A TimeGAN Application for Generating Time Series Related to Climate Prediction

Ca' Foscari University of Venice
Department of Environmental Sciences, Informatics and Statistics

Computer Science Master's Thesis
Year 2020-2021

Graduand Marco Lavazza Seranto (823162)

Supervisor prof. Sebastiano Vascon
Co-Supervisor prof. Davide Zanchettin

Abstract

The aim of this thesis is to verify the possibility of generating data series (temperature, salinity or other) with decennial visibility starting from simulated series in typical models of climatology, reproducing their conditional distribution. The reason for this research is that once the network is properly trained, the generation of the series is very fast, while the standard simulators take up a lot of machine time. The work therefore aims to provide a fast tool for generating data that can be used in the study of oceanography.

Keywords Generative Adversarial Learning, Time Series, Climate Change, Ocean Temperature Simulation

Acknowledgments

The Earth System Model computations were performed at the German Climate Computer Center (DKRZ). I am grateful to Kameswarrao Modali and Wolfgang Mueller for their help in retrieving the data.
I thank Dr. Eleonora Cusinato for her kind and prompt availability. You have contributed with some sumptuous violin graphs that illustrate the results very well. Thanks Eleonora!
Thanks to my supervisors, Professors Sebastiano Vascon and Davide Zanchettin, for their cordial and constant availability, for the suggestions and support they have given me.
A special thanks goes to Professor Flavio Sartoretto who advised me in choosing the thesis and accompanied me in the first contacts.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Problem Description

> "Oceanography: The science that studies the oceans and seas as a whole and therefore: their chemical-physical properties, their movements and the energy exchange between ocean and atmosphere (physical o.), the organisms that live there (plants and animals) including their ecology (biological o.) and the origin and geological structure of ocean basins, as well as the rocks that constitute them and sediments that settle there (geological o.) ... Physical oceanography attempts to understand, simulate and predict the large and small scale mechanisms of marine circulation and interaction with the atmosphere, coasts and ocean floors."

So says the Treccani Encyclopedia.

Oceanography was practically born with the journey of J. Cook (1772 -1775) and then became established when at the end of the nineteenth century communication cables began to be laid on the bottom of the oceans; it was necessary to know the configuration of the ocean floors in order to arrange a cable of suitable length between the two shores.

These enterprises, with their rare successes and many accidents, taught a lot about the ocean and its characteristics, but above all they made us understand how important was the very difficult understanding of the phenomena that play a role on the expanses of water of our planet.

Research in the field of physical oceanogarphy has developed with various programs since the 1960s: GARP (Global atmospheric research program), WCRP (World climate research program), WOCE, (World ocean circulation experiment), TOGA (Tropical ocean and global atmosphere), connected the oceans with the atmosphere, highlighting the interactions and mutual influences.

The oceans with their masses of water, currents, evaporation and interactions with the atmosphere primarily determine the climate not only of coastal regions but of the whole globe.

Various measurement systems are used for temperature, salinity, oxygen content, electrical conductivity, pressure ... both on the surface and in depth with static buoys anchored on the bottom or with appropriately equipped vessels. Other and more sophisticated instruments allow you to obtain wave height, wind speed, current monitoring, chlorophyll concentrations, traces of oil on the surface ...

This mass of data converges to form the database necessary for the simulation of system models useful for making more or less long-term predictions. The vision of the future of our planet is based on these forecasts and the changes that the climate has undergone are already sensitive today.

By now the climate change issue has become everyday, felt topical and interventions to correct the deviations are often only just good intentions and badly managed.

We must take into account what was said by John Forbes Kerry, secretary of state during the second term of Barack Obama:

> "Climate change is real. The challenge is thrilling. And the longer we wait, the harder it will be to solve the problem. "

This is why climate research and its ability to provide robust and efficient forecasting tools are important.

1.2 Thesis Goal

At the forefront of climate research is the prediction of climate evolution over a time horizon of a decade or so. Decadal climate variability bridges the gap between short-term (sub-seasonal to seasonal) predictions of climate, where initial conditions dominates, and long-term predictions of climate changes, where external forcing (or the boundary conditions) dominates. This therefore requires accounting for all the uncertainties that stem from the combination between external forcing of climate and ongoing internal climate variability. Decadal climate prediction has revitalized the interest on ocean circulation and coupled atmosphere-ocean processes, as it has become clear that it is their understanding which provides the key for successful decadal climate predictions. Decadal climate prediction uses state-of-the-art coupled climate and Earth system models that are softwares that resolve the system of fundamental physical and biogeochemical equations that governs climate. The complexity of these tools brings an intrinsic difficulty for decadal climate predictions, that is the contrast between the large number of simulations required to accurately determine the probability associated to various

possible decadal climate evolutions, and the high computational requirements of climate models.

> "The increase in resolution in MPI-ESM-HR[1] results in a higher demand of the computer resources compared to MPI-ESM-LR. MPI-ESM-HR currently has a throughput of 15 model years per day and thus makes it possible to perform climate simulations. For this, however, the model is run on 106 nodes (each with 36 cores). MPI-ESM-LR is considerably faster and requires only 16 nodes for a throughput of 50 model years per day. The data storage increases by a factor of 5 from MPI-ESM-LR to MPI-ESM-HR considering 6-hourly model output. This is mainly due to the doubling of horizontal resolution and the number of vertical levels used form MPI-ESM-LR to MPI-ESM-HR."[11]

The machine used is Mistral, the high-performance computing system for research on the earth system (HLRE-3), a petascale supercomputer.

The basic idea of this thesis is the use of AI with the aim of training efficient and fast models that allow to generate ten-year climate forecasts in a short time with distributions similar to those of the data provided as input to the neural networks for the training.

[1] MPI-ESM is the Earth System Model developed by the Max Planck Institute for Meteorology.

1.3 Contributions

- Use of a neural network to generate synthetic data relating to climatology
- Use of the Wasserstein loss function in TimeGAN
- Local normalization of the original data

1.4 Outline

Chapter 1 Introduction
It briefly describes the context related to Oceanography and its links with Climatology, the objective of the thesis and the original content of the thesis.

Chapter 2 Background Knowledges
The theory of the fundamental tools of this thesis is described: GAN, timeGAN, and loss functions. The proposed model, used for the tests is then described.

Chapter 3 Experiments
It describes the origin and structure of data used for training, the program settings and the coding of esperiments.

Chapter 4 Results
It describes the system used for testing and it describes the system used for the tests, and reports the results of the tests performed.

Chapter 5 Conclusions and future work
It discusses the results and comparing systems used and future works the possible developments of this work are indicated

Chapter 6 Appendix
It describes the structure of the program and the changes made to use it. As an example, the codes of two programs used for the normalization are shown.

Chapter 2

Background Knowledges

2.1 Generative models

Generative models represent a class of statistical models.

They differ from "discriminatory" models in that:

- "discriminatory" models have the task of establishing whether a certain data belongs to a type of data instance rather than to another.
- "generative" models can generate new data instances.

The difference between the two models lies in the fact that a "discriminatory" model does not capture the probability of a given instance, but rather the probability that a label can be applied to that instance, while a "generative" model seeks the most probable: for example, in models that try to suggest the next word in a sentence, words are assigned a probability associated with the previous sequence.

Given a set of data X and a set of labels Y, the "discriminatory" models capture the conditional probability $P(X \mid Y)$ hat is the probability that the label Y applies to instance X, while in in the case of "generative" models, the model will look for the joint probability $P(X, Y)$ or $P(X)$ if there are no labels.

The interest is aimed at generative models, as the goal is to build new instances, in this case temperature evolutions, having distributions similar to those given as examples during the training periods.

Among the various models of generative networks, the networks proposed by Ian Goodfellow in 2014 are of particular interest for the greater simplicity and adaptability [7]. Compared to other solutions, GANs (Generative Adversial Networks) solve intractability problems of other solutions such as restricted [1] or Deep

[1]Boltzman Machines (BM) are symmetric neural networks introduced in 1985 by G. Hinton and Terry Sejnowsky [8]. Restricted Boltzman Machines (RBM) are BMs in which connection restrictions are placed (there are no connections between neurons of the same level)

Boltzmann machines [2]

2.2 GAN

A GAN essentially consists of two parts:

- a Generator who learns to generate plausible data,
- a Discriminator who learns to distinguish fake generator data from real data.

When training begins, the generator produces random and therefore obviously false data and the discriminator rejects the data as false.

As the training progresses the generator gets closer and closer to producing outputs that can fool the discriminator until it begins to classify the false data as real.

Both the generator and the discriminator are neural networks and the generator is connected directly to the discriminator input. The discriminator classification provides a signal that the generator, through backpropagation, uses to update the weights of its structure. Figure 2.1 describes the scheme

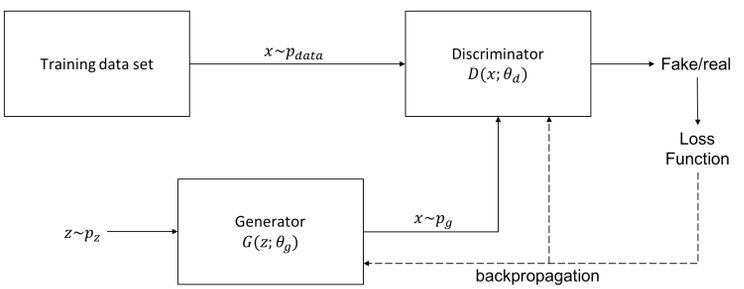


Figure 2.1: Scheme off a GAN

2.2.1 Discriminator

The discriminator is a neural network that acts as a classifier. Its task is to distinguish real data from those created by the generator. In the training phase it uses two data sources: real data, which are the examples that the generator will have to try to replicate, and data from the generator. Real data is used as positive examples in training, while generator instances are used as negative examples. During the training of the discriminator the generator remains unchanged: the weights of its network do not change while it produces data on which the discriminator trains.

[2]Deep Boltzmann Machine or Deep Belief Networks [1][8][12] are multi-layered RBM networks

The discriminator training stages are as follows:

1. The discriminator classifies both real data and (false) generator data
2. The discriminator loss function penalizes the discriminator for errors made (classification of a real instance as false or of a false instance as real)
3. With the backprpagation the discriminator updates the weights of its network

2.2.2 Generator

The generator is trained to create fake instances that can fool the discriminator. The network concerning the generator is composed as follows:

1. random noise generator
2. neural network of the generator that transforms the random noise received in input into a data instance
3. neural network of the discriminator that classifies the instance
4. generator loss function

Generator: since GAN has the task of generating completely new data instances, we have to supply it with noise as input. The neural network of the generator will transform this noise into an instance with the intended meaning. Experimentally, it has been seen that the distribution of the noise source is not very important, therefore a uniform distribution is usually adopted.

Classification of the discriminator: the loss of the generator is produced by the discriminator, therefore the backpropagation passes through both the discriminator and the generator, it does not modify the discriminator but only the generator in order not to change the discriminator during the training of the generator.

The generator training sequence is therefore as follows:

1. Random noise sample
2. Random noise transformation
3. Classification of the discriminator in "true" or "false"
4. Calculation of the loss from the discriminator classification result
5. Backpropagation through discriminator and generator to obtain gradients
6. Using gradients to change the generator network only

2.2.3 GAN training

The Gans contain two separately trained networks and this poses some difficulties.

To solve them, an alternating training system is adopted:

1. the discriminator trains for some epochs (one or more)

2. the generator trains for a few eras (one or more)

3. phases 1 and 2 are repeated several times in order to train the discriminator and generator networks

During the discriminator training phases the generator does not vary its weights. Similarly, the discriminator does not vary its weights during generator training.

In this way the discriminator begins to train with simple instances: the generator provides practically random data easily distinguishable from real instances.

As the training proceeds, the generator becomes more and more capable of producing instances very close to the real ones and the discriminator begins to make mistakes more and more until it produces a completely random feedback obtaining an accuracy of 50%: in practice the answer (true or false) is completely random[6].

Loss Function In the original introductory document of GAN networks[7] the author proposes the following value function

$$\min_{G}\max_{D} V(D,G) = \mathbb{E}_{x\sim p_{data}}\left[\log D(x) + \mathbb{E}_{z\sim p_z}\log\left(1 - D(G(z))\right)\right] \tag{2.1}$$

that the generator tries to minimize while the discriminator tries to maximize, where:

- $D(x)$ is the estimate of the discriminator of the probability that the real data instance x is real

- $G(z)$ is the expected value on all instances of generated data (false)

- $D(G(z))$ is the discriminator's estimate of the probability that a false instance is real

- E_z is the expected value on all random generator inputs

It is a minimax match between D and G and derives from the cross-entropy between the real distribution and the generated one. Actually, since the generator cannot affect $D(x)$, in order to minimize the loss the generator must minimize

$$\log(1 - D(G(z)))$$

.

> "In practice, equation 1 may not provide sufficient gradient for G to learn well. Early in learning, when G is poor, D can reject samples with high confidence because they are clearly different from the training data. In this case, $\log(1 - D(G(z)))$ saturates. Rather than training G to minimize $\log(1 - D(G(z)))$ we can train G to maximize $\log(D(G(z)))$. This objective function results in the same fixed point of the dynamics of G and D but provides much stronger gradients early in learning. "[7]

Mode collapse: it is a problem that occurs

> "when the generator learns to map several different input z values to the same output point"[6]

To avoid this inconvenience, in 2017 a Gan with a loss measured over the Wassertein distance is proposed[2]. In this case the discriminator tries to maximize the so-called "critical loss": $D(x) - D(G(z))$, while the generator will try to maximize the term $D(G(z))$. This type of loss minimizes the problems involved in mode collapse and also those of the 'vanishing gradient' that can occur when a discriminator become too good and does not provide enough information to generator to improve.

Conclusions

GANs represent an interesting generative model useful for producing samples derived from a data distribution:

> "... many tasks intrinsically require realistic generation of samples from some distribution."[6]

In our case something more is needed:

> "A model is not only tasked with capturing the distributions of features within each time point, it should also capture the potentially complex dynamics of those variables acrosstime."[13]

To do this, the "Time-series Generative Adversarial Networks" (TimeGan) have been introduced which we will examine in the next section.

2.3 TimeGAN

Introduced in 2019 at the 33rd Conference on Neural Information Processing Systems (NeurIPS 2019), Vancouver, Canada by Jinsung Yoon and others, TGANs seek to solve the problem by capturing the potentially complex dynamics of time variables.

> "Specifically, in modeling multivariate sequential data $X_{1:T} = (X_1, ..., X_T)$ we wish to accurately capture the conditional distribution $p(X_t|X_{1:t-1})$ of temporal transitions as well." [13]

The problem can be defined as follows: generally the data have a static component, i.e. characteristics that do not vary over time, and a dynamic component, formed by data that change over time: for example, in the case of personal data, the Name , the Surname, the sex are static, the age, the state of health are dynamic ... We call S the vector space containing the static characteristics and X the vector space of the dynamic characteristics and let $\mathbf{S} \in \mathcal{S}$ and $\mathbf{X} \in \mathcal{X}$ two random vectors whose specific values are denoted by s and x. In a certain time interval T we can consider the set

$$(S, X_{1:T})$$

to which a joint probability p is associated, which also includes the random variable T; we therefore call the set of training data as

$$D = (s_n, x_n)_1^{T_n}$$

. What we want to achieve is to produce from the training data, the best approximation

$$\hat{p}\left(S, X_{1:T}\right)$$

to the original density

$$p\left(S, X_{1:T}\right)$$

. This is difficult to achieve with a standard GAN structure.

> "Therefore we additionally make use of the autoregressive decomposition of the joint
>
> $$p\left(S, X_{1:T}\right) = p\left(S\right) \prod_t p\left(X_t \mid S, X_{1:t-1}\right)$$
>
> to focus specifically on the conditionals, yielding the complementary—and simpler—objective of learning a density
>
> $$\hat{p}\left(X_t \mid S, X_{1:t-1}\right)$$

that best approximates

$$p(X_{t|}S, X_{1:t-1})$$

at any time t."[13]

Two distinct objectives arise from this decision. The first can be expressed with:

$$\min_{\hat{p}} D\left(p\left(S, X_{T-1}\right) \parallel \hat{p}\left(S, X_{T-1}\right)\right) \tag{2.2}$$

where D is some measure of the distance between the two distributions, and is global. The second one:

$$\min_{\hat{p}} D\left(p\left(X_t \mid S, X_{1:t-1}\right) \parallel \hat{p}\left(X_t \mid S, X_{1:t-1}\right)\right) \tag{2.3}$$

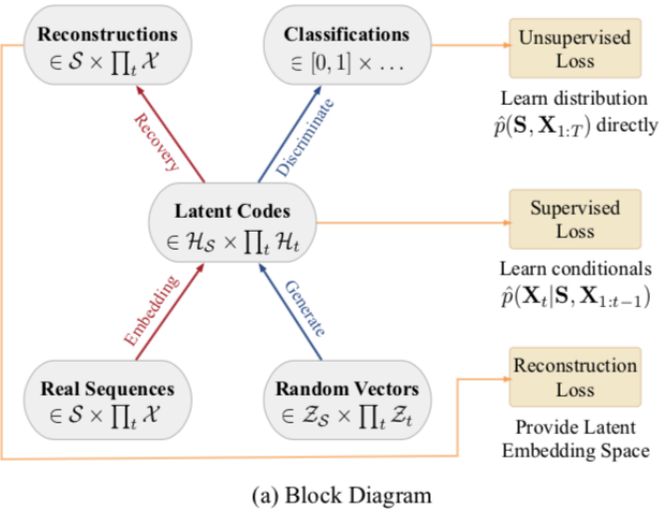
which instead locally measures the point-to-point distance between the original sequence at the moment t and the generated one. From here we configure the objective that combines the GAN, linked to the expression (2) which is configured as the Jensen-Shannon divergence and a supervision training through the maximum-likelihood (ML) which uses the original data and which is configure as the divergence of Kullbach-Leiber, linked to expression (3).

2.3.1 Implementation

The TimeGAN structure consists of four network components:

> "an embedding function, recovery function, sequence generator, and sequence discriminator. The key insight is that the autoencoding components (first two) are trained jointly with the adversarial components (latter two), such that TimeGAN simultaneously learns to encode features, generate representations, and iterate across time. The embedding network provides the latent space, the adversarial network operates within this space, and the latent dynamics of both real and synthetic data are synchronized through a supervised loss."[13]

Figure 2.2 shows the block diagram of TimeGAN.



(a) Block Diagram

Figure 2.2: Block diagram of a TimeGAN[13]

Let's briefly describe the four components.

Embedding and Recovery Functions

The two components of Embedding and Recovery are autoencoders[9] which have the purpose of conveniently mapping the latent space, so that the adversarial network can learn the temporal relationships between data through dimensional reduction. Recalling that the specific values for the static and dynamic characteristics are s and x respectively, and calling h_S and h_t the components of the latent spaces (static and temporal) the embedding network will be a (recurring) network performs the following operations :

$$h_S = e_S(s), \quad h_t = e_{\mathcal{X}}(h_S, h_{t-1}, x_t) \tag{2.4}$$

e_S and e_t are two networks, the first for static functions, the second for temporal functions. The recovery network will have to reconstruct the static $\bar{s}$ and temporal $\bar{x}_{1:T}$ characteristics implemented with two networks r_S and $r_{\mathcal{X}}$:

$$\bar{s} = r_S(h_S), \quad \bar{x}_t = r_{\mathcal{X}}(h_t) \tag{2.5}$$

realized through feedforward networks.

Sequence Generator and Discriminator

Here, too, both the generator and the discriminator operate on static and dynamic data. For the generator:

$$\hat{h}_S = g_S\left(z_S\right) \quad \hat{h}_t = g_{\mathcal{X}}\left(\hat{h}_S, \hat{h}_{t-1}, z_t\right) \tag{2.6}$$

where g_S is a generator network for static functions and $g_{\mathcal{X}}$ a recurring network for temporal functions.

> "Random vector z_S can be sampled from a distribution of choice, and z_t follows a stochastic process; here we use the Gaussian distribution and Wiener process."[13]

For the discriminator:

$$\tilde{y}_S = d_S\left(\hat{h}_S\right) \qquad \tilde{y}_t = d_{\mathcal{X}}\left(\overleftarrow{u}_t, \overrightarrow{u}_t\right) \tag{2.7}$$

> where $\overrightarrow{u} = \overrightarrow{c}_{\mathcal{X}}\left(\hat{h}_{\mathcal{S}}, \hat{h}_t, \overrightarrow{u}_{t-1}\right) and \overleftarrow{u}_t = \overleftarrow{c}_{\mathcal{X}}\left(\hat{h}_{\mathcal{S}}, \hat{h}_t, \overleftarrow{u}_{t+1}\right)$ respectively denote the sequences of forward and backward hidden states, $\overrightarrow{c}_{\mathcal{X}}, \overleftarrow{c}_{\mathcal{X}}$ are recurrent functions, and $d_S, d_{\mathcal{X}}$ are output layer classification functions."[13]

Training scheme

For the training we first have the "reconsruction loss" with which we train the emedding and reconstuction networks for the reconstructions $\tilde{s}, \tilde{X}_{1:T}$ of the original data $s, X_{1:T}$ from latent representations $h_S, h_{1:T}$

$$\mathcal{L}_R = \mathbb{E}_{s, X_{1:T} \sim p}\left[\left\|s - \tilde{s}\right\|_2 + \sum \left\|X_t - \tilde{X}_t\right\|_2\right] \tag{2.8}$$

Secondly, we have the unsupervised loss typical of GAN in which the probability of giving correct classifications is maximized for the discriminator, or minimized for the generator.

$$\mathcal{L}_U = \mathbb{E}_{s, X_{1:T} \sim p}\left[\log y_S + \sum_t \log y_t\right] + \mathbb{E}_{s, X_{1:T} \sim p}\left[\log\left(1 - \hat{y}_S\right) + \sum_t \log\left(1 - \hat{y}_t\right)\right] \tag{2.9}$$

In order to obtain a greater adherence to the conditional distributions of the data, a further loss is introduced which trains the generative network in supervised mode in which the generator receives sequences derived from the embedding of real data in order to generate the next vector.

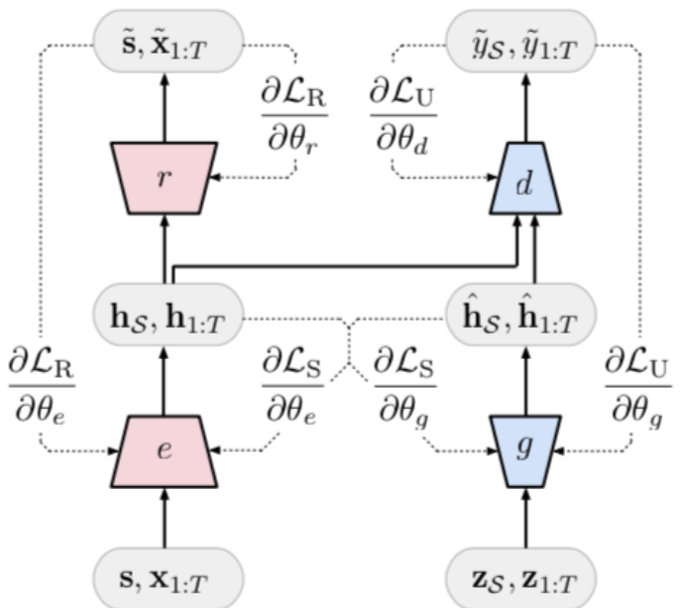
"Gradients can now be computed on a loss that captures the discrepancy between distributions $p(H_t \mid H_S, H_{1:t-1})$ and $\hat{p}\left(H_t \mid H_S, H_{1:t-1}\right)$ (supervised loss)"[13]

$$\mathcal{L}_S = \mathbb{E}_{s, X_{1:T} \sim p}\left[\sum_t \left\| h_t - g_X\left(h_S, h_{t-1}, z_t\right)\right\|\right] \tag{2.10}$$

During training, the difference between the latent vector provided at the next step by the embedding network and the latent vector generated synthetically by the generator is evaluated. $\mathcal{L}_U$ checks that the generator produces plausible sequences for the discriminator, while $\mathcal{L}_S$ checks the gradualness of the transactions.

Optimization

Based on fig. 2.3 consider the parameters of the four networks and are: θ_e the parameters of the embedding network, θ_r the parameters of the recovery network, θ_g the parameters of the generator and θ_d of the discriminator.



(b) Training Scheme

Figure 2.3: b) Training scheme; solid lines indicate forward propagation of data, and dashed lines indicate backpropagation of gradients[13]

The training function for the first two networks uses both reconstruction loss

and supervised loss via a $\lambda \geq 0$ hyperparameter that balances the two losses.

$$\min_{\theta_e, \theta_r} (\lambda \mathcal{L}_S + \mathcal{L}_R) \tag{2.11}$$

The training function for the generator and discriminator networks uses both the reversal of the loss via a hyperparameter $\eta \geq 0$ to balance the two losses.

$$\min_{\theta_g} \left(\eta \mathcal{L}_S + \max_{\theta_d} \mathcal{L}_U \right) \tag{2.12}$$

In practice, TimeGAN is not particularly sensitive to hyperparameters: for all experiments the following values have been set: $\lambda = 1$ and $\eta = 10$

2.4 Proposed model

The original code found at the following link: `https://bitbucket.org/mvdschaar/mlforhealthlabpub/src/master/alg/timegan/`
[13]
was used for the tests. The proposed model was created in Python 3.7 with the following open source software libraries:

- numpy version >= 1.17.2
- tensorflow version == 1.15.0
- tqdm version >= 4.36.1
- argparse version >= 1.1
- pandas version >= 0.25.1
- scikit-learn version >= 0.21.3
- matplotlib version >= 3.1.1

with the following standard settings:

embedder:

Embedding network between original feature space to latent space.
Args: - X: input time-series features - T: input time information
Returns: - H: embeddings
hidden_dim = 24
num_layer = 3
activation_fn = sigmoid

module_name = gru

recovery:

Recovery network from latent space to original space.

Args: - H: latent representation - T: input time information

Returns: - X_tilde: recovered data

hidden_dim = 24

num_layer = 3

activation_fn = sigmoid

module_name = gru

generator:

Generator function: Generate time-series data in latent space.

Args: - Z: random variables - T: input time information

Returns: - E: generated embedding

hidden_dim = 24

num_layer = 3

activation_fn = sigmoid

module_name = gru

discriminator:

Discriminate the original and synthetic time-series data.

Args: - H: latent representation - T: input time information

Returns: - Y_hat: classification results between original and synthetic time-series

hidden_dim = 24

num_layer = 3

activation_fn = sigmoid

module_name = gru

2.5 Loss functions

In general, a loss function measures the degree of accuracy with which a certain statistical model describes a set of empirical data about a certain phenomenon. The goal of a machine learning process is to create an accurate model of a reality. To evaluate its effectiveness and performance, we use a loss function. When working on a Machine Learning or Deep Learning problem, the loss functions are used to optimize the model during training. The goal is almost always to minimize the loss function. The smaller the loss, the better the model.

2.5.1 Cross-entropy

[5] Since in a classification problem we get a value that indicates the probability that a given sample belongs to a specific class or not, we want to increase the degree of accuracy of the model as much as possible. Considering the discrete case, the cross-entropy loss function is:

$$L_{CE} = -\sum_{i=1} p(x) \log q(x) \tag{2.13}$$

where $p(x)$ e $q(x)$ are two probability distributions. Cross-entropy is used to correct model weights during training. In this case the formula (13) becomes:

$$L_{CE} = \sum_{i=1}^{n} t_i \log(p_i) \tag{2.14}$$

in cui t_i is the truth value relative to the i-th class (0,1) and p_i the probability value that leaves the network for the i-th class. For each epoch the value of L_{CE} must decrease and the weights are adjusted with the backpropagation. The optimization process (adjusting the weights so that the output is close to real values) continues until the training is finished.

2.5.2 Wasserstein loss function

The Wasserstein loss function solves the problem of low or zero gradients. It is based on the Wasserstein distance or Earth mover's distance metric between two probability distributions. Interpreting the two probability distributions as piles of earth, the Wasserstein distance is the minimum cost necessary to transform one pile into another. The two piles must have the same amount of land and the cost is the amount of land to be moved to turn one pile into the other. WGANs are GANs where the loss function is the Wasserstein loss function[2]. Wasserstein GANs are less vulnerable to freezing than minimax-based GANs and avoid problems with null gradients. Earthmoving distance also has the advantage of being a true metric: a measure of distance in a space of probability distributions. Cross-entropy is not a metric in this sense. The loss function can be implemented by multiplying the expected label for each sample by the predicted score (element-wise), then calculating the mean.

Below is the python code related to the Wasserstein loss function.

```
from keras import backend
def wasserstein_loss(y_true, y_pred):
  return backend.mean(y_true * y_pred)
```

Chapter 3

Experiments

3.1 Datasets

The data are taken from the preoperational MiKlip system for decadal climate predictions.

The first phase of MiKlip was funded by the Federal Ministry of Education and Research of Germany (https://www.fona-miklip.de/research/miklip-i-miklip-first-phase/)

The MiKlip system[10] is based on the high-resolution version of the Max Planck Institute - Earth System Model (MPI-ESM1.2-HR)[11]. MPI-ESM1.2-HR is a conglomerate of a coupled general circulation model and subsystem models for land and vegetation, and biogeochemistry. In MPI-ESM1.2-HR, the atmospheric general circulation model ECHAM6.3 uses a T127/ 100 km horizontal resolution and 95 hybrid sigma pressure levels that extend up to 0.01hPa; the ocean-sea ice model MPIOM[9] features a tripolar grid with an eddy-permitting global resolution of 0.4° with 40 z-levels. MPI-ESM-HR and similar configurations of MPI-ESM have been widely tested and used in studies of climate dynamics and variability.

We use the MiKlip simulations contributing to the dcppA experiment of the Decadal Climate Prediction Panel (DCPP)[3] . The simulations include an historical assimilation run covering the period 1958/11-2018/11 and hindcasts initialized on November 1st of each year between 1960 and 2018 and using historical forcing. Each hindcast[1] consist of five ensemble members differing in the initial state (r1,. . . ,r5). The data are available from the Earth System Grid Federation An AMV index is calculated from monthly MPIOM output as spatially-averaged North Atlantic sea-surface temperature data over the domain spanning 0–60°N

[1]n oceanography and meteorology, backtesting is also known as hindcasting: a hindcast is a way of testing a mathematical model; researchers enter known or closely estimated inputs for past events into the model to see how well the output matches the known results.[]

latitude and 0°–80°W longitude. Before the index calculation, systematic model errors including drift and biases are removed from grid-point data according to the DCPP guidelines.

3.1.1 Data structure

The data provided is presented as fifty-nine 122x5 matrices as can be seen in the following figure **??**:

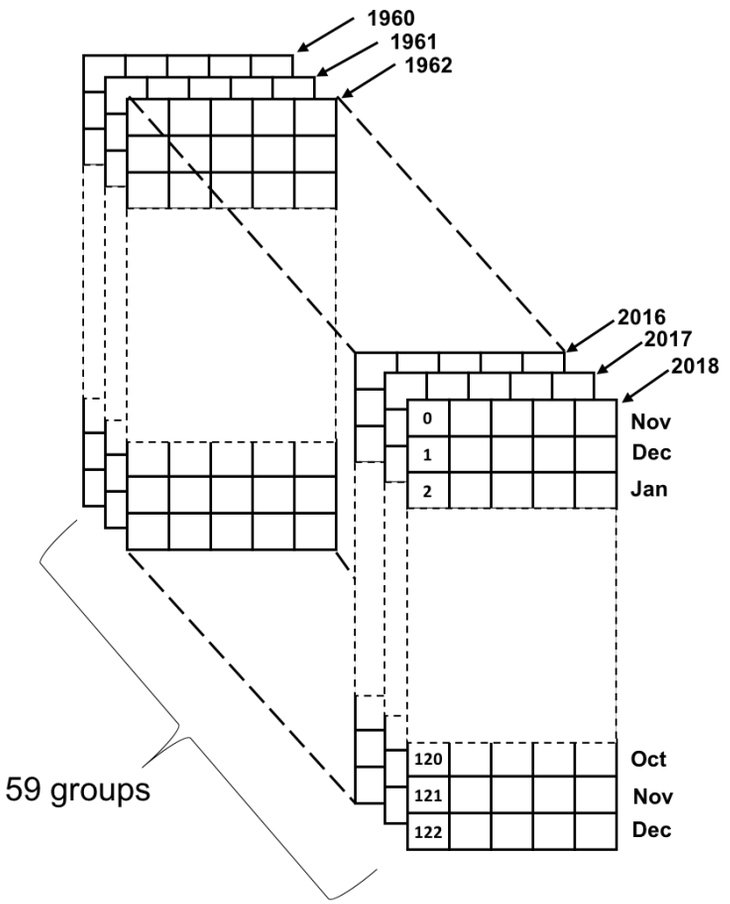


Figure 3.1: Input data structure

Each matrix consists of 5 simulations of North Atlantic sea temperatures. Each column carries the average monthly temperatures starting from November of the year indicated for 10 years: the first matrix carries 5 simulations from November 1960 to December 1970, the second from November 1961 to December 1971 and so on up to the last one covering the period from 2018 to 2028.

Temperatures range from 17,857 ° C (minimum) to 24,271 ° C (maximum). The average temperatures are: 20,881 ° C with a standard deviation of 1,615 ° C. The data, before being given as input to the program, have been normalized. Two different normalizations were initially tested:

1. Scaling or unity-based feature applied to the entire data set: $\bar{x} = \frac{x - \min}{\max - \min}$
2. Standard score applied to the entire set of data: $\bar{x} = \frac{x - \mu}{\sigma}$

The two normalizations produced unsatisfactory results: the shape of the curves was correct, but did not allow to capture the growth dynamics of the input data, providing results with completely flat averages.

Therefore, the same normalizations applied not to the whole data set but locally to each single column of the matrices were applied. Section 4.3 Settings describes the abbreviations relating to normalizations.

3.2 Evaluation Protocols

3.2.1 Settings

The program was set as follows:

- module: gru (original)
- num_layer: 3 (original)
- hidden_dim: 24 (original) or 50
- Loss function (for discriminator): Cross entropy loss (original) (CE) or Wasserstein loss (W)

The training data was normalized a priori with four types of normalization functions:

- Normalization functions:
  - none (original)
  - Unity-based (G) $\Longrightarrow \bar{x} = \frac{x - \min}{\max - \min}$
  - Unity-based local (S) $\Longrightarrow \bar{x}_{loc} = \frac{x_{loc} - \min_{loc}}{\max_{loc} - \min_{loc}}$ (we calculate the values on each column)
  - Standard score (Ss) $\Longrightarrow \bar{x} = \frac{x - \mu}{\sigma}$
  - Standard score local (SsL) $\Longrightarrow \bar{x}_{loc} = \frac{x_{loc} - \mu_{loc}}{\sigma_{loc}}$ (we calculate the values on each column)

3.2.2 Experiment coding

Experiments are performed with the following settings.

Normalization	N	Ss	SsL	S	G	N	Ss	SsL	S	G
Loss	CE	CE	CE	CE	CE	W	W	W	W	W
hidden_dim	24	24	24	24	24	50	50	50	50	50

Table 3.1: Coding experiments

The generated file has the name created as follows: Dgen_X_Y_xxHD where X the type of loss (CE or W), Y represents the normalization type (Ss, SsL, G or S), and xx (24 or 50) a digit representing the number of hidden_dimensions (HD). The folder that stores the trained model will have the name created as follows: str_X_Y_xxHD with the same conventions already described. For example the name of a file generated with loss = Cross-entropy, Normalization = Standard score and with hidden_dim = 24 is:
Dgen_CE_Ss_24HD.npy
and the folder where the structure of model is saved is called:
str_CE_Ss_24HD.
From each trained model 100 samples were created in an array of size 100x59x122x5, stored in a file that is named as the originally originated file with the addition of the digit 100 before the extension .npy. In order to then compare the results with the original, the array is renormalized, applying the inverse function used to normalize using the previously stored scale values.

- Re-normalization:
  - none (original)
  - Unity-based (G):$x = \bar{x} \cdot (\max - \min) + \min$
  - Unity-based local (S): $x_{loc} = \bar{x}_{loc} \cdot (\max_{loc} - \min_{loc}) + \min_{loc}$
  - Standard score (Ss): $x = \bar{x} \cdot \sigma + \mu$
  - Standard score local (SsL): $x_{loc} = \bar{x}_{loc} \cdot \sigma_{loc} + \mu_{loc}$

Chapter 4

Results

The tests were carried out with the code described and, in the first instance, the original settings were kept (see section 4.3.1 Settings).

The code also provided for being able to change modules.Gru, lstm and lstmLN were available.

Once the model had been trained, it was necessary to produce a series of samples on the basis of the saved parameters, on which statistical analysis could be carried out. Unfortunately, with the two modules lstm and lstmLN the restore program gave serious system errors (probably due to a tensorflow bug); therefore the tests were made only with the gru module.

A Gated Recurent Unit (gru) is a recurrent neural network introduced in 2014[4] that solves the vanishing gradient problem. To solve the vanishing gradient problem, GRU uses two vectors called "update gate" and "reset gate" which have the task of establishing which information must be passed to the output.

4.1 Test environment

The tests were performed on a MacBook pro laptop with macOS High Sierra version 10.13.6 operating system and 8 GB of ram.

For each test, 1000 iterations were performed to train the model. Initially, tests were carried out with more than 1000 epochs (2000, 5000), without obtaining particular improvements in the results.

The time it took to train each model was just over an hour, while it took less than a minute to generate 100 samples from the trained model. Initially the program was tested with the original data proposed by the author. Having verified the functioning of the code, we started with the original datasets, the real tests started.

4.1.1 Without external standardization

The program foresees a pre-treatment of the data, for which the original data and with all the original settings were used as a first test.

It is interesting to see how the generated data develop over time. In the following figure 4.1 you can see the result of the data generated as the training periods increase. In the upper part there is the trend (annual for ten years) of a simulation (the first of the year 1960), in the lower part there is the evolution of the generated curve.

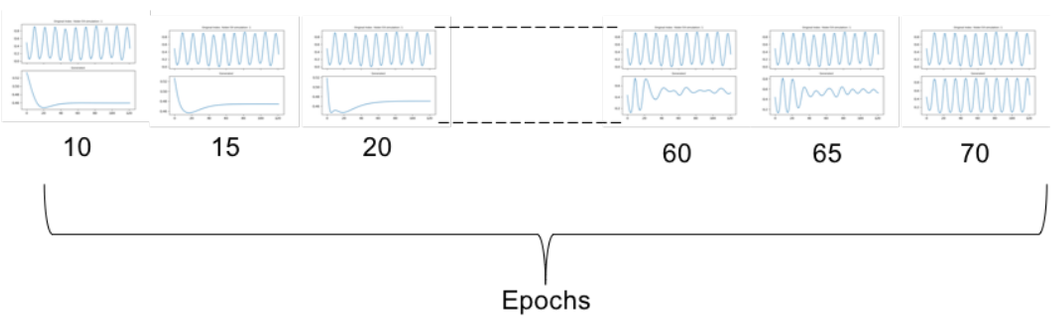


Figure 4.1: Development of the generated data

The following figure 4.2 shows the result of the processing. The red curves represent the average values per year of the original data, the black curves the average values per year of the generated data (500), the green curves the average values per column of the original data and the blue curves the same values for the generated data.

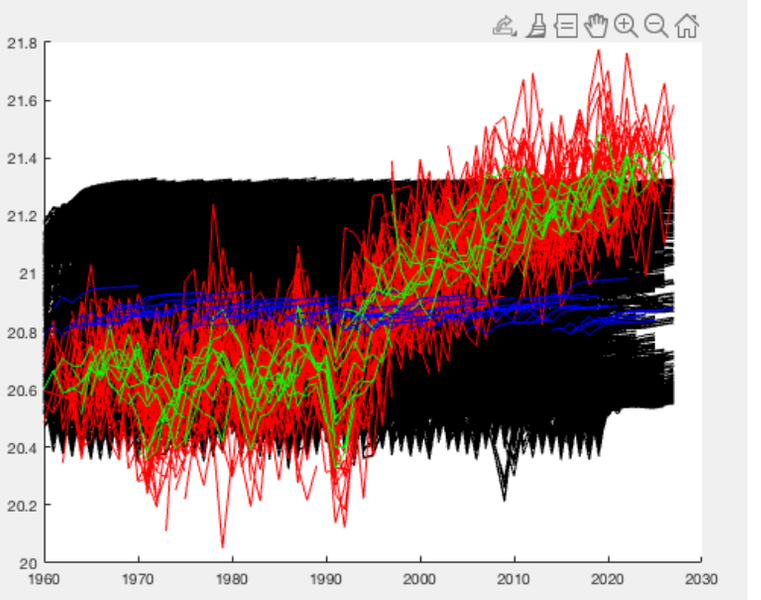


Figure 4.2: Results for CE_N_24HD

As you can see, the result does not copy the trend of the red curves: the averages of the generated data are flat and the growth dynamics are not captured.

4.1.2 Without external normalization with Wasserstein loss

The previous result suggested that the system had entered collapse mode, that is, that the generator is stuck in a local minimum. To avoid this it is suggested to use the Wasserstein distance as the loss function. For this reason the program has been modified to allow the use of the standard function loss (cross entropy: ce) or the Wasserstein loss (w). The python library, keras, gives the possibility to build this function with a few lines of code. The results can be seen in the following figure 4.3.

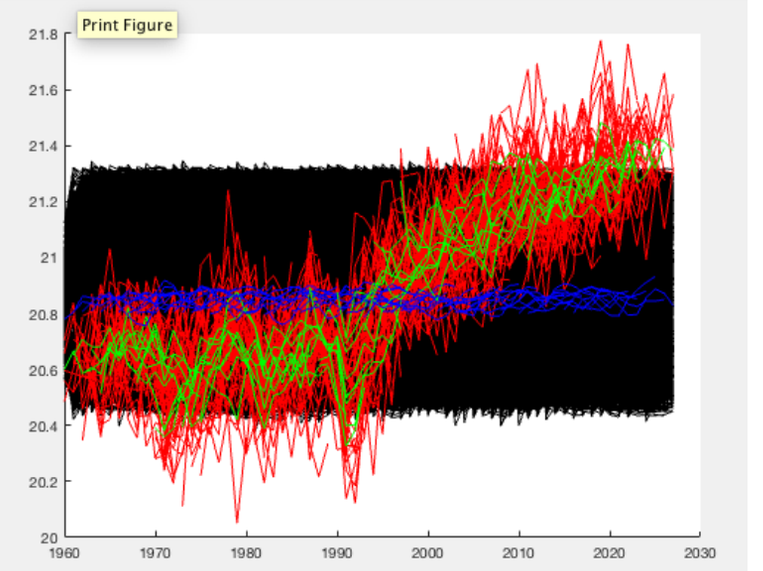


Figure 4.3: Results for W_N_24HD

Also in this case the growth dynamics are not captured.

4.1.3 With external global normalization

This result suggested that the type of normalization adopted was not useful for capturing the temporal dynamics of the data.

The following programs have therefore been created to perform the proposed normalizations (see section 4.3.1 Settings):

- rescaler_g.py: for Unity-based normalization on all data. The values necessary for the renormalization are stored in the two-dimensional MaMi array in which MaMi [0] = min (date) and MaMi [1] = max (data) and saved in the MaMi.npy file

- rescaler_Ss.py: for Standard-score normalization on all data. The values necessary for the renormalization are stored in the two-dimensional Stat array in which Stat [0] = mean (data) and Stat [1] = std (data) and saved in the Stat.npy file

The appendix shows the rescaler_g.py code as an example.

"Global" normalizations, ie on all data were therefore tested, . The figure shows 4.4 the result of one of the tests carried out which, however, always show the same unsatisfactory trend.

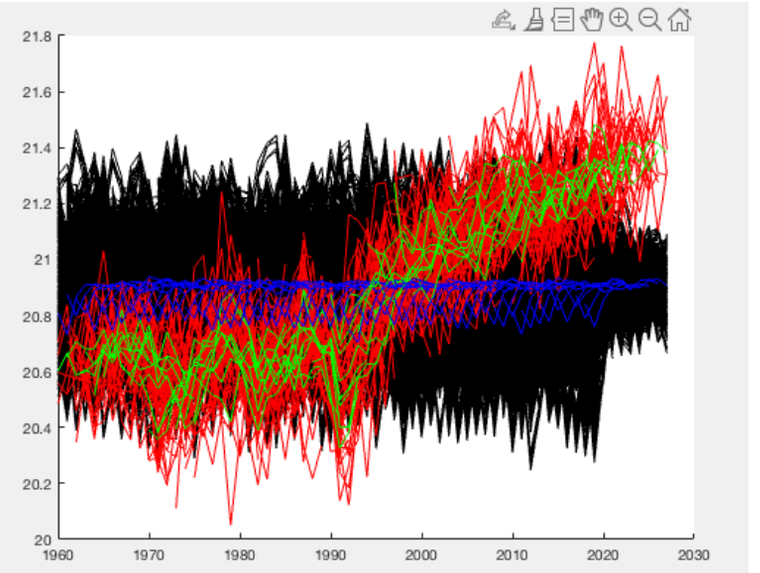


Figure 4.4: Results for CE_G_24HD

4.1.4 With local external normalization

Since even the "global" normalizations did not give acceptable results, it was thought that the choice of parameters referring to the whole data set could somehow mask the dynamics of the original data by flattening the results around the values used in the normalization. Two other programs were then created to locally normalize the data (see section 7 Settings):

- rescaler.py: for Unity-based normalization on each column. The values necessary for the renormalization are stored in the mmsMat array of dimensions (59x2x5) where mmsMat [x, 0, y] = min (dataMat [x,:, y]) and mmsMat [x, 1, y] = max (dataMat [x,:, y]) and saved in the mmsMat.npy file

- rescaler_ls.py: for Standard-score normalization on each column. The values necessary for the renormalization are stored in the meanStd array of dimensions (59x2x5) where meanStd [x, 0, y] = mean (dataMat [x,:, y]) and meanStd [x, 1, y] = std (dataMat [x,:, y]) and saved in the meanStd.npy file

The appendix shows the rescaler.py code as an example. Trials with locally normalized data immediately yielded encouraging results.

The following figure 4.5 shows the result of a test performed with the Wasserstein loss function and Unity-based normalization.

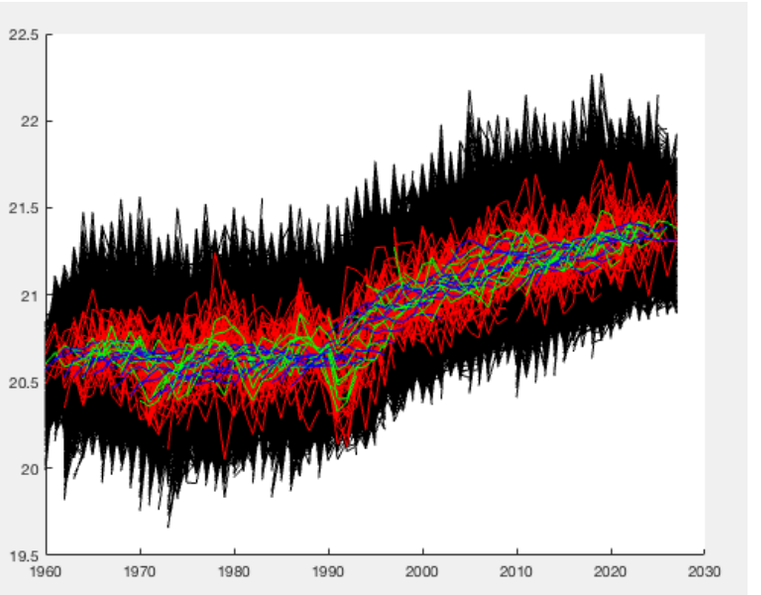


Figure 4.5: Results for W_S_24HD

As it is easy to see, compared to the previous results, in this case the generated data try to adapt to the dynamics of the original data while having a much larger variation on the average values. In the next figure 4.6 we see a different representation: in red there are the average values calculated by column, of the original data, in black there are the generated ones. Each column shows 10 years. The first box shows the column averages of the first 5 years, the second box shows the averages of the second 5 years.

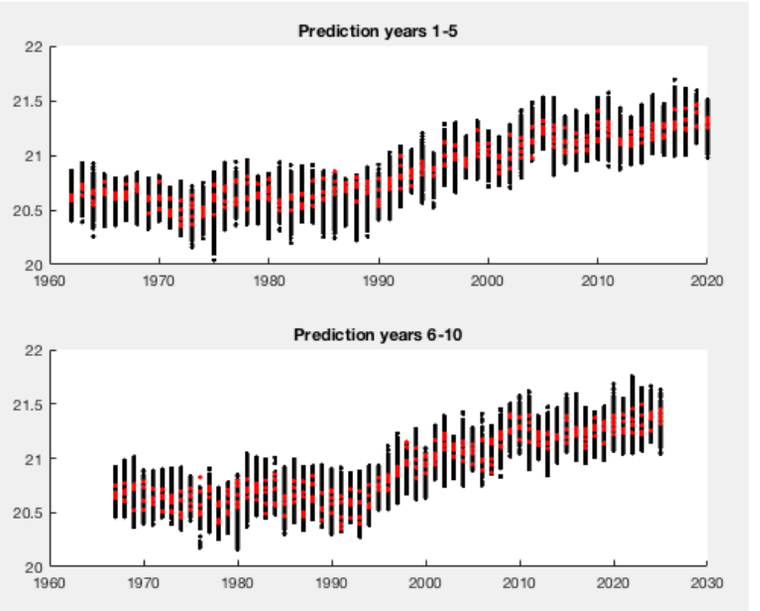


Figure 4.6: Differences for W_S_24HD

Here we can clearly see the greater amplitude of the averages of the generated data. We then proceeded to try the Standard-score normalization with cross-entropy as a loss function. The results are those in the figure 4.7.

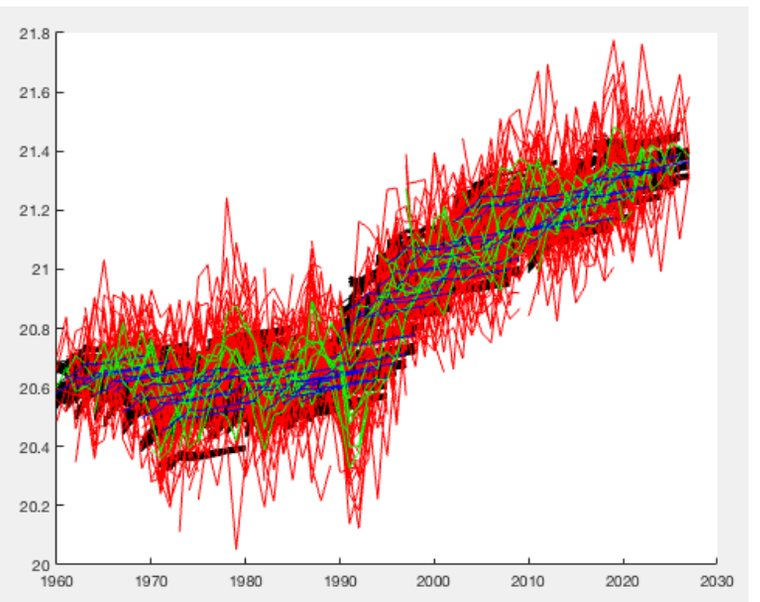


Figure 4.7: Results for CE_SsL_24HD

As you can see, the generated values have significantly smaller averages than the original values. In the representation "by differences" it is much clearer (figure 4.8.

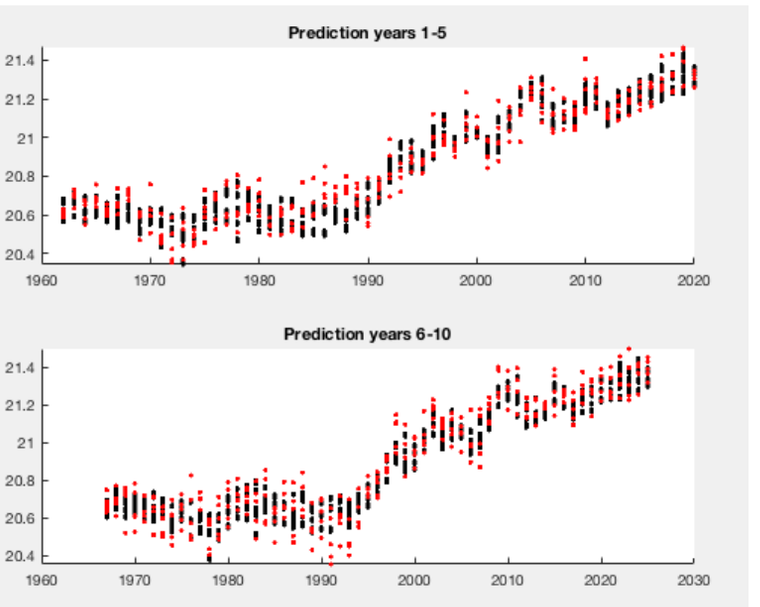


Figure 4.8: Differences for CE_S_24HD

In all likelihood, the system has gone into collappse mode. The Wassesrstein loss function was then applied, obtaining the following result (figure 4.9).

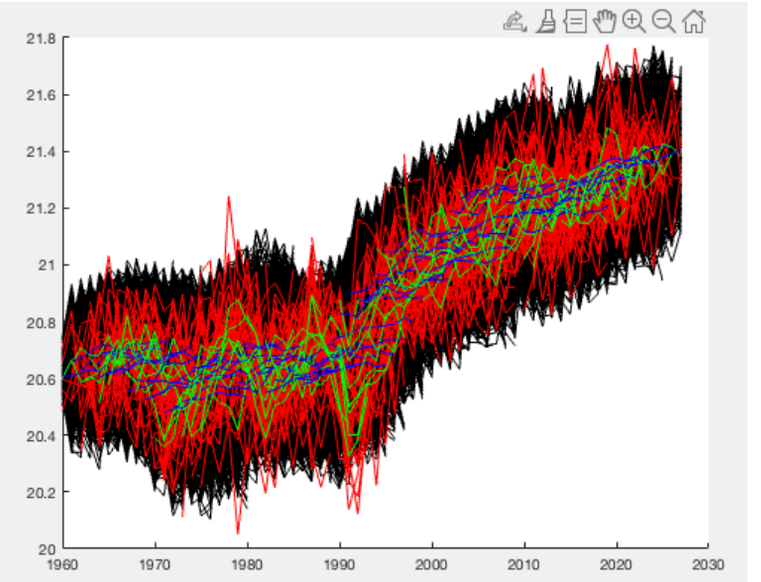


Figure 4.9: Results for W_SsL_24HD

The result is much more interesting, despite the fact that in some cases the averages of the generated values are still a little too low. To better evaluate this result, a so-called "violin plot" was made. The diagram is divided into two parts for clarity: in the first part for each year there is the distribution of the average values of the first five years of each simulation (column), on the left that of the original data (5 elements) in blue, on the right that summary data (500 elements) in red; in the second part, again for each year, there is the distribution of the average values of the second five years. The central year of each processing is indicated on the abscissa; for example in the first column of the first diagram (years 1960-1964) the year 1962 is indicated, in the first column of the second diagram (years 1965-1969) the year 1967 is indicated. (figures 4.10) and 4.11))

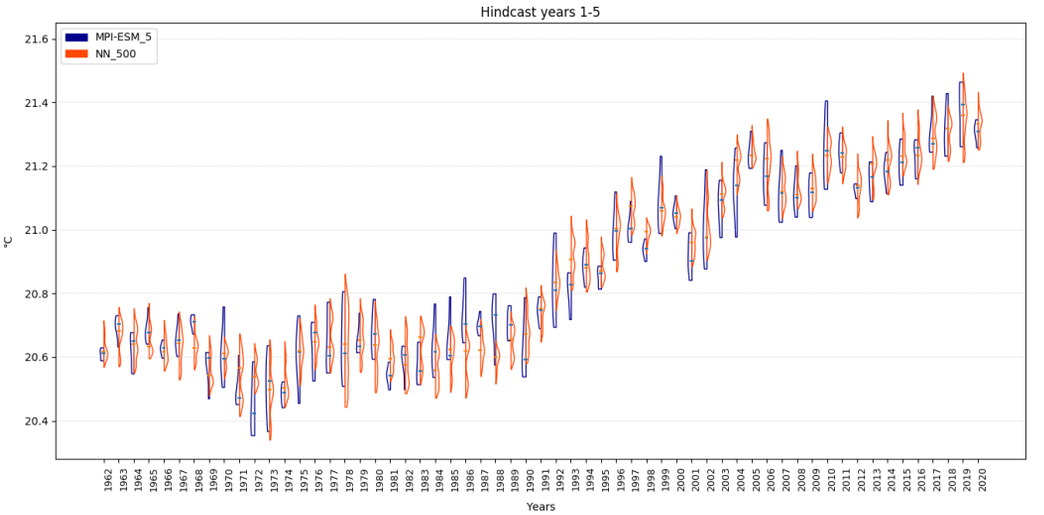


Figure 4.10: Violin plot for W_SsL_24HD part one

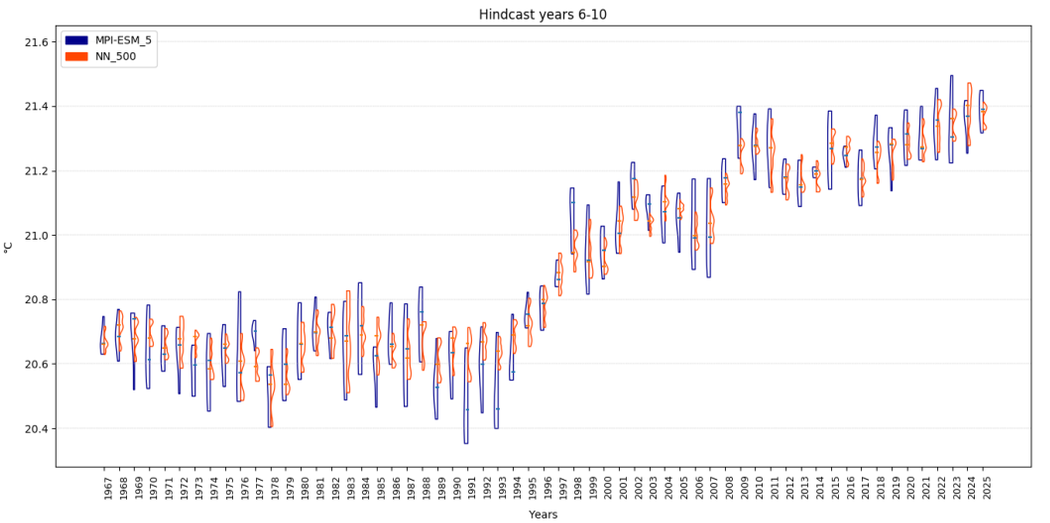


Figure 4.11: Violin plot for W_SsL_24HD part two

Finally, a further experiment was done with the localized Standard-score external normalization, the Wasserstein loss function and hidden_dimension = 50. The results are shown in the following figure 4.12):

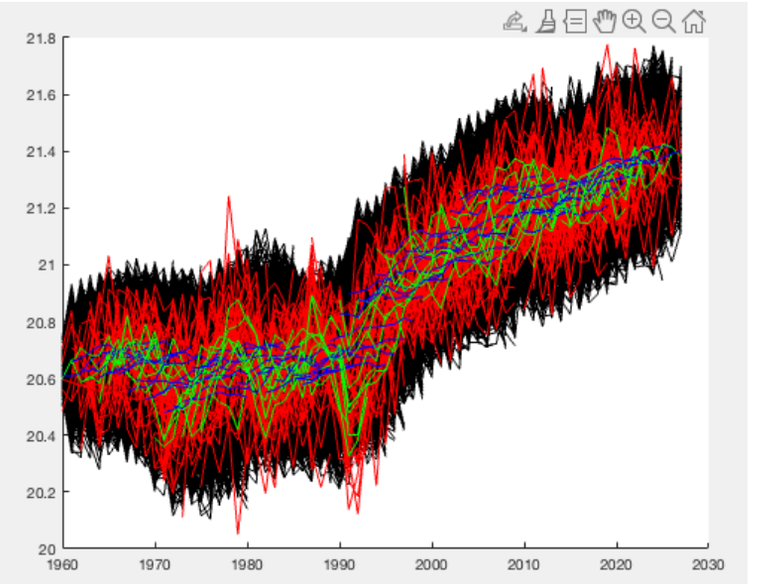


Figure 4.12: Results for W_SsL_50HD

The following two violin plots clearly illustrate the result, showing how the synthetic data conforms to the dynamics of the original data and maintain average values similar to the original ones. (figures 4.13 and 4.14)

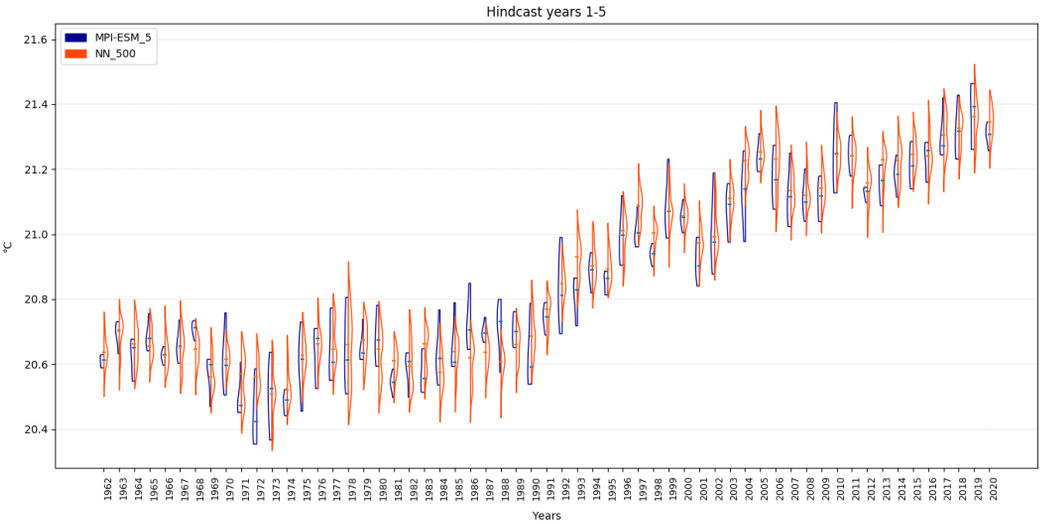


Figure 4.13: Violin-plot for W_SsL_50HD part one

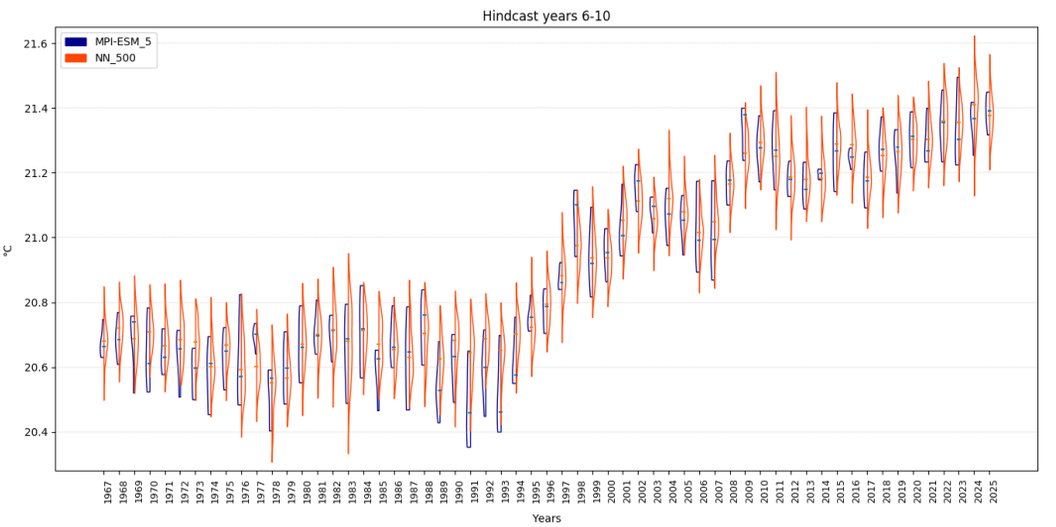


Figure 4.14: Violin-plot for W_SsL_50HD part two

Chapter 5

Conclusions and future works

5.1 Conclusions

The work aimed to quickly create synthetic data using a neural network. The TimeGAN network introduced by Jinsung Yoon in 2019 was chosen, which tries to mimic the temporal dynamics.

The network consists of four modules, embedder, recovery, generator and discriminator as described above. The cross-entropy loss function was originally adopted for each module.

It was thought useful to avoid collapse mode, and in fact it was found to be, to insert the possibility of choosing the Wasserstein loss function for the discriminator module instead of the cross-entropy loss function.

Furthermore, it was decided to normalize the training data locally, ie on each single simulation, rather than globally, ie on the whole data set.

The best realizations were precisely those that used the local Standard-score external normalization and the Wasserstein loss function.

Since the purpose of the thesis was to obtain synthetic data with the same trend as the original data, in a short time, the following table compares the computer resources used to generate the simulations between MPI-ESM and this work:

	Machine	Nodes	Cores/node	years/time
MPI-ESM	Mistral	16	36	50/day
this thesis	Mac Book pro 13'	1	2	30x5x100/hour

Table 5.1: Resource comparison

As you can see, the savings in computer resources are considerable. On a small machine it is possible to produce about 500 simulations over 30 years in just over

an hour of machine work.

5.2 Future Works

The continuation of this work requires studying the application for the generation of synthetic data starting from original multivariate data including several types of physical quantities (temperature, salinity ...).

To this we could add the geolocation in order to generate maps that illustrate the situation of the climate dynamically over time.

Another topic to be addressed is that of assimilation with regard to certain unpredictable events, such as a volcanic eruption or other that introduce even considerable variations to the values of the quantities. It should be made possible to introduce fixed points in correspondence with these phenomena whose date and influence on the parameters is known. The network should take these facts into account by forcing the curves to pass through fixed points and then take them into account in the subsequent development of the synthetic data.

These variants could require much higher computer resources than those used in this thesis, but still within the reach of a computer with an adequate GPU.

Chapter 6

Appendix

6.1 Structure of the program

The program consists of the following modules in python:

- data_loading.py: program that pre-processes the original data provided as an example by the author [9] which are a series of sine wave segments of stock market sales data (https://finance.yahoo.com/quote/ GOOG / history? P = GOOG) and energy consumption data

  (http://archive.ics.uci.edu/ml/datasets/Appliances+energy+prediction). In our case this program was not used, since the data were transformed into a format compatible with phython and subsequently normalized in various ways.

- main_timegan.py: startup program and network preparation. Initially, all model parameters were defined internally in the code. The definition of the original data file name, the hidden_dim and the num_layer have been kept internal and changed little. All the other parameters have been made available to the user who, when starting the program, must define them appropriately. The next figure shows the startup screen with some choices. In particular, note the choice of mode, which allows you to establish whether or not to save the trained model in order to be able to reload it by taking the data from the structure folder and to be able to create any number of simulations in a few seconds. Details on the tensorflow save and restore function will be seen in the timeganM.py program

```
Chose the modality (1 = normal, 2 = save model, 3 = load model): 2
number of iteration for training: 1000
number of iteration for metrics (default = 10): 10
fraction of batch size (default = 5) : 2
Type of module (gru, lstm, lstmLN): gru
file name for data generated (without suffix): prova
Folder name for structure: str_prova
Chose the type of loss (ce or w); ce
```

Figure 6.1: Home screen of main_timegan.py

- timeganM.py: is the heart of the network. The following changes have been made to the original code (timegan.py):
  - added the possibility to choose for the discriminator between the cross-entropy function loss and the Wesserstain function loss
  - added the possibility to save the trained model and then be able to retrieve and reuse it without having to retrain it. To do this, the session must be saved with the construct:

    ```
    saver.save(sess, 's_fname', global_step = 1000)
    ```

    and to restore:

    ```
    new_saver = tf.train.import_meta_graph('s_fname.meta')
    new_saver.restore(sess2, tf.train.latest_checkpoint('./'))
    ```

- utils.py: program that contains the following service modules:
  - train_test_divide: Divide train and test data for both original and synthetic data.
  - extract_time: Returns Maximum sequence length and each sequence length.
  - rnn_cell: Basic RNN Cell.
  - random_generator: random vector generator
  - batch_generator: mini-batch generator
- metrics: in the metrics folder there are three programs for the calculation of discriminatve_metrics and predictiive_metrics as well as a program for displaying the PCA_plot and the t-SNE_plot.

The following programs necessary for carrying out the tests have been added:

- programs for external standardization
  - rescaler_g.py

  - rescaler_Ss.py
  - rescaler.py
  - rescaler_Ssl.py Each of these programs takes as input an array of numpy and performs a nomalization or a renormalization. For details see section 7 Settings.

- ExportData.py: this program transforms an array of numpy into a file with the extension .mat which is needed by Matlab to produce the plots illustrating the result.

6.2 rescaler_g.py

```
def scaler(dataMat):
  dim = dataMat.shape
  nFolder = dim[0]
  nData = dim[1]
  nSim = dim[2]
  nelem = nFolder*nData*nSim
  data = np.reshape(dataMat, nelem)
  dataMatS = np.zeros(dim)
  MaMi = np.zeros(2)
  MaMi[0] = min(data)
  MaMi[1] = max(data)
  # np.save('data/mmsMat.npy', mmsMat)
  np.save('data/MaMi.npy', MaMi)
  delta = MaMi[1] - MaMi[0]
  for i in range(nFolder):
    for j in range(nData):
      for n in range(nSim):
        dataMatS[i,j,n] = (dataMat[i,j,n] - MaMi[0])/delta
return dataMatS
```

6.3 rescaler_ls.py

```
def scaler_ls(dataMat):
# format of data matrix
    dim = dataMat.shape
    nFolder = dim[0]
    nData = dim[1]
    nSim = dim[2]
    dataMatS = np.zeros(dim)
    # create the meanStd matrix: (_,0,_)=mean, (_,1,_)=std
    dimS = (nFolder, 2, nSim)
```

```
    meanStd = np.zeros(dimS)
    # popolate mmsMat with min and max of each data coloumn
    for i in range(nFolder):
      for j in range(nSim):
        meanStd[i, 0, j] = np.mean(dataMat[i, :, j])
        meanStd[i, 1, j] = np.std(dataMat[i, :, j])
    np.save('data/meanStd.npy', meanStd)
    # scale the dataMat with the function: val = (val - mean)/std
    for i in range(nFolder):
      for j in range(nSim):
        for n in range(nData):
          dataMatS[i,n,j] = (dataMat[i,n,j] - meanStd[i,0,j]) /
  meanStd[i,1,j]
    return (dataMatS)
```

Bibliography

[1] David H. Ackley, Geoffrey E. Hinton, and Terrence J. Sejnowski. A learning algorithm for boltzmann machines. *Cogn. Sci.*, 9(1):147–169, 1985. doi: 10.1207/s15516709cog0901_7. URL https://doi.org/10.1207/s15516709cog0901_7.

[2] Martín Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein generative adversarial networks. In Doina Precup and Yee Whye Teh, editors, *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*, volume 70 of *Proceedings of Machine Learning Research*, pages 214–223. PMLR, 2017. URL http://proceedings.mlr.press/v70/arjovsky17a.html.

[3] G. J. Boer, D. M. Smith, C. Cassou, F. Doblas-Reyes, G. Danabasoglu, B. Kirtman, Y. Kushnir, M. Kimoto, G. A. Meehl, R. Msadek, W. A. Mueller, K. E. Taylor, F. Zwiers, M. Rixen, Y. Ruprich-Robert, and R. Eade. The decadal climate prediction project (dcpp) contribution to cmip6. *Geoscientific Model Development*, 9(10):3751–3777, 2016. doi: 10.5194/gmd-9-3751-2016. URL https://gmd.copernicus.org/articles/9/3751/2016/.

[4] Kyunghyun Cho, Bart van Merrienboer, Çaglar Gülçehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using RNN encoder-decoder for statistical machine translation. In Alessandro Moschitti, Bo Pang, and Walter Daelemans, editors, *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing, EMNLP 2014, October 25-29, 2014, Doha, Qatar, A meeting of SIGDAT, a Special Interest Group of the ACL*, pages 1724–1734. ACL, 2014. doi: 10.3115/v1/d14-1179. URL https://doi.org/10.3115/v1/d14-1179.

[5] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

[6] Ian J. Goodfellow. NIPS 2016 tutorial: Generative adversarial networks. *CoRR*, abs/1701.00160, 2017. URL http://arxiv.org/abs/1701.00160.

[7] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. NIPS'14, page 2672–2680, Cambridge, MA, USA, 2014. MIT Press.

[8] Geoffrey E Hinton. Learning multiple layers of representation. *Trends in cognitive sciences*, 11(10):428—434, October 2007. ISSN 1364-6613. doi: 10.1016/j.tics.2007.09.004. URL https://doi.org/10.1016/j.tics.2007.09.004.

[9] J. H. Jungclaus, N. Fischer, H. Haak, K. Lohmann, J. Marotzke, D. Matei, U. Mikolajewicz, D. Notz, and J. S. von Storch. Characteristics of the ocean simulations in the max planck institute ocean model (mpiom) the ocean component of the mpi-earth system model. *Journal of Advances in Modeling Earth Systems*, 5(2):422–446, 2013. doi: https://doi.org/10.1002/jame.20023. URL https://agupubs.onlinelibrary.wiley.com/doi/abs/10.1002/jame.20023.

[10] Jochem Marotzke, Wolfgang A. MÃŒller, Freja S. E. Vamborg, Paul Becker, Ulrich Cubasch, Hendrik Feldmann, Frank Kaspar, Christoph Kottmeier, Camille Marini, Iuliia Polkova, Kerstin Prammel, Henning W. Rust, Detlef Stammer, Uwe Ulbrich, Christopher Kadow, Armin Kahl, JÃŒrgen Krager, Tim Kruschke, Joaquim G. Pinto, Holger Pohlmann, Mark Reyers, Marc Schrader, Frank Sienz, Claudia Timmreck, and Markus Ziese. Miklip: A national research project on decadal climate prediction. *Bulletin of the American Meteorological Society*, 97(12):2379 – 2394, 2016. doi: 10.1175/BAMS-D-15-00184.1. URL https://journals.ametsoc.org/view/journals/bams/97/12/bams-d-15-00184.1.xml.

[11] W. A. Müller, J. H. Jungclaus, T. Mauritsen, J. Baehr, M. Bittner, R. Budich, F. Bunzel, M. Esch, R. Ghosh, H. Haak, T. Ilyina, T. Kleine, L. Kornblueh, H. Li, K. Modali, D. Notz, H. Pohlmann, E. Roeckner, I. Stemmler, F. Tian, and J. Marotzke. A higher-resolution version of the max planck institute earth system model (mpi-esm1.2-hr). *Journal of Advances in Modeling Earth Systems*, 10(7):1383–1413, 2018. doi: https://doi.org/10.1029/2017MS001217. URL https://agupubs.onlinelibrary.wiley.com/doi/abs/10.1029/2017MS001217.

[12] Ruslan Salakhutdinov and Geoffrey Hinton. Deep boltzmann machines. In David van Dyk and Max Welling, editors, *Proceedings of the Twelth International Conference on Artificial Intelligence and Statistics*, volume 5 of *Proceedings of Machine Learning Research*, pages 448–455, Hilton Clearwater Beach Resort, Clearwater Beach, Florida USA, 16–18 Apr 2009. PMLR. URL http://proceedings.mlr.press/v5/salakhutdinov09a.html.

[13] Jinsung Yoon, Daniel Jarrett, and Mihaela van der Schaar. Time-series generative adversarial networks. In Hanna M. Wallach, Hugo Larochelle, Alina Beygelzimer, Florence d'Alché-Buc, Emily B. Fox, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, pages 5509–5519, 2019. URL https://proceedings.neurips.cc/paper/2019/hash/c9efe5f26cd17ba6216bbe2a7d26d490-Abstract.html.