
Greek Statistical Institute
Proceedings of the 34th Panhellenic Statistics Conference (2022), pp. 154−166

SYNTHESIS OF TIME-SERIES WITH MISSING OBSERVATIONS USING GENERATIVE ADVERSARIAL NETWORKS

***Owen D. Jones*[1], *Thomas Poudevigne-Durance*[1], *Yipeng Qin*[2]**
[1] School of Mathematics, Cardiff University, U.K.
{JonesO18, Poudevigne-DuranceT}@cardiff.ac.uk
[2] School of Computer Science and Informatics, Cardiff University, U.K.
QinY16@cardiff.ac.uk

ABSTRACT
We introduce a new method for the data synthesis of time-series using Generative Adversarial Networks (GANs) that can be applied directly to data with missing observations. Unlike previous GAN-based time-series models which use Recurrent Neural Networks (RNNs), our approach directly models the conditional distribution of the current observation given past observations, which it does using an auxiliary GAN trained on the joint distribution of the current and past observations. A benefit of this approach is that it allows us to use a Masked Wasserstein GAN (MaWGAN) to train the model, which can directly accommodate missing values, unlike existing time-series GANs. The veracity of the approach is demonstrated with a simulation experiment, for which we get good results even with high levels of missing data.

Keywords: time-series; data synthesis; missing data; generative adversarial network.

1. INTRODUCTION

Data synthesis refers to the simulation of data while preserving privacy. Many national statistics organisations and state-owned enterprises are interested in data synthesis as a way of distributing the analytic value of data without revealing confidential personal details (Kaloskampis et al. 2019, Sallier 2020). A promising development for data synthesis has been the advent of Generative Adversarial Networks (GANs: Goodfellow et al. 2014); see e.g. Hitawala (2018) for a review. GANs use two neural nets, one to generate synthetic data, and the other to build a critic which is used to train the generator (also called a discriminator). The generator and critic are trained iteratively, so that as the generator improves the critic becomes more discerning, allowing further refinement of the generator. GANs are capable of reproducing complex dependencies in data, and are amenable to privacy protection approaches such as *differential privacy* (Campbell 2019, Jordon et al. 2022). In what follows we will focus on using GANs to model temporal dependencies and will not explicitly consider privacy issues, however we will

assess the performance of our generator using distribution based measures, implicitly judging its ability to reproduce some underlying distribution rather than a specific realisation from it (the data).

Traditional time-series models focus on predictive power rather than data synthesis. Formally they model the signal but not the noise (or texture) around the signal, and for data synthesis both are required. An advantage of a non-parametric model such as a GAN in this setting is that it models both noise and signal simultaneously. Mogren (2016), Esteban et al. (2017) and Yoon et al. (2019) have all previously used GANs for time-series prediction and synthesis. Their approaches all use recurrent neural network architectures (RNNs). RNN-GANs effectively model the conditional distribution of the current observation conditioned on past observations. Information from past observations is encoded as features in one or more hidden layers that are fed back as inputs into the generator for the current observation. Unfortunately there is no clear way to translate missing values into features in these hidden layers, so we take a different approach.

We introduce a two-stage approach to build a model for the conditional distribution of the current observation given past observations. Firstly we build a GAN model for the joint distribution of present and past observations, then secondly we leverage the generator and critic from the joint distribution to build a forecaster that directly models the target conditional distribution.

Missing data is ubiquitous and is as much of an issue for data synthesis as elsewhere. Until recently missing data has been a problem for GANs as existing training/fitting algorithms require complete observations, so users have had to either first impute the missing data or just discard incomplete observations. However in a recent paper the authors introduced a novel GAN algorithm that can directly train a synthetic data generator from datasets with missing values, which we have called MaWGAN for *Masked Wasserstein GAN* (Poudevigne-Durance et al. 2022). MaWGAN is based on a modification of the Wasserstein distance and is easily implemented by incorporating into the critic masks generated from the pattern of missing data in the original dataset. Moreover we will see that this approach also works in our current setting, so that our approach to modelling the conditional distribution of the present given the past can deal directly with missing data.

The Wasserstein distance or Kantorovich–Rubinstein metric (also called the Earth-Mover distance) is a distance function defined between probability distributions. In the context of a GAN the critic is trained to estimate the Wasserstein distance between the distribution that the data were sampled from and the distribution represented by the generator. Kantorovich & Rubinstein (1958) famously showed that the Wasserstein distance can be written as a Lipschitz metric, which is the form we use (see Section 2.1).

The original GAN effectively used the Kullback-Leibler divergence to measure the distance between data and generator (Arjovsky et al. 2017), however—unlike the Wasserstein distance—this approach is susceptible to the so-called vanishing gradients problem, whereby the critic rejects all samples from the generator and does not allow it to

learn. Other approaches to measuring the distance between data and generator have been proposed, for example Variational Divergence (Nowozin et al. 2016) and Maximum Mean Discrepancy (Li et al. 2017), the latter being a special case of scoring rule minimisation (Pacchiardi et al. 2021).

In what follows we give some background on MaWGAN before describing our two-stage approach for applying GANs to time series. We then provide pseudo-code showing how to incorporate MaWGAN so that the method is applicable to time-series data with missing values. We also give some preliminary test results in which we test the method using data simulated from an auto-regressive AR(3) model. The results are promising and show that the method can cope with missing data, though there is scope for further tuning of the parameters and architecture of the GAN nets (generator, critic and forecaster).

2. METHODOLOGY

2.1 MaWGAN

A basic description of the *Masked Wasserstein GAN* is needed for what follows. See Poudevigne-Durance et al. (2022) for more details. In what follows our vectors are all row vectors by default.

WGAN-GP MaWGAN builds on the WGAN-GP algorithm (Arjovsky et al. 2017, Gulrajani et al. 2017). Let $\mathbf{x}_1, \ldots, \mathbf{x}_n \in \mathbb{R}^d$ be an i.i.d. sample from some (unknown) distribution $\mathcal{P}$, and let $G : (0,1)^d \to \mathbb{R}^d$ be our generator. G takes a vector of i.i.d. $U(0,1)$ random variates and returns a vector with distribution $\mathcal{Q}$ say. The WGAN-GP critic calculates an estimate of the Wasserstein distance, so that the generator is trained to minimise the distance between $\mathcal{P}$ and $\mathcal{Q}$ as measured by the Wasserstein distance.

The Wasserstein distance can be written as

$$W(\mathcal{P}, \mathcal{Q}) = \sup_{\|f\|_L \leq 1} \left(\mathbb{E}_{X \sim \mathcal{P}} f(X) - \mathbb{E}_{Y \sim \mathcal{Q}} f(Y) \right)$$

where $\|f\|_L$ is the Lipschitz constant of f. Let $C : \mathbb{R}^d \to \mathbb{R}_+$ be our critic, let $\mathbf{y}_1, \ldots, \mathbf{y}_n$ be a sample from the generator G, and for $\epsilon_i \sim U(0,1)$ put $\mathbf{z}_i = \epsilon_i \mathbf{x}_i + (1 - \epsilon_i)\mathbf{y}_i$, then we train the critic to maximise

$$\frac{1}{n} \sum_i C(\mathbf{x}_i) - \frac{1}{n} \sum_i C(\mathbf{y}_i) - \lambda \frac{1}{n} \sum_i \left(\|\nabla C(\mathbf{z}_i)\|_2 - 1 \right)^2 .$$

The key idea here is that the regularisation term will restrict the critic C to be close to a Lipschitz function with Lipschitz constant 1. $\lambda > 0$ controls the degree of regularisation and can be tuned to improve the convergence of the critic.

MaWGAN MaWGAN is based on a variation of the Wasserstein distance that incorporates a random mask to capture the effect of missing data. For our purposes a mask $\mathbf{m} = (m_1, \ldots, m_d)$ is an element of $\{0,1\}^d$ and a random mask is just a measure $\mathcal{M}$ on $\{0,1\}^d$. Given a data point $\mathbf{x} = (x_1, \ldots, x_d)$ and a mask $\mathbf{m}$, x_j is treated as missing if and only if $m_j = 0$. We define the $\mathcal{M}$-Wasserstein distance as

$$W_{\mathcal{M}}(\mathcal{P}, \mathcal{Q}) = \sup_{\|f\|_L \leq 1} \mathbb{E}_{M \sim \mathcal{M}} \left(\mathbb{E}_{X \sim \mathcal{P}} f(X \odot M) - \mathbb{E}_{Y \sim \mathcal{Q}} f(Y \odot M) \right)$$

where $\odot$ represents pointwise multiplication. It can be shown that if the data is *Missing Completely At Random* (MCAR, see Rubin (1976) for classifications of missing data) then $W_{\mathcal{M}}$ and W generate the same topology on the space of measures on $\mathbb{R}^d$, so that a sequence of measures $\mathcal{Q}_i$ (representing a sequence of improving generators) will converge to $\mathcal{P}$ w.r.t. the Wasserstein distance if and only if they converge to $\mathcal{P}$ w.r.t. the $\mathcal{M}$-Wasserstein distance.

We approximate the $\mathcal{M}$-Wasserstein distance analogously to the WGAN-GP approach. Let $\mathbf{m}_i$ be the mask corresponding to data point $\mathbf{x}_i$, then using our previous notation, we train the critic to maximise

$$\frac{1}{n} \sum_i C(\mathbf{x}_i \odot \mathbf{m}_i) - \frac{1}{n} \sum_i C(\mathbf{y}_i \odot \mathbf{m}_i) - \lambda \frac{1}{n} \sum_i \left(\|\nabla C(\mathbf{z}_i \odot \mathbf{m}_i)\|_2 - 1 \right)^2 .$$

Here we interpret $\mathbf{x}_i \odot \mathbf{m}_i$ as replacing the missing values in $\mathbf{x}_i$ with zeros, and $\mathbf{y}_i \odot \mathbf{m}_i$ replaces the corresponding values of $\mathbf{y}_i$ with zeros. It is this modified critic that defines the MaWGAN methodology.

2.2 Two-stage GAN model for time-series data

Let $\ldots, X_{-1}, X_0, X_1, \ldots$ be a real-valued stationary time-series with dependency of lag k. That is, X_i is conditionally independent of X_{i-k-j} for $j \geq 1$, conditioned on $(X_{i-1}, \ldots, X_{i-k})$. Let $M_i = 0$ if X_i is missing and 1 if not. We will assume that the M_i are independent of each other and of the X_i (so the X_i are MCAR). Our target is the conditional distribution of $X_i | (X_{i-1}, \ldots, X_{i-k})$. This is completely determined by the joint distribution of $(X_i, X_{i-1}, \ldots, X_{i-k})$, and so our approach is to use MaWGAN to fit a generator G and critic C to this joint distribution—which we can do in the presence of missing values—then use them to train a model for the conditional distribution.

Let $x_1, \ldots, x_n$ and $m_1, \ldots, m_n$ be observations of $X_1, \ldots, X_n$ and $M_1, \ldots, M_n$ respectfully, and put $\mathbf{u}_i = (x_i, \ldots, x_{i+k})$ and $\mathbf{v}_i = (m_i, \ldots, m_{i+k})$ for $i = 1, \ldots, n - k$. Given the $\mathbf{u}_i$ and $\mathbf{v}_i$ we use MaWGAN to train a generator $G : (0,1)^{k+1} \to \mathbb{R}^{k+1}$ and critic $C : \mathbb{R}^{k+1} \to \mathbb{R}_+$. If $\mathbf{z}$ is a vector of independent $U(0,1)$ random variables, then the distribution of $G(\mathbf{z})$ will approximate that of $(X_i, \ldots, X_{i+k})$. Our goal is to train a forecaster:

$$F : \mathbb{R}^k \times (0,1) \to \mathbb{R}.$$

For $Z \sim U(0,1)$ we want $F(x_1, \ldots, x_k, Z) \sim X_{k+1} | (X_1 = x_1, \ldots, X_k = x_k)$. To train F we generate $\mathbf{w} = (w_1, \ldots, w_{k+1})$ from G and z from a $U(0,1)$ then put

$$\mathbf{y} = (w_1, \ldots, w_k, F(w_1, \ldots, w_k, z))$$

Repeat this m times to get a synthetic sample $\mathbf{y}_i$, $i = 1, \ldots, m$, where m is the batch size. The $\mathbf{y}_i$ should look like realisations of $(X_1, \ldots, X_{1+k})$, so we can measure the performance of F by comparing the sample $\mathbf{y}_1, \ldots, \mathbf{y}_m$ to a batch of m of the $\mathbf{u}_i$, and we can do this comparison using the critic C.

As is usual for GANs, we iteratively train F and C, but we don't continue training G. Continuing to train C should encourage the critic to concentrate on the distribution of the last element of the sample given the other k elements, because the training of F will have no effect on the distribution of the first k elements being passed to the critic. Once F is trained we can use it to forecast using the most recent k observations, or generate synthetic series (starting with a single sample from G).

2.3 Pseudo-code

We suppose that for $i = 1, \ldots, n-k$ we have data $\mathbf{u}_i$ and masks $\mathbf{v}_i$, which are contiguous subsequences of length $k+1$ taken from a time-series with missing values. We have a generator $G : (0,1)^{k+1} \to \mathbb{R}^{k+1}$, critic $C : \mathbb{R}^{k+1} \to \mathbb{R}_+$ parameterised by weights $\boldsymbol{\theta}_C$, and a forecaster $F : \mathbb{R}^k \times (0,1) \to \mathbb{R}$ parameterised by weights $\boldsymbol{\theta}_F$. We assume that G and C have already been trained with data $\mathbf{u}_i$ and masks $\mathbf{v}_i$ using MaWGAN.

For any $\mathbf{w} = (\mathbf{w}(*), w(k+1)) \in \mathbb{R}^{k+1}$ we will write $\mathbf{w}(*)$ for the first k elements and $w(k+1)$ for the last element.

Require: forecaster weights $\boldsymbol{\theta}_F$ and critic weights $\boldsymbol{\theta}_C$, learning rates α_F and α_C.
Require: num. epochs t_F, forecaster batch size m_F, critic iterations t_C, critic batch size m_C, critic regularisation λ.

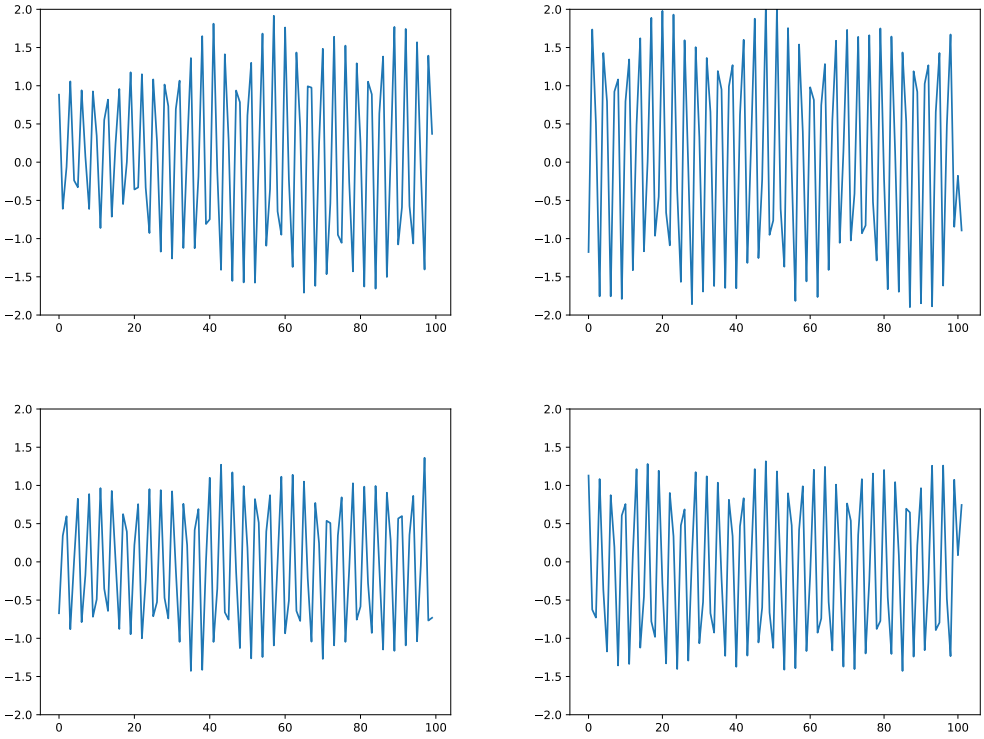
for $s = 1, \ldots, t_F$ **do** ▷ update the forecaster
  for $t = 1, \ldots, t_C$ **do** ▷ update the critic
    choose a batch σ of size m_C from $\{1, \ldots, n-k\}$
    for $i = 1, \ldots, m_C$ **do** ▷ calculate critic loss
      $\bar{\mathbf{u}}_i \leftarrow \mathbf{u}_{\sigma(i)} \odot \mathbf{v}_{\sigma(i)}$
      $\mathbf{w}_i = (\mathbf{w}_i(*), w_i(k+1)) \leftarrow G(\mathbf{a})$ for $\mathbf{a} \sim U(0,1)^{k+1}$
      $w_i(k+1) \leftarrow F(\mathbf{w}_i(*), b)$ for $b \sim U(0,1)$
      $\bar{\mathbf{w}}_i \leftarrow \mathbf{w}_i \odot \mathbf{v}_{\sigma(i)}$
      $\mathbf{z}_i \leftarrow \epsilon \bar{\mathbf{u}}_i + (1-\epsilon)\bar{\mathbf{w}}_i$ for $\epsilon \sim U(0,1)$
      $L_C^i \leftarrow C(\bar{\mathbf{u}}_i) - C(\bar{\mathbf{w}}_i) - \lambda(\|\nabla C(\mathbf{z}_i)\|_2 - 1)^2$
    end for
    $L_C \leftarrow \frac{1}{m_C} \sum_{i=1}^{m_C} L_C^i$
    update $\boldsymbol{\theta}_C$ using gradient of L_C (increasing L_C) and learning rate α_C
  end for
  for $i = 1, \ldots, m_F$ **do** ▷ calculate forecaster loss

16: $\mathbf{w}_i = (\mathbf{w}_i(*), w_i(k+1)) \leftarrow G(\mathbf{a})$ for $\mathbf{a} \sim U(0,1)^{k+1}$
17: $w_i(k+1) \leftarrow F(\mathbf{w}_i(*), b)$ for $b \sim U(0,1)$
18: $L_F^i \leftarrow C(\mathbf{w}_i)$
19: **end for**
20: $L_F \leftarrow \frac{1}{m_F} \sum_{i=1}^{m_F} L_F^i$
21: update $\boldsymbol{\theta}_F$ using negative gradient of L_F (decreasing L_F) and learning rate α_F
22: **end for**

3. TEST CASE

To test the method we used sequences of length 100 generated from an AR(3) model with parameters $(0.1, -0.3, 0.9)$ and error variance 0.1^2. That is $X_n = 0.1X_{n-1} - 0.3X_{n-2} + 0.9X_{n-3} + \epsilon_n$ where the ϵ_n are i.i.d. $N(0, 0.1^2)$. Some typical training sequences are plotted in Figure 1.

Figure 1: *LEFT: Samples of length 100 from an $AR(3)$ with parameters $(0.1, -0.3, 0.9)$ and error variance 0.1^2. RIGHT: Samples of length 100 from a GAN forecaster trained using the sample to the left (with no missing data).*



In Figure 1 we also plot some synthetic data from the models trained using each sample (a separate model was trained for each sample). Qualitatively the GAN forecaster

performs well, capturing both the high and low frequency oscillations in the data. We also observe that a sequence of length 100 is not long enough to capture the full variety of behaviour that the AR(3) process can exhibit. For example in Figure 1 the scale of the output top left is clearly different to the scale of the output bottom left, even though both are realisations of the same process. In both cases the GAN forecaster has got the scale correct, however we only expect the GAN forecaster to synthesise output at the scale to which it has been trained. That is, while the same AR(3) model produced both the upper and lower output on the left, we don't expect the GAN forecaster that produced the top right output to be able to produce the bottom right output, and vice versa. To get a single GAN forecaster that could reproduce both we would need to train it on a sequence long enough that it contained both types of behaviour.

3.1 Performance measures

Quantifying the performance of the GAN forecaster is not straight-forward, as we are not interested in its performance as a forecaster, but rather how well it captures the (temporal) dependence structure of the data. We used two performance measures. The first is the *Likeness Score* L introduced by Guan & Loew (2020). Suppose we have observations $\mathbf{x}_1, \ldots, \mathbf{x}_n$ from some distribution and observations $\mathbf{y}_1, \ldots, \mathbf{y}_m$ from a second distribution, then to calculate L we first generate three auxiliary sets of information

$$\begin{aligned} S_{\mathbf{x}} &= \{\|\mathbf{x}_i - \mathbf{x}_j\|_2\}_{i \neq j} \\ S_{\mathbf{y}} &= \{\|\mathbf{y}_i - \mathbf{y}_j\|_2\}_{i \neq j} \\ S_{\mathbf{x},\mathbf{y}} &= \{\|\mathbf{x}_i - \mathbf{y}_j\|_2\}_{i,j} \end{aligned}$$

For $A, B \subset \mathbb{R}$ let $\kappa(A, B) \in [0, 1]$ be the Kolmogorov-Smirnov distance between A and B, namely the maximum absolute difference between the empirical cumulative distribution functions of A and B. The Likeness Score for our two sets of observations is then

$$L = 1 - \kappa(S_{\mathbf{x}}, S_{\mathbf{x},\mathbf{y}}) \vee \kappa(S_{\mathbf{y}}, S_{\mathbf{x},\mathbf{y}}),$$

where $\vee$ indicates the maximum. Note that $L \in [0, 1]$ and the two sets of observations have likeness one if and only if they are identical, with lower scores indicating greater dissimilarity. To have a likeness of zero the two datasets would need to have disjoint ranges.

In our application the $\mathbf{x}_i$ will always be the $\mathbf{u}_i$ defined in Section 2.2, and the $\mathbf{y}_i$ will be the analogous subsequences taken from a sample of synthetic data generated by a GAN forecaster (the same length as the original sample).

Our second performance measure is more ad-hoc. Let $\boldsymbol{\theta} = (0.1, -0.3, 0.9)$ be the coefficients of our AR(3) process and let $\hat{\boldsymbol{\theta}}$ be the usual maximum likelihood estimate of $\boldsymbol{\theta}$, calculated using a synthetic sample from the GAN forecaster, then our performance measure is just the mean-squared error $M := \|\boldsymbol{\theta} - \hat{\boldsymbol{\theta}}\|_2^2/3$, which we denote the *AR Coefficient Score*. Clearly smaller values are better.

To reduce the variation due to sampling from the generator we calculate L and M 100 times using different samples of synthetic data, then take the average. In what follows we take L and M to be these averaged values. To allow for the variation in performance due to the original sample used and the stochastic nature of the GAN fitting, we generated 30 different samples from the AR(3) process and fitted a GAN to each one.

When estimating the performance of the GAN forecaster there are three sources of variation:

- From the sampling of the data. As this is a simulation experiment we can gauge this by using multiple independent data samples (30 in our case).
- From the sampling of the generator. We mitigate this by comparing each data sample with 100 independent synthetic samples and averaging the results.
- From the fitting of the generator. The process of fitting a GAN is stochastic due to the random selection of observation batches, and for each separate data sample we re-fit the GAN. We mitigate this by using a very large number of training iterations, to be reasonably confident that the GAN has converged.

The variation represented by the confidence intervals in Figures 2 and 3 below is mainly due to the sampling of the data, though is necessarily confounded with the other two sources of variation. The length n of the data samples will also clearly effect the performance of the GAN model, however we do not explore this here and just fix $n = 100$.

3.2 Results

To assess the effect of missing data, for each original AR(3) sequence we generated six auxiliary sequences with increasing levels of missing data: 10%, 20%, ..., 60%. Observations were removed uniformly and independently, so the data is Missing Completely At Random (MCAR). The auxiliary sequences were nested so that all the points missing in one are also missing in those with higher levels of missing data.

For our experiments we took $k = 3$, and the critic and generator both had 1 hidden layer with 100 nodes. For the initial training of the generator and critic we used learning rate $\alpha = 0.0001$; batch size 30; iterations $t_G = 5000$ and $t_C = 10$; and critic regularisation $\lambda = 10$ (using the notation of Poudevigne-Durance et al. 2022). For the training of the forecaster we found that the same parameters worked, namely the learning rates were $\alpha_F = 0.0001$ and $\alpha_C = 0.0001$; batch sizes were $m_F = 30$ and $m_C = 30$; iterations were $t_F = 5000$ and $t_C = 10$; and the critic regularisation was $\lambda = 10$. Some optimisation of these parameters is required, as for any WGAN-GP based algorithm, but this was not done particularly systematically.

The performance of the GAN forecaster is summarised in Figures 2 (Likeness Score) and 3 (AR Coefficient Score). For both figures we give the average performance of the GAN forecaster for different levels of missing data (with 95% confidence intervals). For both measures the performance of the GAN forecaster is not much effected by levels of missingness up to 40%.

Figure 2: *Likeness Scores. Each point is the mean of 30 calculations of the Likeness Score L, comparing a sequence of length 100 from an AR(3) process with synthetic data generated using a GAN forecaster, trained using the AR(3) sample. The error bars give 95% confidence intervals. The level of missing data is the proportion of observations removed at random from the original sample before training the GAN forecaster, though note that the Likeness Score is always calculated using the original data with no missing observations.*

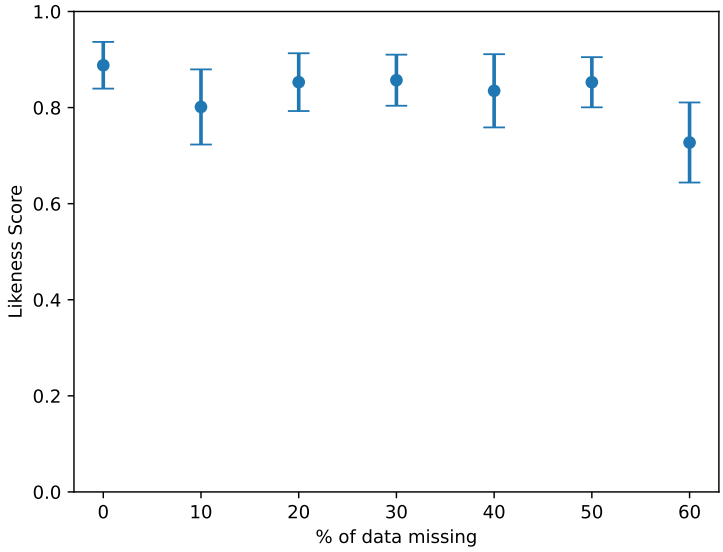
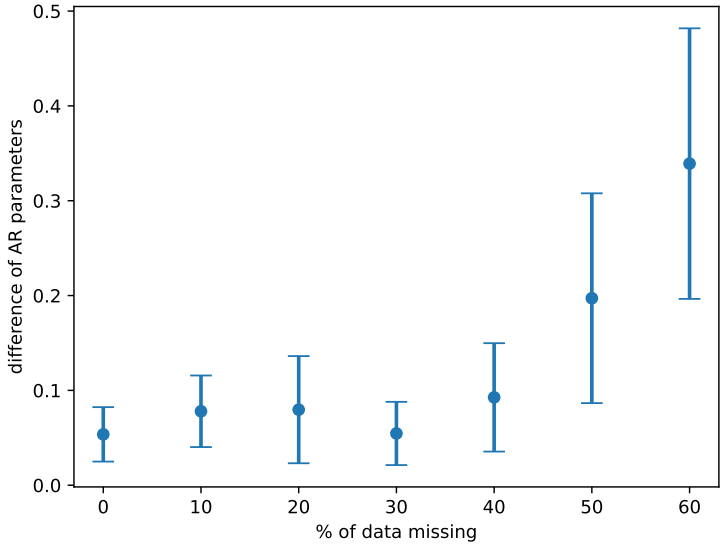


Figure 3: *AR Coefficient Scores. Each point is the mean of 30 calculations of M, the average squared distance between the true parameters of an AR(3) process and the estimated parameters from a synthetic sample generated using a GAN forecaster, trained using a sample of size 100 from the AR(3) model. The error bars give 95% confidence intervals. The level of missing data is the proportion of observations removed at random from the original sample before training the GAN forecaster.*



To have a basis for comparison we fitted an AR(3) process to each of the original samples we generated, and then calculated L and M as above. For the AR Coefficient Score we got a mean of 0.0027 with 95% CI $(0.0012, 0.0042)$, and for the Likeness Score we got a mean of 0.809 and 95% CI $(0.751, 0.867)$. That is, according to the AR Coefficient Score the fitted AR(3) processes gives a better fit, which is to be expected as we are using a correctly specified model, however according to the Likeness Score the GAN forecaster gives a better fit. Possibly this is because the GAN forecaster fits to the sample it is given and the Likeness Score is measuring how well you match that sample, rather than the parameters of the original process. Formally we can interpret this as saying the GAN forecaster is doing a good job "locally" but could do better "globally". As noted at the start of Section 3, a sample of length 100 is probably too short for the GAN forecaster to learn all the possible behaviours that this AR(3) process can exhibit, so better global performance would require a larger data sample. In future work we will compare the performance of our GAN forecaster with the methods of Esteban et al. (2017) and Yoon et al. (2019), in the case of no missing data.

4. DISCUSSION

Our GAN forecaster has produced some promising results, however more testing is required, and there is scope for generalising and tuning the method. At the time of writing our systematic experimentation has been restricted to the case presented in Section 3, however clearly of interest for further investigation is the effect on performance of the length of the data sequence n and the lag k. Moreover our method easily generalises to multivariate time-series, and indeed can be used to model any conditional distribution, so it would also be of interest to see how well it can capture the dependencies of time-series models such as the Periodic Autoregressive or Spatial Autoregressive (see e.g. Holan et al. 2010, LeSage & Pace 2009).

In practice securing the convergence of a GAN requires a balance between the speed at which the generator and critic converge. Our methodology requires the convergence of the generator, critic and forecaster, so we have three things balance. We simplified the interplay of the three nets by training the generator and critic first, then switching to the critic and forecaster, however allowing simultaneous training of all three nets could improve the overall speed at which they converge, and improve their performance, but at the expense of more tuning parameters. We also need to be mindful that the initial training of the critic may mean it is too specialised for the early training of the forecaster, causing it to focus on details rather than broad features. One way of alleviating this problem somewhat is to pause training of the critic while the forecaster "catches up".

It is clear that in practice the choice of lag is important and may not be as straightforward as for our case study. The obvious guideline is that the lag should be large enough to encompass any features in the data, such as seasonal effects or irregular cycles, however the larger the lag the more complicated the forecaster has to be, which translates into more and larger internal layers for all the GAN nets (generator, critic and forecaster), making the fitting slower and more temperamental.

Finally we note that the effect of the dependencies between the $\mathbf{u}_i$ on the convergence of the GAN nets is something that warrants investigation, as is the question of how we use the architecture of these nets to exploit the temporal structure of the $\mathbf{u}_i$. For example dimension-reducing feature layers have proved effective in recurrent neural networks, and may do so here as well (Yoon et al. 2019).

ΠΕΡΙΛΗΨΗ

Στην παρούσα εργασία προτείνεται μια νέα μέθοδος σύνθεσης χρονοσειρών με τη χρήση των Generative Adversarial Networks (GANs) η οποία μπορεί να αξιοποιηθεί και στην περίπτωση ελλειπουσών παρατηρήσεων. Η μέθοδος μοντελοποιεί την δεσμευμένη κατανομή της τρέχουσας παρατήρησης δοθέντων των παρελθόντων παρατηρήσεων, κάτι που επιτυγχάνει με ένα βοηθητικό GAN εκπαιδευμένο στην από κοινού κατανομή της τρέχουσας και των παρελθόντων παρατηρήσεων. Ένα πλεονέκτημα της μεθόδου είναι η χρήση ενός Masked Wasserstein GAN (MaWGAN) για

την εκπαίδευση του μοντέλου το οποίο μπορεί να συνεκτιμήσει και τις ελλείπουσες παρατηρήσεις. Η αποτελεσματικότητα της μεθόδου ακόμα και για μεγάλα ποσοστά ελλιπουσών παρατηρήσεων, επιβεβαιώνεται με ένα πείραμα προσομοίωσης.

REFERENCES

Arjovsky, M., Chintala, S. and Bottou, L. (2017). Wasserstein Generative Adversarial Networks. *Proceedings of the 34th International Conference on Machine Learning, Sydney, Australia, 6–11 August 2017*, pp.214–223.

Campbell, M. (2019). Synthetic data: How AI is transitioning from data consumer to data producer and why that's important. *Computer*, **52**(10):89–91.

Esteban, C., Hyland, S.L. and Rätsch, G. (2017). Real-valued (medical) time series generation with recurrent conditional GANs, *arXiv*:1706.02633.

Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A. and Bengio, Y. (2014). Generative Adversarial Nets. *Advances in Neural Information Processing Systems*, **27**.

Guan, S. and Loew, M.H. (2020). Measures to evaluate Generative Adversarial Networks based on direct analysis of generated images. *arXiv*: 2002.12345.

Gulrajani, I., Ahmed, F., Arjovsky, M., Dumoulin, V. and Courville, A.C. (2017). Improved training of Wasserstein GANs. *Advances in Neural Information Processing Systems*, **30**.

Hitawala, S. (2018). Comparative Study on Generative Adversarial Networks, *arXiv*: 1801.04271.

Holan, S.H., Lund, R. and Davis, G. (2010). The ARMA alphabet soup: A tour of ARMA model variants. *Statistics Surveys*, **4**:232–274.

Jordon, J., Szpruch, L., Houssiau, F., Bottarelli, M., Cherubin, G., Maple, C., Cohen, S.N. and Weller, A. (2022). Synthetic Data–what, why and how?. *arXiv*:2205.03257.

Kaloskampis, I., Pugh, D., Joshi, C. and Nolan, L. (2019). Synthetic data for public good. *ONS Data Science Campus blog.* `https://datasciencecampus.ons.gov.uk/projects/synthetic-data-for-public-good/`

Kantorovich, L.V. and Rubinstein, G.Sh. (1958). On a space of completely additive functions. *Ser. Mat. Mekh. i Astron.*, **13**(7):52–59. (In Russian)

LeSage, J. and Pace, R.K. (2009). *Introduction to spatial econometrics.* Chapman and Hall/CRC.

Li, C.L., Chang, W.C., Cheng, Y., Yang, Y. and Póczos, B. (2017). MMD GAN: Towards deeper understanding of moment matching network. *Advances in Neural Information Processing Systems*, **30**.

Mogren, O. (2016). C-RNN-GAN: Continuous recurrent neural networks with adversarial training, *arXiv*:1611.09904.

Nowozin, S., Cseke, B. and Tomioka, R. (2016). f-GAN: Training generative neural samplers using variational divergence minimization. In *Proceedings of the*