\mathbf{X}}$ (note that the mask $\mathbf{M}$ is completely observable, because given a sample of the dataset, it is possible to establish which components are missing and which are observed).

So, the output of D is a vector whose elements are probabilities; precisely, the i-th component of this vector is the probability that $\hat{\mathbf{x}}^i$ was observed.

C. Hint mechanism

As mentioned before, D needs a hint mechanism for G to learn the probability distribution of the data. For this reason, a random variable $\mathbf{H}$ is used which is dependent on $\mathbf{M}$, so that in $\mathbf{H}$ there is part of the information of $\mathbf{M}$, in such a way that D can partly know which of the components of the sample $\hat{\mathbf{X}}$ are observed or missing.

So, with this perpetuated help during training, D will gradually become more accurate in discriminating the components observed from the imputed ones and, consequently, G will have to improve its generative model, to put D in difficulty.

Therefore, the random variable $\mathbf{H}$ has probability density $P(\mathbf{H}|\mathbf{M} = \mathbf{m})$, i.e. the probability of having a certain realization of $\mathbf{H}$ is conditioned by the value of the mask of the sample $\tilde{\mathbf{X}}$. The definition of $\mathbf{H}$ can occur in different ways, so as to be able to control the amount of information present in $\mathbf{M}$ that is intended to be shown to D.

So, differently from the GAN method, D has now two input, $\hat{\mathbf{X}}$ and $\mathbf{H}$, while its output is a vector whose elements are probabilities, precisely, the i-th component of $D(\hat{\mathbf{X}}, \mathbf{H})$ is the probability that $\hat{\mathbf{x}}^i$ is not missing (i.e. the probability of being an observed value). Formally

$$D : \mathcal{X} \times \mathcal{H} \to [0,1]^d$$

where $\mathcal{H}$ is the *hint* space where the hint variables $\mathbf{H}$ live.

D. Minimax formulation of the problem

GAIN, like all GANs, is a minimax game: we train D to maximize the probability of correctly predicting $\mathbf{M}$ and we train G to minimize the probability of D predicting $\mathbf{M}$. Then the objective function is

$$\begin{aligned} V(D,G) = \mathbb{E}_{\hat{\mathbf{X}},\mathbf{M},\mathbf{H}}[\mathbf{M}^T \log D(\hat{\mathbf{X}}, \mathbf{H}) \\ + (\mathbf{1} - \mathbf{M})^T \log(\mathbf{1} - D(\hat{\mathbf{X}}, \mathbf{H}))] \end{aligned} \tag{5}$$

where dependence on G is through $\hat{\mathbf{X}}$ and the minimax problem to solve is

$$\min_G \max_D V(D,G). \tag{6}$$

The minimax game can be defined in a more explicit notation through the loss function $\mathcal{L} : \{0,1\}^d \times [0,1]^d \to \mathbb{R}$ constructed as

$$\mathcal{L}(\mathbf{a}, \mathbf{b}) = \sum_{i=1}^{d} [a_i \log(b_i) + (1 - a_i) \log(1 - b_i)].$$

Then the objective function (5) may be written as

$$\min_G \max_D V(D,G) = \mathbb{E}[\mathcal{L}(\mathbf{M}, \hat{\mathbf{M}})].$$

E. Theoretical analysis

Yoon [12] gave a theoretical analysis of the minimax problem (6). In particular, they assume that $\mathbf{M}$ is independent on $\mathbf{X}$, i.e. the mechanism for the data is MCAR.

It is shown that if $\mathbf{H}$ does not provide "enough" information about $\mathbf{M}$, it is not possible to guarantee that G learns the desired probability distribution.

Moreover, they give the explicit expression of the hint $\mathbf{H} = (H_1, \ldots, H_d)$ which ensures that G learns to replicate the desired probability distribution [12]. It is

$$\mathbf{H} = \mathbf{B} \odot \mathbf{M} + 0.5(\mathbf{1} - \mathbf{B}) \tag{7}$$

where $\mathbf{B} = (B_1, \ldots, B_d) \in \{0,1\}^d$ is a random variable; it is obtained by first sampling k from $\{1, \ldots, d\}$ uniformly at random, where each value in this set has identical possibility of be extracted, then defining

$$B_j = \begin{cases} 1, & \text{if } j \neq k \\ 0, & \text{if } j = k. \end{cases}$$

It results that H_i may only assume values $0, 0.5, 1$; moreover if $H_i = 0$ or $H_i = 1$, then $M_i = H_i$, while $H_i = 0.5$ does not give information on M_i.

Since the hint mechanism defined as (7) ensures that the generator learns to replicate the desired distribution, we will use it in our implementation of the GAIN algorithm.

VI. GAIN METHOD: THE ALGORITHM

In GAIN the minimax game is solved through an iterative process where G and D are implemented as fully connected NNs. The corresponding Matlab code, which will be used in all subsequent experiments, will be available on request. The training has been performed exploiting a custom loop within the `Matlab` Deep Learning toolbox (R2020). The steps of the loop are described in *Algorithm 1*.

To start, the size N of the mini-batches to be used is fixed.

Algorithm 1 Pseudo-code GAIN Training

Set mini-batch dimension to N
epoch ← 1
while training loss has not converged OR epoch ⩽ maxNoEpoch **do**:
  iteration ← 1
    While iteration ⩽ fix (No. Training Obs / N)
    (1) Mini-batch selection
    First Index ← $1 + (iteration - 1) \times N$
    Last Index ← $iteration \times N$
    Draw N samples from $\{ (\tilde{x}(j), m(j)) \}_{j=FirstIndex}^{LastIndex}$
    Draw N samples from $\{ z(j) \}_{j=FirstIndex}^{LastIndex}$
    Draw N samples from $\{ b(j) \}_{j=FirstIndex}^{LastIndex}$
    Draw N samples from $\{ h(j) \}_{j=FirstIndex}^{LastIndex}$
    Draw N samples from $\{ x(j) \}_{j=FirstIndex}^{LastIndex}$
    (2) Discriminator Optimization
    $InputGenerator$ ← Concatenate $\tilde{x}(j)$ and $m(j)$
    $XGenerated$ ← G($InputGenerator$)
    $InputDisc$ ← $\tilde{x}(j) \odot m(j)$ + $XGenerated \odot$ (1-$m(j)$)
    $InputDisc$ ← Concatenate $InputDisc$ and $h(j)$
    $DiscProbability$ ← D($InputDisc$)
    Compute Discriminator Loss
    Compute Discriminator Gradients
    Update Discriminator parameters through *Adam* solver
    (3) Generator Optimization
    Compute Generator Loss
    Compute Generator Gradients
    Update Generator parameters through *Adam* solver
    iteration ← iteration + 1
  epoch ← epoch + 1
end while

A. Definition of Layers

The generator has seven layers: the first one defines the inputs. In practice, for each sample in the mini-batch $(\tilde{\mathbf{x}}_j, \mathbf{m}_j)$ we draw N independent samples of $\mathbf{Z}$ and $\mathbf{B}$, say $\mathbf{z}_j$ and $\mathbf{b}_j$, and compute $\hat{\mathbf{x}}_j$ and $\mathbf{h}_j$ accordingly, by following formulas (4) and (7) respectively. The other 6 layers are three fully connected layers alternated with the rectified linear activation function (or ReLU for short); this is a piecewise linear function that will output the input directly if it is positive otherwise it will output zero. It introduces non-linearity in the system. The fully connected layers multiply the outputs from the previous layer by a weight matrix and then add a bias vector.

The structure of the discriminator is the same as that of the generator. The only difference is the presence of an eighth layer that applies the sigmoid function $f(x) = 1/(1 + e^{-x})$ to the outputs of the last ReLU activation layer. The discriminator has to predict a probability, therefore, since the sigmoid function generates value between 0 and 1, it fits better this need.

B. Loss Functions

Two loss functions are defined in the context of the GAIN method, one to train the generator and one for the discriminator; they are clearly different because different are the scopes of the two networks. For D we recall that the theoretical analysis by Yoon [12] ensures that we may focus on the outputs of D corresponding to $b_i = 0$ for each sample, that is, those values not suggested by $\mathbf{H}$. Therefore its loss function is defined as

$$\mathcal{L}_D(\mathbf{m}, \hat{\mathbf{m}}, \mathbf{b}) = \sum_{i:b_i=0} [m_i \log(\hat{m}_i) + (1 - m_i) \log(1 - \hat{m}_i)] \tag{8}$$

and we train D according to

$$\min_D - \sum_{j=1}^{N} \mathcal{L}_D(\mathbf{m}_j, \hat{\mathbf{m}}_j, \mathbf{b}_j)$$

with $\hat{\mathbf{m}}_j = D(\hat{\mathbf{x}}_j, \mathbf{m}_j)$.

Even if the first step is to optimize D, we preliminarily need the previsions of G since $\hat{\mathbf{x}}$ actually involves the output $G(\tilde{\mathbf{x}}_j, \mathbf{m}_j, \mathbf{z}_j)$.

For the optimization of G the attention is on two fronts since G outputs the entire data vector (so both $m_i = 0$ and $m_i = 1$): for the missing components the goal is to impute the missing values in such a way to fool D. On the other hand, when $m_i = 1$, the observed components must be close to the actually observed data. For these reasons, the loss function for G combines two different functions: the first part is

$$\mathcal{L}_G(\mathbf{m}, \hat{\mathbf{m}}, \mathbf{b}) = - \sum_{i:b_i=0} (1 - m_i) \log(\hat{m}_i). \tag{9}$$

It is evident that

$$\mathcal{L}_D, \mathcal{L}_G : \{0,1\}^d \times [0,1]^d \times \{0,1\}^d \to \mathbb{R}.$$

The loss function (9) applies to the missing components (namely, those with $m_i = 0$) and it is smaller when $\hat{m}_i$ is closer to 1. So $\mathcal{L}_G$ is smaller when D falsely categorizes an imputed value as observed. To work on the observed components another loss function is defined, that is $\mathcal{L}_M : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ such that

$$\mathcal{L}_M(\mathbf{x}, \mathbf{x}') = \sum_{i=1}^{d} m_i L_M(x_i, x_i') \tag{10}$$

where L_M changes expression if x_i is a continuous or binary random variable, namely

$$L_M(x_i, x_i') = \begin{cases} (x_i' - x_i)^2 & \text{if } x_i \text{ is continuous} \\ -x_i \log(x_i') & \text{if } x_i \text{ is binary.} \end{cases}$$

$\mathcal{L}_M$ is minimized when the value assigned to the observed data is close to the real observed value.

To train G we need to solve the problem

$$\min_G \sum_{j=1}^{N} \mathcal{L}_G(\mathbf{m}(j), \hat{\mathbf{m}}(j), \mathbf{b}(j)) + \alpha \mathcal{L}_M(\tilde{\mathbf{x}}(j), \hat{\mathbf{x}}(j))$$

where the parameter α is a hyperparameter used to control the training process on G.

For the stochastic optimization the Adaptive Moment Estimation (*Adam*) Solver [15] has been considered while for the weights initializations of both G and D the Xavier method [16] has been used.

VII. Implementation Issues

We now describe the specific features we used in the implementation of Algorithm 1.

A. Dataset Building and Understanding

Numerical evaluations were performed on a dataset of temperatures collected from meteorological stations in Apulia (southern Italy). They come from 255 stations and all data belong to 2019. The data were taken from the DarkSky website (https://darksky.net/dev/docs); it is a web service that provides weather data for any desired location.

From each station we have extracted as many consecutive sequences of 16 values as possible. The choice of a sequence length of 16 values has two reasons: first of all to improve the computational efficiency. Secondly, it is a trade-off research where the data distribution must be respected while the length of the sequence could not be too large, since the number of weights in the neural networks would increase as well. In other words, small sequences mean less weights and thus less parameters to estimate. However, considering different rows as independent, the underlying data distribution could not be seen with few values per each line of the dataset. If you look at a plot with 10 values, you can have a better understanding of the phenomenon compared to a plot with 3 values. However if the number of points is 1 billion, a great part of the information is lost anyway because you cannot look at all points at once. The rationale behind the choice of the sequence length of 16 values is the same. Each sequence had to respect few rules: the absence of missing data and a regular time step between the observations of the sequences. In other words all observations have the same temporal distance. The result is a dataset of 75,980 sequences for a total of 1,215,680 temperatures. The training phase of GAIN has been done on the 80% of the complete dataset, while for the test phase the remaining 20% has been used.

VIII. Parameter Settings

The generation of the Mask depends on a custom parameter which defines the percentage of missing values introduced in the dataset. For our numerical tests this parameter has been varied between 0.05 and 0.8. It is fundamental to underline that the generation of the Mask, and thus the introduction of the missing data in the dataset, should represent a completely random process. Therefore, the overall ratio between the number of missing values and the total number of data must respect the setted value. However, there are no constraints between different rows of the training set. In other words, the number of missing values between different rows of the dataset could be different. This should help the NN to better associate the presence of a zero in the mask with the presence of a missing value in the sequence. The same process applies to the Hint mechanism.

All parameters used in the numerical tests of Section IX are summarized in Table I. In particular, the parameters of `Adam` Solver [15] are set by default.

A. To measure the accuracy

In every artificial intelligence algorithm it is essential to identify the right way to measure the performance of the algorithm itself. In this specific case, the distance between the output of the Generator and the real data is of interest. The score function should take into consideration only the values that the NNs saw as missing during the training. Indeed, in a real scenario, no one would discard the real data, but only trying to compute the missing ones. In other words, the Generator score function must evaluate the error between the imputed values and the real data that were replaced by the noise. For this reason the Generator Score is defined as

TABLE I
Summary of the Algorithm Parameters

Parameter	Values
Mini-Batch size	256
Missing (%)	5, 10, 20, 30, 40, 50, 60, 70, 80
Hint Information (%)	90
Loss parameter α	10
Test Size / Tot observations (%)	20
Max. no. epoch	30
Max. no. iterations	237

$$\frac{mean\left(\left(\mathbf{1}-\mathbf{M}\right)\odot\mathbf{X}-\left(\mathbf{1}-\mathbf{M}\right)\odot\bar{\mathbf{X}}\right)^2}{mean\left(\mathbf{1}-\mathbf{M}\right)} \tag{11}$$

where the squared error is computed only for the values where $(\mathbf{1}-\mathbf{M})=1$, according to (11).

To assess the quality of the GAIN method we make use of the Root Mean Square Error (RMSE) defined as

$$RMSE=\sqrt{\frac{1}{n}\sum_{i=1}^{n}e_i^2}$$

where $e_1,\ldots,e_n$ are samples of the model error we want to quantify. With respect to the commonly used Mean Absolute Error (MAE), the RMSE turns out to be more reliable when the error follows a normal distribution [17], as we are assuming here.

IX. Numerical Tests

We present two kinds of numerical tests: in the first one, to test the effectiveness of GAIN, we work on a complete dataset, that is, a dataset with no missing data, and we artificially remove some of the observed data. In the second example we consider a dataset with missing values and we impute them by using GAIN.

A. Testing GAIN on a complete dataset

The algorithm has been tested introducing different levels of missing values. They were introduced through the generation of the Mask, and their percentage has been varied in order to show the effectiveness of the numerical method under different conditions. The results are summarized in Table II and are visible in Fig. 2.

TABLE II
Root Mean Squared Error for Different Levels of Missing Values

% missing data	RMSE train °C	RMSE test °C
5	1.1500	1.1170
10	1.0203	1.0132
20	1.0972	1.1010
30	1.1017	1.1312
40	1.1655	1.1764
50	1.3242	1.3270
60	1.4703	1.4696
70	2.8976	2.9205
80	2.7256	2.7511

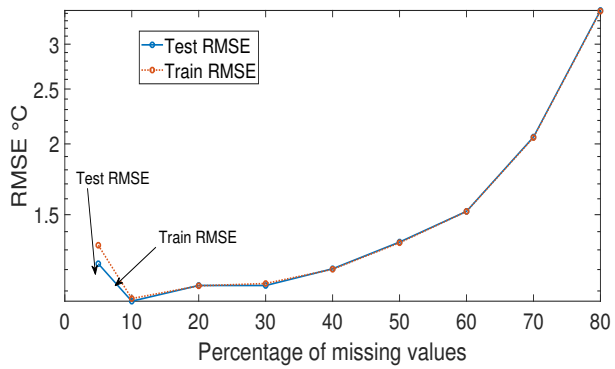

Fig. 2. The RMSE (in °C) as the percentage of Missing Values changes, measured with respect to the test set (continuous line) and the training set (dotted line). From the graphic point of view, the curves can be practically superimposed in pairs.

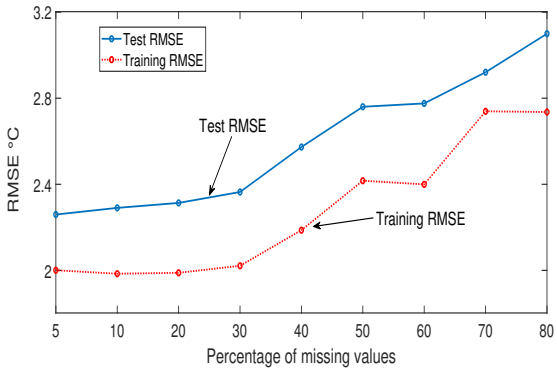

Fig. 3. The RMSE (in °C) for ARMA as the percentage of Missing Values changes, measured with respect to the test set (continuous line) and the training set (dotted line).

As expected the number of missing values and the error are strictly related, while the first increases the second follows. Generally speaking, the errors are very close to 1°C in the range 5-30% of missing data. This is a very common situation in data science researches, especially those related to climate investigation. These numbers confirm the power of the GAIN method in this context. As the percentage of missing values severely increases, say up to the 80%, the error almost increases to 3.5°C. Moreover, another fundamental quality we may observe is the very similar behavior of the two curves, the one corresponding to the test set and the other for the training set. This resemblance shows that GAIN does not overfit the data and is able to keep its specific features.

We run the same test by considering the Auto-Regressive-Moving-Average (ARMA) method: it was first introduced in 1951 by Peter Whittle and is still largely used for the modeling of time series. We follow the built-in implementation by Matlab.

TABLE III
ROOT MEAN SQUARED ERROR OF ARMA FOR DIFFERENT LEVELS OF MISSING VALUES

% missing data	RMSE train °C	RMSE test °C
5	2.001	2.259
10	1.984	2.290
20	1.989	2.313
30	2.021	2.364
40	2.187	2.573
50	2.417	2.760
60	2.399	2.776
70	2.739	2.921
80	2.736	3.100

Interestingly, unlike GAIN, for the ARMA method the differences between the training and the test phases are quite large as shown in Table III. This feature is evident also from Figure 3, where the curves for the RMSE are sensibly distant. This issue may suggest the superiority of GAIN over ARMA in this kind of applications.

In the following plots we refer to the "Epoch" number; however, it is important to stress that each epoch contains an internal iterative process which, in our example, consists of 237 iterations.

It could be of interest to visualize the trend of the score during the training, as the percentage of the missing values changes. In Fig.4 - Fig. 6 we report this comparison. While the number of epochs increases, the score decreases: the descent is very rapid when the 20% of data are missing (Fig. 4) while it is smoother when the 40% are missing (Fig. 6). Remember that the score is an error index, therefore the smaller it is, the better. During the training, the score is calculated over normalized values, therefore it is not a true error index and it is not comparable with the values in Table II.

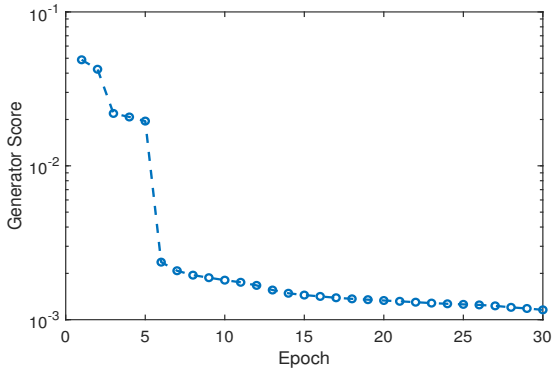

Fig. 4. The Generator Score (11) as the Number of epochs varies with the 20% of missing values.

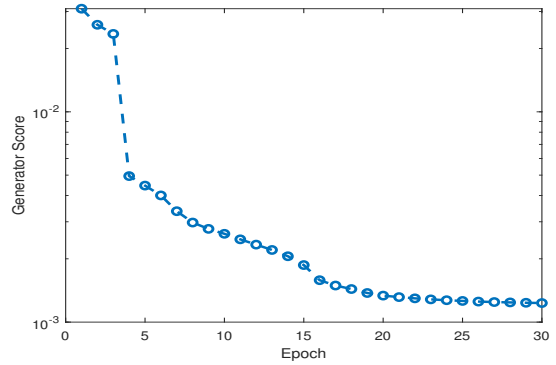

Fig. 5. The Generator Score (11) as the Number of epochs varies with the 30% of missing values.

As regard the loss functions calculated during the training, their trend is represented in Fig 7 which refers to the training

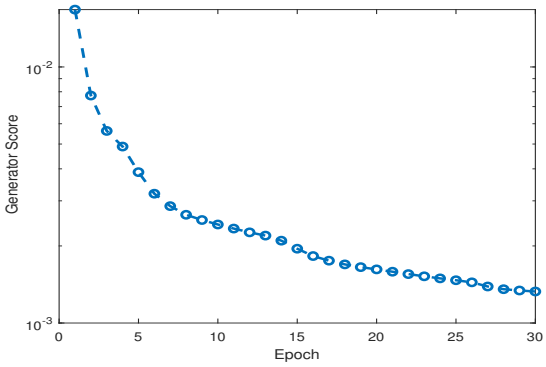

Fig. 6. The Generator Score (11) as the Number of epochs varies with the 40% of missing values.

of the NNs with the 20% of missing values. Both the Generator Loss and the Discriminator Loss functions are monotonically decreasing and they stagnate as the epochs increase. However, the Generator Loss quickly reaches its minimum while the Discriminator Loss has a smoother descent and remains on larger values.

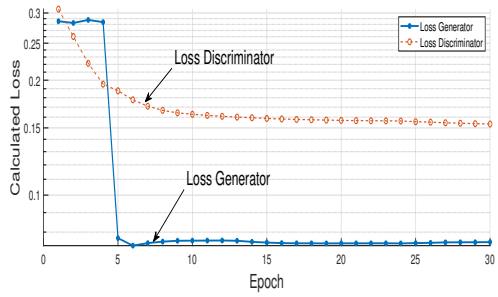

Fig. 7. Loss Functions' trend as the training proceeds (in the case of 20% of missing values).

Generally speaking, every artificial intelligence model is evaluated comparing its score on the test set and on the training set. This helps to refuse those models who are affected by over-fitting. However, it is also true that models trained with vectors full of missing values are rare. Among these rare cases is GAIN. This peculiarity could be visualized through the next four Figures 8-11, each of them referring to a particular level of missing values. In each picture there are four plots. They represent the RMSE evaluated for the training and test sets and on the real and missing values. The error associated to the missing values measures the distance between the original data, that the generator has never seen, and the imputed value after the generator's training. The error associated to the real data measures the distance between the imputed values and all the data that the generator has seen during the training process. It is interesting to notice that in all cases the curves for the test set and the training set almost coincide, meaning that the imputed values well recover the dataset features. Moreover, the errors with respect to the missing values and the real values are very close, although the former is always larger, as expected for what explained before.

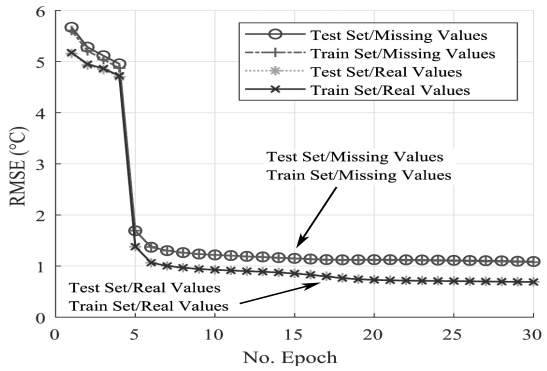

Fig. 8. RMSE evaluated for the training and the test set with respect to the real and missing values (in the case of 10% of missing values). From the graphic point of view, the curves can be practically superimposed in pairs.

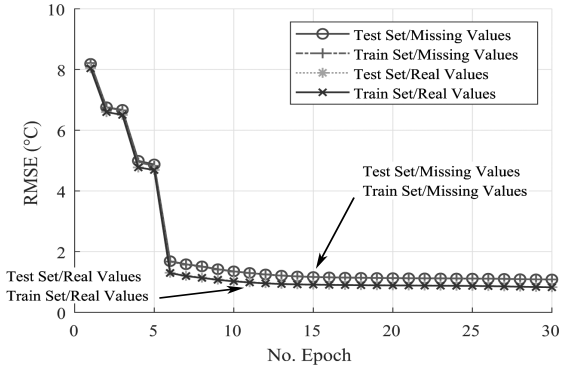

Fig. 9. RMSE evaluated for the training and the test set with respect to the real and missing values (in the case of 20% of missing values). From the graphic point of view, the curves can be practically superimposed in pairs.

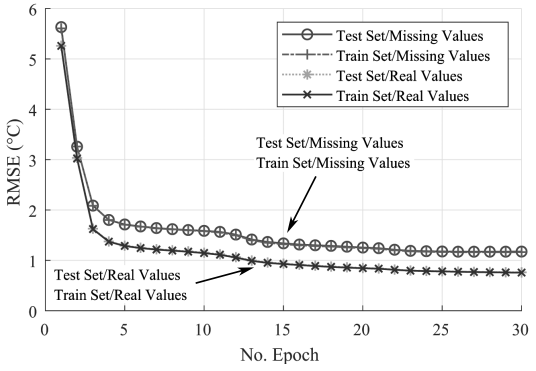

Fig. 10. RMSE evaluated for the training and the test set with respect to the real and missing values (in the case of 30% of missing values). From the graphic point of view, the curves can be practically superimposed in pairs.

B. Imputing missing values

We now test the trained network to impute real missing values. The aim of this test is to evaluate the ability of GAIN to reproduce data in a real research scenario. In the previous test a dataset without any missing value has been used. However, originally, it came from a bigger dataset with some missing values. The original dataset, as stated above, stored temperature data of the Italian region Puglia over 2019. All missing values were identified and isolated. Each missing value was preceded and followed by a total number of 15

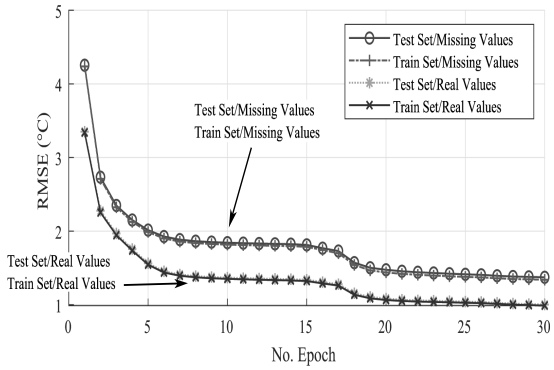

Fig. 11. RMSE evaluated for the training and the test set with respect to the real and missing values (in the case of 40% of missing values). From the graphic point of view, the curves can be practically superimposed in pairs.

values. These were the predecessors and the successors of the missing values in the original signal. If the missing value was far enough from the beginning and from the tail of the sequence, it was centered, with 8 value on the left and 7 on the right. Otherwise, a sufficient number of values before or after the missing value was selected to fulfill the requirement of 16 values as input to the generator (see Section VI-A). In this way a matrix of 3145 rows and 16 columns has been manually created. To better visualize the result of this preparation process, see Table IV. The imputation of the missing values required few more steps. First of all, the Mask M, the Noise Z and the Hint H matrices have been created using the processes introduced above and then $\widetilde{X}$ has been obtained trough formula (2). The values have been then scaled. The Min and Max values for the operation came from the old training set used to train the Generator. The normalized values of $\widetilde{X}$ and M have been concatenated to obtain the 32-columns matrix usable as input for G. Through the *forward* function in Matlab Deep Learning toolbox, it was possible to get the imputed values. The whole imputed values for the samples in Table IV are summarized in Table V. Of course, no one would use imputed values instead of real ones, therefore they have been combined through the formula for $\bar{X}$ (3).

The final result is summarized in Table VI.

TABLE IV
FEW SAMPLES OF ORIGINAL SEQUENCES EXTRACTED IN PRESENCE OF MISSING VALUES

Sequences with missing values															
19.0	20.0	23.6	24.7	25.5	26.3	26.9	26.7	0.0	28.2	28.8	28.8	27.5	27.3	22.6	19.9
19.1	19.1	22.8	24.0	25.0	25.8	26.5	26.2	0.0	27.8	28.4	28.4	27.2	27.1	22.0	19.3
17.0	18.2	22.2	23.6	24.7	25.5	26.2	25.9	0.0	27.4	28.2	28.2	28.0	26.7	21.6	18.9
20.5	20.2	23.1	24.0	25.1	25.7	26.3	26.0	0.0	27.9	28.0	28.2	28.0	26.7	23.3	20.6
17.1	18.2	19.5	20.8	21.9	22.9	23.6	24.3	0.0	25.4	25.8	26.0	26.0	25.9	17.6	17.4
2.9	2.4	1.9	1.3	0.9	0.5	0.3	0.1	0.0	0.0	0.0	0.1	0.1	-0.1	0.2	0.4
3.0	2.6	2.1	1.5	1.1	0.7	0.5	0.3	0.2	0.3	0.3	0.4	0.4	0.0	0.3	0.5

Finally, we give a graphic representation of the imputation results: Figure 12 shows the temperature data in the original sequence, while Fig. 13 shows the temperature after the GAIN imputation. We may appreciate that the latter plot seems to be generally consistent. In an imminent work we aim to better analyze this case, with an extensive statistical analysis of the results' consistency.

TABLE V
IMPUTED SEQUENCES FROM THE SAMPLES IN TABLE IV

Sequences with missing values															
19.1	20.5	22.2	24.3	25.6	26.9	27.9	28.2	28.4	28.4	28.0	27.5	26.7	25.8	24.8	24.1
18.5	19.9	21.6	23.7	25.0	26.4	27.4	27.7	27.9	28.0	27.5	27.0	26.2	25.4	24.4	23.7
17.2	18.8	20.6	23.0	24.4	26.0	27.1	27.6	27.9	28.0	27.5	27.0	26.2	25.3	24.2	23.4
19.7	20.9	22.3	24.1	25.2	26.3	27.2	27.5	27.7	27.8	27.6	27.2	26.7	26.1	25.3	24.7
16.2	17.5	19.1	21.1	22.4	23.8	24.7	25.1	25.3	25.4	25.0	24.5	23.8	23.0	22.0	21.3
0.8	0.6	-0.2	0.9	0.8	0.5	0.3	0.0	2.1	0.4	0.2	0.6	-0.1	1.7	0.4	0.9
1.9	1.3	0.8	0.3	0.3	0.0	0.0	-0.5	0.7	-0.1	0.4	0.7	0.6	1.9	1.9	1.9

TABLE VI
TRUE VALUES COMBINED WITH IMPUTED ONES IN MISSING VALUES POSITIONS

Sequences with missing values															
19.0	20.0	23.6	24.7	25.5	26.3	26.9	26.7	28.4	28.2	28.8	28.8	27.5	27.3	22.6	19.9
19.1	19.1	22.8	24.0	25.0	25.8	26.5	26.2	27.9	27.8	28.4	28.4	27.2	27.1	22.0	19.3
17.0	18.2	22.2	23.6	24.7	25.5	26.2	25.9	27.9	27.4	28.2	28.2	28.0	26.7	21.6	18.9
20.5	20.2	23.1	24.0	25.1	25.7	26.3	26.0	27.7	27.9	28.0	28.2	28.0	26.7	23.3	20.6
17.1	18.2	19.5	20.8	21.9	22.9	23.6	24.3	25.3	25.4	25.8	26.0	26.0	25.9	17.6	17.4
2.9	2.4	1.9	1.3	0.9	0.5	0.3	0.1	0.0	0.4	0.0	0.1	0.1	-0.1	0.2	0.4
3.0	2.6	2.1	1.5	1.1	0.7	0.5	0.3	0.2	0.3	0.3	0.4	0.4	1.9	0.3	0.5

X. CONCLUSIONS

In this work authors propose a method based on neural network to face the problem of missing data imputation for environmental data series. In particular, the Generative Adversarial Imputation Networks (GAIN) have been used. This method can be considered as a specialization of the GAN framework. It is a deep-learning-based generative model that consists in training a generative model as a supervised learning problem with two sub-models: the generator model is trained to generate new examples, while the discriminator model tries to classify examples as either real (from the domain) or fake (generated). To carry out a more comprehensive evaluation of the proposed method, the obtained results have been compared with those obtained using an ARMA model. ARMA is a well known method that is often used in many applications. The obtained results show that the performance obtained using GAIN in terms of RMSE are better than those obtained using ARMA. On the other hand, increasing the percentage of missing values, the RMSE increases more slowly using the ARMA model than using GAIN. Furthermore, it should be highlighted the very small difference between the performance obtained using GAIN on training set and on test set. This difference grows up using ARMA model. This means that the neural networks have

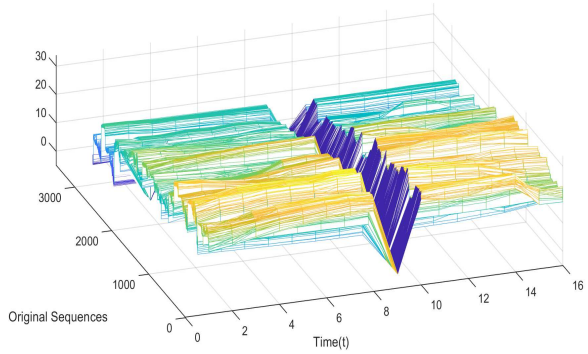

Fig. 12. Temperature in the original sequence (with missing data).

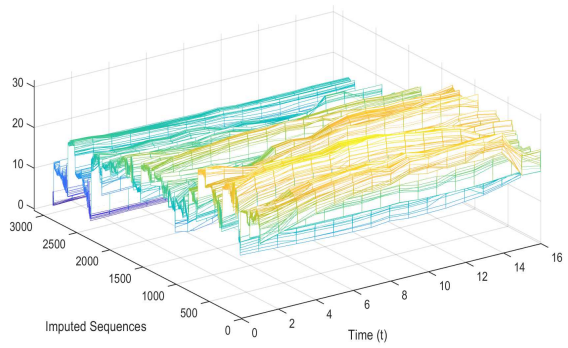

Fig. 13. Temperature after the GAIN imputation.

"learned" the model underlying the time series data and that they are able to generalize its behavior.

REFERENCES

[1] F. Allan and J. Wishart, "A method of estimating the yield of a missing plot in field experimental work," *The Journal of Agricultural Science*, vol. 20, no. 3, pp. 399–406, 1930.

[2] A. A. Afifi and R. M. Elashoff, "Missing observations in multivariate statistics I. Review of the Literature," *Journal of the American Statistical Association*, vol. 61, no. 315, pp. 595–604, 1966.

[3] R. J. Little and D. B. Rubin, *Statistical analysis with missing data*. John Wiley & Sons, 2019, vol. 793.

[4] L. Silva and L. Zarate, "A brief review of the main approaches for treatment of missing data," *Intelligent Data Analysis*, vol. 18, pp. 1177–1198, 01 2014.

[5] A. P. Dempster, N. M. Laird, and D. B. Rubin, "Maximum likelihood from incomplete data via the EM algorithm," *Journal of the Royal Statistical Society: Series B (Methodological)*, vol. 39, no. 1, pp. 1–22, 1977.

[6] S. Marso and M. EL Merouani, "Bankruptcy prediction using hybrid neural networks with artificial bee colony." *Engineering Letters*, vol. 28, no. 4, pp. 1191–1200, 2020.

[7] S. Nurmaini, A. E. Tondas, R. U. Partan, M. N. Rachmatullah, A. Darmawahyuni, F. Firdaus, B. Tutuko, R. Hidayat, and A. I. Sapitri, "Automated detection of covid-19 infected lesion on computed tomography images using faster-rcnns." *Engineering Letters*, vol. 28, no. 4, pp. 1295–1301, 2020.

[8] K. Zheng, Z. Xia, Y. Zhang, X. Xu, and Y. Fu, "Speech emotion recognition based on multi-level residual convolutional neural networks." *Engineering Letters*, vol. 28, no. 2, pp. 559–565, 2020.

[9] L. Jie, Y. Mingqiang, L. Yupeng, W. Yanyan, Z. Qinghe, and W. Deqiang, "Dynamic hand gesture recognition using multi-direction 3d convolutional neural networks," *Engineering Letters*, vol. 27, no. 3, pp. 490–500, 2019.

[10] W. Wei and Y. Tang, "A generic neural network approach for filling missing data in data mining," *Proceedings of the IEEE International Conference on Systems, Man and Cybernetics*, vol. 1, pp. 862–867, Nov. 2003.

[11] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," in *Advances in Neural Information Processing Systems 27*, Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, Eds. Curran Associates, Inc., 2014, pp. 2672–2680.

[12] J. Yoon, J. Jordon, and M. van der Schaar, "GAIN: missing data imputation using generative adversarial nets," *CoRR*, vol. abs/1806.02920, 2018.

[13] D. B. Rubin, "Inference and missing data," *Biometrika*, vol. 63, no. 3, pp. 581–592, 1976.

[14] J. K. Kim and J. Shao, *Statistical methods for handling incomplete data*. CRC press, 2013.

[15] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *CoRR*, vol. abs/1412.6980, 2014.

[16] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," in *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, 2010, pp. 249–256.

[17] T. Chai and R. R. Draxler, "Root mean square error (RMSE) or mean absolute error (MAE)?–Arguments against avoiding RMSE in the literature," *Geoscientific model development*, vol. 7, no. 3, pp. 1247–1250, 2014.