Universidad Michoacana de San Nicolás de Hidalgo

División de Estudios de Posgrado de la
Facultad de Ingeniería Eléctrica

GENERATION OF SYNTHETIC DAY SCENARIOS FOR RENEWABLE ENERGY PRODUCTION USING GENERATIVE ADVERSARIAL NETWORKS

TESIS

Que para obtener el grado de

MAESTRO EN CIENCIAS EN INGENIERÍA ELÉCTRICA

Presenta

L.C.F.M. Josué Daniel González Parra

Director de Tesis

Dr. José Juan Flores Romero

Morelia, Michoacán.

Agosto 2021

GENERATION OF SYNTHETIC DAY SCENARIOS FOR RENEWABLE ENERGY PRODUCTION USING GENERATIVE ADVERSARIAL NETWORKS

Los Miembros del Jurado de Examen de Grado aprueban la **Tesis de Maestría en Ciencias en Ingeniería Eléctrica** de **Josué Daniel González Parra.**

Dr. José Antonio Camarena Ibarrola
Presidente del Jurado

Dr. Juan José Flores Romero
Director de Tesis

Dr. Claudio Rubén Fuerte Esquivel
Vocal

Dr. Félix Calderón Solorio
Vocal

Dr. Rodrigo López Farias
Revisor Externo (CONACYT Cd. México)

Dr. Roberto Tapia Sánchez
Jefe de la División de Estudios de Posgrado de la Facultad de Ingeniería Eléctrica. UMSNH
(Por reconocimiento de firmas)

UNIVERSIDAD MICHOACANA DE SAN NICOLÁS DE HIDALGO
Agosto 2021

Conferences Papers

"Evolving SARIMA Models Using cGA for Time Series Forecasting"
Juan J. Flores, Josué D. González, Baldwin Cortes, Cristina Reyes and Felix Calderón
IEEE Autumn Meeting on Power, Electronics and Computing. November, 2019.

"Data Augmentation vs Regularization for Time Series Forecasting"
Juan J. Flores, Miguel Reynoso, Josué D. González and Felix Calderón
Mexican Society for Artificial Intelligence (SMIA) and Universidad Panamericana. October, 2020.

Resumen

Las Redes Neuronales Generativas Adversariales (Generative Adversarial Networks, en inglés) son un tipo de redes neuronales de inteligencia artificial usadas dentro del aprendizaje profundo (Deep Learning) que permiten la generación de nuevos datos a partir de ejemplos.

Este tipo de redes pueden crear nuevos tipos de datos, como imágenes, videos o música, tan reales que sea imposible de distinguir si la muestra fue creada desde cero por un algoritmo o fue generada por alguna persona.

En esta tesis resolvemos si las redes generativas adversariales pueden crear series de tiempo con resolución diaria como si fueran datos reales y así poder tomar en consideración qué se espera en ese día. Cada muestra generada ayudará a contemplar las interrupciones en la energía debido las intermitencias del clima.

Usaremos datos de series de tiempo de: casi dos años de datos de irradiancia solar y temperatura ambiente, y casi un año de datos de velocidad de viento. Cada una de esas series tiene sus propias características y por ello creamos una red para cada una de ellas que sea capaz de modelar los patrones y perturbaciones diarias.

Los resultados obtenidos fueron satisfactorios porque se logró capturar la dinámica, la función de distribución, y las fluctuaciones debido al clima, de cada serie de tiempo.

Palabras Clave— Redes neuronales, modelos generativos, GAN, series de tiempo, aprendizaje profundo.

Abstract

Generative Adversarial Networks are a type of neural networks in artificial intelligence model, found in deep learning and allow generating new samples from examples. This type of neural networks can create new types of data, such as images, videos o music, so realistic that it is impossible to distinguish if the sample was entirely computer generated or was made by a person.

In this thesis we answer the question if generative adversarial networks can generate daily time series as if actually happened and taking into account what is expected on that day. Each generated sample will help to anticipate the interruption on the energy due to the fluctuations of weather conditions.

We will use three different time series: approximately two years of daily samples of solar irradiance and ambient temperature, and almost a year of daily data for wind speed. Each of these time series has its own features and thus we will design a different models for each model to be capable of replicate the daily patterns and disturbances.

The obtained results were successful because it was possible to capture the dynamics, the function probability distribution, and the fluctuations due to the weather for each time series.

Keywords— Neural networks, generative model, GAN, time series, deep learning.

Table of Contents

Conferences Papers . . . iv
Resumen . . . v
Abstract . . . vi
Table of Contents . . . vii
List of Figures . . . ix
List of Tables . . . xi
List of Algorithms . . . xii
Glossary . . . xiii

1 Introduction 1
1.1 Time series . . . 1
1.2 Artificial neural networks . . . 3
1.3 Generative adversarial networks . . . 3
1.4 Problem statement . . . 4
1.5 Objectives . . . 4
1.6 Justification . . . 5
1.7 Chapter description . . . 6

2 Deep Learning 7
2.1 Artificial neural networks . . . 7
2.2 Deep learning . . . 9
2.3 Discriminative and generative models . . . 10
2.4 Chapter conclusions . . . 12

3 Generative Adversarial Networks 13
3.1 Generative adversarial networks . . . 14
3.1.1 Loss function . . . 16
3.1.2 Training algorithm . . . 17
3.1.3 Problems with GANs . . . 18
3.2 Wasserstein GAN . . . 22
3.2.1 Wasserstein loss . . . 22
3.2.2 The Lipschitz constraint . . . 23
3.2.3 Weight clipping . . . 24
3.2.4 Gradient penalty . . . 25

3.3 Inception score . . . 27
3.3.1 Definition . . . 28
3.3.2 Calculation . . . 28
3.3.3 Implementation . . . 29
3.4 Chapter conclusions . . . 30

4 Experiments and Results 31
4.1 Time series description . . . 32
4.2 Experiment description . . . 34
4.3 Solar irradiance . . . 37
4.3.1 10-minute resolution . . . 38
4.3.2 30-minute resolution . . . 40
4.3.3 60-minute resolution . . . 40
4.4 Ambient temperature . . . 42
4.4.1 10-minute resolution . . . 43
4.4.2 30-minute resolution . . . 43
4.4.3 60-minute resolution . . . 44
4.5 Wind speed . . . 46
4.5.1 60-minute resolution . . . 46
4.6 Chapter conclusions . . . 48

5 Conclusions 49
5.1 General conclusions . . . 49
5.2 Future work . . . 51
5.2.1 Use deep convolutional GANs . . . 51
5.2.2 Forecasting and missing data . . . 52

References 53

List of Figures

1.1 Time series. 1

2.1 Perceptron. 9
2.2 Architecture for a multilayer neural network. 9
2.3 Generation of fake faces. 12
2.4 Images taken from the site ThisPersonDoesNotExist.com. 12

3.1 Structure of a generative adversarial network. 15
3.2 Real and generated distributions. 18
3.3 Convergence failure. 20
3.4 Mode collapse. 20
3.5 Sigmoid function and its derivative. 21
3.6 A Lipschitz continuous function. 24
3.7 Interpolation between real and generated samples. 26

4.1 Real vs generated images. 31
4.2 Time series. 33
4.3 Daily time series. 34
4.4 Creating the training sets. 35
4.5 Wasserstein generator. 36
4.6 Wasserstein critic. 36
4.7 Scenarios for solar irradiance using area under the curve. 37
4.8 Generated samples for solar irradiance 10-minute resolution. 38
4.9 Generated samples for solar irradiance 30-minute resolution. 40
4.10 Generated samples for solar irradiance 60-minute resolution. 41
4.11 Ambient temperature. 42
4.12 Generated samples for ambient temperature 10-minute resolution. 43
4.13 Generated samples for ambient temperature 30-minute resolution. 44
4.14 Generated samples for ambient temperature 60-minute resolution. 45
4.15 Scenarios for wind speed using area under the curve. 47
4.16 Generated samples for wind speed 60-minute resolution. 47

5.1 Scenario comparison. 50
5.2 Encoding map of Gramian Angular Fields. 51

5.3 Missing values and forecasting. 52

List of Tables

4.1 Results for 10-minute resolution solar irradiance. 39
4.2 Results for 30-minute resolution solar irradiance. 41
4.3 Results for 60-minute resolution solar irradiance. 42
4.4 Results for 10-minute resolution ambient temperature. 44
4.5 Results for 30-minute resolution ambient temperature. 45
4.6 Results for 60-minute resolution ambient temperature. 46
4.7 Confusion matrix for 60-minute resolution wind speed. 48
4.8 Results for the Inception Score. 48

List of Algorithms

3.1 Train_GAN(Generator, Discriminator, n_epochs, batch_size) 19
3.2 Gradient_Penalty(Discriminator, real_samples, generated_samples) 26
3.3 WGAN_GP(Generator, Discriminator, n_epochs, batch_size, n_critic, λ) . . . 27
3.4 Inception_Score(p_yx, n_split) 30

Glossary

ANN Artificial neural network. 3, 7, 8

AUC Area under the curve. 33, 37

BCE Binary cross-entropy. 23

CIFAR Canadian Institute For Advanced Research. 31

DCGAN Deep convolutional generative adversarial network. 51

DL Deep learning. 6, 9

GAN Generative adversarial network. 3, 4, 6, 11, 13, 14, 18, 22, 30–32

IS Inception score. 27–29, 48

KL Kullback-Leibler. 29

ML Machine learning. 7, 10

MLP Multilayer perceptron. 3, 8

MNIST Modified National Institute of Standards and Technology. 31

ReLU Rectified Linear Unit. 35, 36

TS Time series. ix, 1, 32–38, 40–47

WGAN Wasserstein generative adversarial network. 22, 36, 38, 48, 49

Chapter 1

Introduction

1.1 Time series

A time series is a time-ordered sequence of observed values of a variable made at equally spaced time intervals, represented as a vector of values $[x_1, x_2, x_3, \cdots, x_N]$ [Palit and Popovic, 2006]. Time series are found in many fields, such as economics, sociology, meteorology, medicine, seismology, oceanography, geomorphology, astronomy, etc. [Granger et al., 2014]. Time series analysis helps to detect regularities in the observations of a variable, determines a suitable model, or exploit all information included in this variable to better predict future developments, [Kirchgassner and Wolters, 2008].

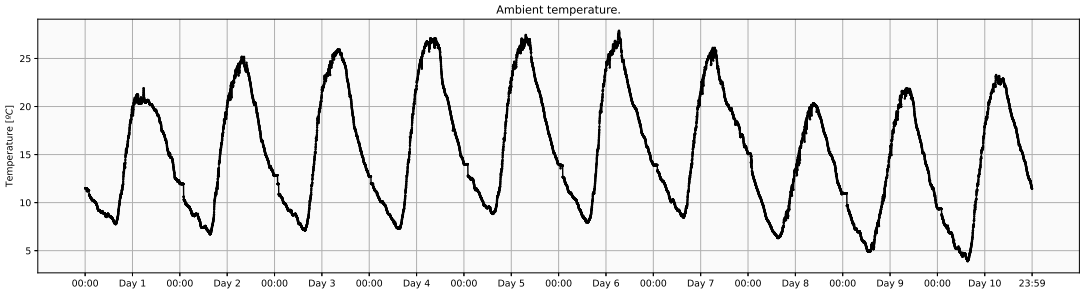


Figure 1.1: Dataset from ambient temperature.

Time series forecasting is currently a very important research area, due to the importance of prediction and understand information about the phenomena in many fields. Applications range from natural phenomena such as wind speed, ambient temperature,

solar irradiance, etc., to anthropic phenomena like stock price, electric energy consumption, etc. Time series forecasting is an area in which past observations of the same variable are collected and analyzed to develop a model the describes the underlying relationship. The model is then used to extrapolate the time series into future scenarios [Zhang, 2003].

Over the past several decades, many studies have been conducted to develop innovative forecasting approaches and improve their accuracy. In general, these models can be categorized into three types: statistical models, artificial intelligence models and hybrid models [Xu et al., 2019].

Statisticians and econometricians tend to rely on autoregressive integrated moving average (ARIMA) and derived or related models, while the artificial intelligence community mainly looks at neural networks, either using multilayer perceptrons or recurrent networks [Lemke and Gabrys, 2010].

After fitting a time series model, one can evaluate it with forecast fit measures. The researcher may subtract the forecast value from the observed value of the data at that time point and obtain a measure of error. The statistics used to describe this error are similar to the univariate statistics just mentioned, except that the forecast is often substituted for the average value of the series.

Time series analysis normally mix several methods to work with time series data, in order to extract potentially useful information oriented to pursue two main goals:

Determination of the time series behavior. Identification of the important parameters and characteristics, which adequately describe the time series behavior, and

Time seres forecasting. Forecasting the future values of the time series, depending on its actual and past values. Even

Both of these goals require the time series model identification. As soon as the model is identified, it can be exploited to interpret the time series behavior. The model can also be used to forecast its future values.

1.2 Artificial neural networks

According to [Khashei et al., 2008] and [de Oliveira et al., 2000], artificial neural networks approach has been suggested as an alternative technique to time series forecasting, and this approach has gained immense popularity in the last few years. Artificial neural networks try to recognize regularities and patterns in the input data, learn from experience and then provide generalized results based on their previous knowledge.

The main artificial neural network used is in forecasting problems are the **Multi-layer perceptrons (MLPs)**. These models are characterized by a network of three types of layers (input, hidden and output), they consist of neurons arranged in layers in which every neuron is connected to all neurons of the next layer [Shirvany et al., 2009]. This feed forward neural network model in fact performs a non-linear functional mapping from the past observations of the time series to the future value.

Two of the main types of neural networks used to solve the forecasting task are:

- Recurrent neural networks [Géron, 2017], and
- Time lagged neural networks [Adhikari and Agrawal, 2013].

This thesis proposes the use of the generative adversarial networks created by [Goodfellow, 2017].

1.3 Generative adversarial networks

Generative adversarial networks are algorithmic architectures that use two neural networks, competing one against the other in order to generate new synthetic instances of data that can pass for real. They are widely used for voice, image, or video generation [Langr and Bok, 2019].

The two different neural networks used in a generative adversarial network, are **generator** G and **discriminator** D. The first one is responsible for the generation of synthetic data, and the discriminator function is to judge the quality of the generated data and provide feedback to the generator [Takahashi et al., 2019].

The discriminator learns using traditional supervised learning techniques, dividing inputs into real or fake classes; meanwhile the generator must learn to create samples that are drawn form the same distribution as the training data, i.e., the generator captures the probability distribution of the underlying process that produce the original data.

1.4 Problem statement

This thesis proposes the use of generative models, specifically the use of **generative adversarial networks** [Goodfellow et al., 2014] to generate plausible scenarios.

A fundamentally different approach to forecasting is *scenario-based forecasting*. The aim of this approach is to generate forecasts based on plausible scenarios. In contrast to other forecasting techniques where the resulting forecast is intended to be a likely outcome, each scenario-based forecast may have a low probability of occurrence [Hyndman and Athanasopoulos, 2018].

Building forecast based on scenarios allows a wide range of possible forecasts to be generated, for example "best" or "worst" case scenarios are presented and allow the decision makers to understand the results.

Given a data set of time series with similar characteristics (as high ambient temperature or low solar irradiance), the generative adversarial networks are able to create a scenario with those same characteristics. This way we can model the patterns of the time series.

1.5 Objectives

The main objetive from this thesis is to use generative adversarial networks to create a model capable of make plausible scenarios with minimum error based on previous data. Generated scenarios should be capable of describing the same stochastic processes as training samples and exhibiting a variety of different modes representing all possibles variations and patterns seen during training.

In particular, this thesis aims to solve the following problems:

- Design and develop algorithms capable of training the generative adversarial neural networks from the real data.

- Generate a competitive method to forecast, and

- Compare it against the state of the art methods for time seres forecasting.

1.6 Justification

Renewable energies, such as wind power, solar energy or hydraulic energy, are sources of clean, inexhaustible and increasingly competitive energy. They differ from fossil fuels principally in their diversity, abundance and potential for use anywhere on the planet, but above all in that they produce neither greenhouse gases – which cause climate change – nor polluting emissions. Their costs are also falling at a sustainable rate, whereas the general cost trend for fossil fuels is in the opposite direction in spite of their present volatility.

However, if more renewable energies are introduced, the power grid system might be destabilized due to the fluctuations of weather conditions. To keep the power grid system stable even when we introduce more renewable energy to the grid, we need to predict the output of renewable energy and compensate the fluctuations by thermal power plants, hydroelectric power plants, and/or batteries [Bigdeli, 2016].

With the growing penetration of renewable energy sources in power systems, it becomes increasingly important to characterize their inherent variability and uncertainty [Qiao et al., 2021]. One way to try to mitigate energy fluctuations due to weather, is to create a time series scenario in order to model when will be an energy interruption, and so can take an informed decision [Conejo, 2010].

Generative adversarial networks are able to imitate pictures or human voice, and it is possible that GANs can also imitate the time series behavior and make an extrapolation on the data. Therefore we believe we can use them to generate a realistic scenario for time series.

1.7 Chapter description

This thesis is composed of five chapters.

Chapter 2 Deep Learning. Describes the theoretical foundations of machine learning and neural networks. In this chapter we'll build on these foundations to give the core concepts of deep networks.

Chapter 3 Generative Adversarial Networks. Introduce what are the Generative Adversarial Networks (GANs) and provide a high-level explanation of how they work. GANs consist of two separate neural networks (the Generator and the Discriminator), and the networks are trained through a competitive dynamic.

Chapter 4 Experiments and Results. Presents the scenarios generated with our scheme, along with the comparison with the real data.

Chapter 5 Conclusions. Presents a summary of the results and recommendations for future work regarding the subject.

Chapter 2

Deep Learning

This chapter builds the foundations of machine learning and neural networks, and use them to give the core concepts of deep learning. This is useful when describing *generative modeling* and how to use it to generate time series scenarios.

2.1 Artificial neural networks

One family of Machine Learning (ML) models are **Artificial Neural Networks** (ANNs); they have been around for at least 50 years. Many important architectural advances were made in the mid-1980s and early 1990s; however, the interest in this type of model lost interest because the data and time needed to get good results were too big. Deep learning emerged in earlys 2000s along with the expansion of computational power and development of new optimization algorithms able to deal with big data. Deep learning models won many important machine learning competitions [Patterson and Gibson, 2017].

While inspired by the human brain and how its neurons interact with each other, ANNs are not meant to be realistic models of the brain. Instead, they are inspired by it, allowing us to draw analogies between a very basic model of the brain and how we can mimic some of its behavior through artificial neural networks.

A biological neuron is an excitable unit that can process and transmit information via electrical and chemical signals; a biological *neural network* is composed of, approximately, 86 billion neurons connected to many other neurons.

There are two main properties of artificial neural networks that follow the general idea of how the brain works. First is that the most basic unit of the neural network is the artificial neuron (or node). Artificial neurons are modeled on the biological neurons of the brain, and like biological neurons, they are stimulated by inputs. These artificial neurons pass on some, but not all, information they receive to other artificial neurons, often with transformations.

Second, much as the neurons in the brain can be trained to pass forward only signals that are useful in achieving the larger goals of the brain, we can train the neurons of a neural network to pass along only useful signals.

One of the first neural network model came from [McCulloch and Pitts, 1943]. This network was a binary classifier, capable of recognizing two different categories based on some input. The problem was that the weights used to determine the class label for a given input needed to be manually tuned by a human.

Then, in the 1950s the seminal Perceptron algorithm was published by Rosenblatt [Rosenblatt, 1958], this model could automatically learn the weights required to classify an input (no human intervention required). An example of the Perceptron architecture can be seen in Figure 2.1, we can see it forms and acyclic and directed graph. Using perceptron architecture with the automatic training procedure formed the basis of Stochastic Gradient Descent (SGD) which is still used to train very deep neural networks today [Rosebrock, 2017]

The most well-known, and simplest to understand neural network, is the feed-forward multilayer perceptron (MLP). It has an input layer, one or many hidden layers, and a single output layer. Each layer can have a different number of neurons and each layer is fully connected to the adjacent layer. The connections between the neurons in the layers form an acyclic graph, see Figure 2.2.

Feed-forward multilayer neural networks are *universal approximators*, if given enough nodes they are capable of approximating any function [Csáji, 2001]. It is generally trained by a learning algorithm called *backpropagation learning*. This algorithm uses gradient descent on the weights of the connections in a neural network to minimize the error on the output of the network. Although, backprogation can stuck in local minima, in

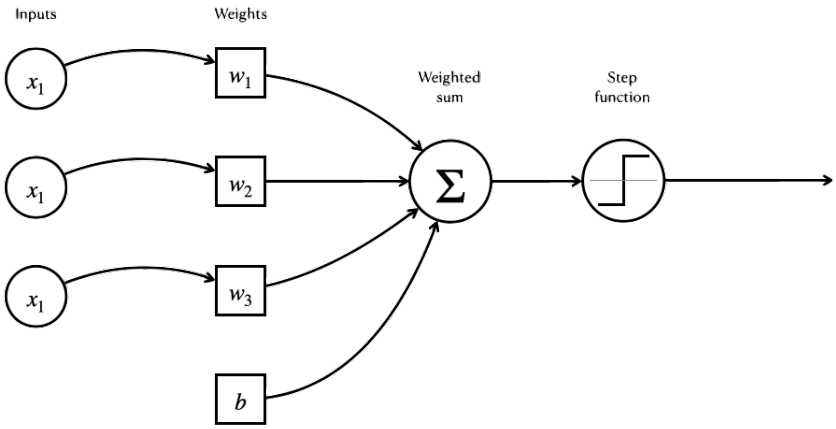


Figure 2.1: An example of the simple Perceptron network architecture that accepts a number of inputs, computes a weighted sum, and applies a step function to obtain the final outcome.

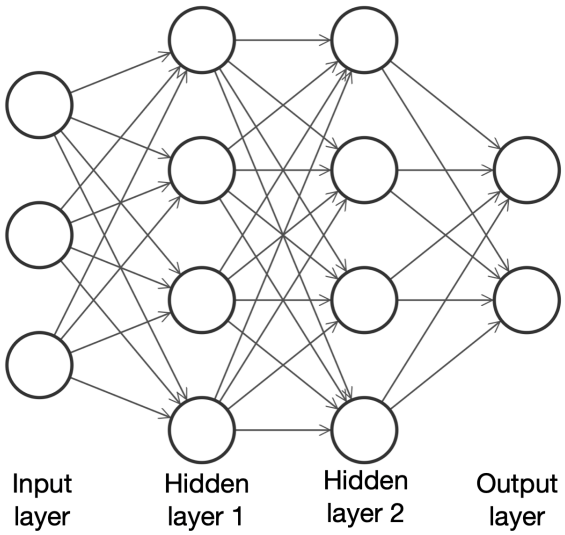


Figure 2.2: A multi-layer, feedforward network architecture with an input layer (3 nodes), two hidden layers with 3 nodes each, and an output layer with 2 nodes.

practice it generally performs well [Patterson and Gibson, 2017].

2.2 Deep learning

It is difficult to define what *deep learning*s is, because it has changed forms over the past years. One of the first attempts to define it was *"a neural network with more than*

two layers", another attempts to define it are:

- "Deep learning is a particular kind of machine learning that achieves great power and flexibility by learning to represent the world as a nested hierarchy of concepts, with each concept defined in relation to simpler concepts, and more abstract representations computed in terms of less abstract ones" [Goodfellow et al., 2016].
- "A neural network with a large number of parameters and layers in one of four fundamental network architectures: unsupervised pretrained networks, convolutional neural networks, recurrent neural networks, or recursive neural networks" [Patterson and Gibson, 2017].
- "Deep learning is a class of machine learning algorithm that uses multiple stacked layers of processing units to learn high-level representations from unstructured data" [Foster, 2019].

A deep artificial neural network consists of a series of stacked *layers*. Each layer contains *units*, that are connected to the previous layer's units through a set of *weights*. Deep artificial neural networks can have any number of hidden layers. For example, ResNet [He et al., 2015] designed for image recognition, contains 152 layers.

2.3 Discriminative and generative models

Machine learning algorithms, such as neural networks, are great at recognizing patterns in existing data and using that insight for tasks such as *classification* and *regression*, this is called discriminative modeling.

In discriminative modeling, we are interested in developing a model to predict a class label given an example of inputs variables, i.e., use the training data to find a discriminant function $f(x)$ that maps each sample x onto a class label [Bishop, 2006].

Discriminative models gave the correct category to a sample. The model learns the conditional probability of the target variable given the input variable $P(Y|X = x)$, e. g., logistic regression or linear regression. These models focus on modeling the boundary between classes.

A generative model describes how a dataset is generated, in terms of a probabilistic model. By sampling from this model, we are able to generate new data. A generative model must also be *probabilistic* rather than *deterministic* [Ahirwar, 2019]. This type of model must include some random element that influence the individual samples generated by the model.

Generative models understand how the data was created in order to generate new data. The model learns the joint probability distribution of the input variable and the output variable $P(X, Y) = P(X|Y)P(Y)$; if the model wants to predict something, it uses the Bayes theorem and computes the conditional probability of the target variable given the input variable $P(Y|X) = \dfrac{P(X, Y)}{P(X)}$.

The distribution $P(Y|X = x)$ for the discriminative models is the natural distribution for taking input x and producing an output Y (classification). Using generative models to learn the distribution $P(X, Y)$, we can generate likely output given a certain input.

The advance on generative models over discriminative ones is that we can use them to create new instances of data, because the model learns the distribution function of the data itself, which is not possible using a discriminator.

In 2014 Ian Goodfellow and colleagues [Goodfellow et al., 2014] at the University of Montreal, first published the Generative Adversarial Networks, or GANs. This technique has enable computers to generate realistic data by using two separated neural networks [Langr and Bok, 2019].

Figure 2.3 shows how far the machine data generation has advanced thanks to GANs on the synthesis of human faces. Before the GANs were invented, back to 2014, the best that machines could produce were blurred images, and even that was celebrated as groundbreaking success. After three years, by 2017, advances in GANs enable computers to generate fake faces whose quality rivals high-resolution portrait photographs.

The website ThisPersonDoesNotExist.com, created by Philip Wang, a software engineer at Uber, uses the chip designer Nvidia to create an endless stream of fake portraits (Figure 2.4). The algorithm behind it is trained on a huge dataset of real images, then uses a GAN to fabricate new examples.

Figure 2.3: Progress in human face generation made by GANs. Image taken from "The Malicious use of Artificial Intelligence: Forecasting, Prevention and Mitigation" by Miles Brundage *et al.* [Brundage et al., 2018].

Figure 2.4: Images taken from the site ThisPersonDoesNotExist.com.

2.4 Chapter conclusions

This chapter, introduced the canonical feed-forward artificial neural networks. Inspired by networks of biological neurons, feed-forward networks are the simplest artificial neural networks. They are composed by an input layer, one or many hidden layers, and an output layer. We also introduced the field of generative modeling, an important branch of machine learning that complements the more widely studied discriminative modeling.

Chapter 3

Generative Adversarial Networks

Ian Goodfellow of Google Brain presented a tutorial entitled "Generative Adversarial Networks" to the delegates of the Neural Information Processing Systems (NIPS) conference in Barcelona [Goodfellow, 2017]. The ideas presented in the tutorial are now regarded as one of the key turning points for generative modeling and have spawned a wide variety of variations on his core idea that have pushed the field to even greater heights.

Generative adversarial networks present a way of training a generative model by framing the problem as a supervised learning problem with two networks: the generator model that we train to generate new examples, and the discriminator model that tries to classify examples as either real (from the domain) or generated. The two models are trained together in a zero-sum game, adversarial, until the discriminator model is fooled about half the time, meaning the generator model is generating realistic examples.

Some interesting applications of GANs to help develop an intuition for the types of problems where GANs can be used and useful are:

Image generation. Generative networks can be used to generate realistic images after being trained on sample images, as shown at [Goodfellow et al., 2014, Radford et al., 2016, Karras et al., 2018, Brock et al., 2019].

Text-to-image synthesis. Generating images from text descriptions is an interesting use case of GANs [Zhang et al., 2017a, Reed et al., 2016b, Reed et al., 2016a].

Face aging. An age-cGAN network can generate images at different ages, which can then be used to train a robust model for face verification [Antipov et al., 2017, Zhang et al., 2017b].

Anomaly Detection. MIT researchers proposed an unsupervised anomaly detection approach named *TadGAN*, that allows time series reconstruction and effectively flag anomalies in the data [Geiger et al., 2020, Luer et al., 2019].

3.1 Generative adversarial networks

Generative adversarial networks (GANs) are an exciting and rapidly changing field, delivering on the promise of generative models in their ability to generate realistic examples across a range of problem domains [Brownlee, 2019] and store knowledge within it as the learn the pattern of the true data distribution and try to generate new samples that look like the samples from this true data distribution.

In generative adversarial networks (GANs), we use a generator, G, to produce new samples, and we use a discriminator, D, to tell if a given sample is an original training sample or it has been produced by the generator. Both models, G and D, work in an "adversarial" setup, i.e., they compete with each other and eventually both of them are improving in their tasks.

The term *adversarial* points to the game-like, competitive dynamics between the two models that constitute the GAN framework. The generator's goal is to create samples that are indistinguishable from the real data in the training set. The discriminator's objective is to distinguish the fake examples produced by the generator from the real examples coming from the training dataset. The two networks are continually trying to outwit each other: the better the generator gets at creating convincing data, the better the discriminator needs to be at distinguishing real examples from the fake ones.

A GAN consists of a generator (G) and a discriminator (D), both are feed-forward neural networks, since the adversarial modeling framework is most straightforward to apply with weights θ_G and θ_D respectively [Goodfellow et al., 2014]. The bottom left corner of

Figure 3.1 shows the theoretical[1] distribution function of the original data; on the top left corner, is the normal distribution (although we can choose any other distribution) from where we will sample randomly data, z, to feed the generator and obtain $G(z)$ which will have the same domain of the original data. Finally, both, the generated data and the original data, will be the input to the discriminator and this will provide the probability of belonging to the original data.

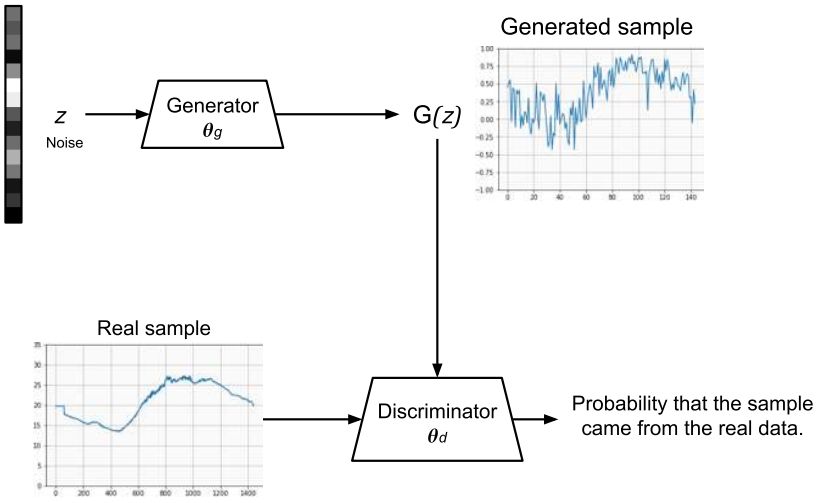


Figure 3.1: Structure of a generative adversarial network.

The discriminator tries to maximize the chances of detecting fake data, but the generator tries to fool the discriminator, i.e., both the generator and the discriminator are playing a two-player minimax game. This kind of framework is optimized by the following:

- The generator, G, is optimized to generate data that deceives the discriminator D, and
- The discriminator, D, is optimized to distinguish the source of the input, namely the generator G or realistic dataset.

In more technical terms, the generator's goal is to produce examples that capture the characteristics of the training dataset, so much so that the samples it generates look

[1]The real distribution is unknown. All we know is a finite set of samples drawn from the theoretical distribution.

indistinguishable from the training data. The generator can be thought of as an object recognition model in reverse. *Object recognition algorithms* learn the patterns in images to discern an image's content [Langr and Bok, 2019]. Instead of recognizing the patterns, the generator learns to create them essentially from scratch; indeed, the input into the generator is often no more than a vector of random numbers.

The generator learns through the feedback it receives from the discriminator's classifications. The discriminator's goal is to determine whether a particular example is real (coming from the training dataset) or generated (created by the generator). Accordingly, each time the discriminator is fooled into classifying generated image as real, the generator knows it did something well. Conversely, each time the discriminator correctly rejects a generator-produced image as generated, the generator receives the feedback that it needs to improve.

The discriminator continues to improve as well. Like any classifier, it learns from how far its predictions are from the true labels (real or generated). So, as the generator gets better at producing realistic-looking data, the discriminator gets better at telling fake data from the real, and both networks continue to improve simultaneously. The next section will construct the value function to be minimize and maximize.

3.1.1 Loss function

The adversarial game previously described, can be formalized by minimax of a target function between the discriminator function $D : \mathbb{R}^n \to [0, 1]$ and the generator function $G : \mathbb{R}^d \to \mathbb{R}^n$. The generator G turns random samples $z \in \mathbb{R}^d$ from the normal distribution $N(0, 1)$ into generated samples $G(z)$. The discriminator D tries to tell them apart from the training samples coming from the (empirical) real distribution γ, while G tries to make the generated samples as similar in distribution to the training samples. The target loss function proposed by [Goodfellow et al., 2014] is:

$$V(D, G) = \mathbb{E}_{x\sim\gamma} \ln [D(x)] + \mathbb{E}_{z\sim N(0,1)} \ln [1 - D(G(z))] \tag{3.1}$$

where $\mathbb{E}$ denotes the expectation with respect to a distribution specified in the subscript.

Generative adversarial networks solve the minimax problem

$$\min_{\theta_G} \max_{\theta_D} V(D,G) = \min_{\theta_G} \max_{\theta_D} \left(\mathbb{E}_{x\sim\gamma} \ln \left[D(x) \right] + \mathbb{E}_{z\sim N(0,1)} \ln \left[1 - D(G(z)) \right] \right) \tag{3.2}$$

For a given generator G, $\max_{\theta_D} V(D,G)$ optimizes the discriminator D to reject generated samples $G(z)$ by attempting to assign high values to samples from the distribution γ and low values to generated samples $G(z)$. Conversely, for a given discriminator D, $\min_{\theta_G} V(D,G)$ optimize G so that the generated samples $G(z)$ will attempt to confuse the discriminator D into assigning high values [Wang, 2020].

The first term in equation (3.2) is the expectation of the discriminator output when the input came from the real data distribution; and the second term is the discriminative predictions when the inputs came form the false data.

At the end of the minimax game, the generator and discriminator interaction translate to a more general objective for the whole GAN architecture. That is, to make the real and generated data distributions very similar, i.e., trying to get the generated distribution as close as possible to the real distribution. During this whole training process, the discriminator is trying to delineate the real and fake distribution as much as possible, whereas the generator is trying to make the generated distribution look more like the real samples, see Figure 3.2.

3.1.2 Training algorithm

The GAN training algorithm involves training both the discriminator and the generator model in parallel. Algorithm 3.1 shows how GAN training works.

First, a batch of random points from the latent space must be selected for use as input to the generator model to provide the basis for the generated samples (lines 2 and 3 from Algorithm 3.1).

Then a batch of samples from the training dataset must be selected for input to the discriminator as the original samples, line 4.

Using both original and generated samples, we associate labels 1 and 0 to each original and fake sample respectively. The samples are passed into the discriminator to get

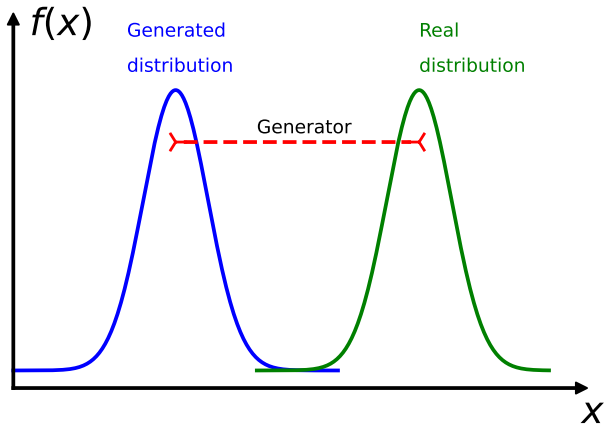


Figure 3.2: The, theoretical, real distribution (green) and the normal distribution (blue). The generator tries to minimize the distances between the two distribution (red line) while the discriminator tries to distinguish between samples.

predictions and use the labels and cost function to calculate the loss (line 5). The gradient of the cost function with respect of the discriminator's parameters tell us how to change each parameter to most efficiently increase the loss function by Equation (3.3) (line 6).

$$\nabla\theta_d \frac{1}{m} \sum_{i=1}^{m} \Big(\ln\left[D\left(x_i\right)\right] + \ln\left[1 - D\left(G(z_i)\right)\right] \Big) \tag{3.3}$$

In order to update the generator model, we sample another batch of random noise and transform them with the generator to get the generated samples (lines 7 and 8). The objective is only to update the generator's parameters by Equation (3.4) (line 10); this is done by taking the gradient of the loss function with respect to these parameters (line 9).

$$\nabla\theta_g \frac{1}{m} \sum_{i=1}^{m} \ln\left[1 - D(G(z_i))\right] \tag{3.4}$$

3.1.3 Problems with GANs

While GANs are a major breakthrough for generative modeling, they are also notoriously difficult to train. GANs are difficult to train because both the generator and the discriminator are trained simultaneously in a minimax game. This means that improvements to one model come at the expense of the other model. The goal of training two

Algorithm 3.1: Train_GAN(Generator, Discriminator, n_epochs, batch_size)

```
for i = 1 to n_epochs do
    // Update the discriminator
    z = Get_Noise_Vectors(size=batch_size)
    generated_samples = Generator(z)
    original_samples = Get_Real_Samples(size=batch_size)
    Loss_g, Loss_d = Discriminator([real_samples, generated_samples])
    Discriminator.UpdateWeights([Loss_g, Loss_d])
    // Update the generator
    z = Get_Noise_Vectors(size=batch_size)
    generated_samples = Generator(z)
    Loss_g = Generator(generated_samples)
    Generator.UpdateWeights(Loss_g)
end
```

models involves finding an equilibrium point between the two competing concerns. It also means that every time the parameters of one of the models are updated, the nature of the optimization problem that is being solved is changed.

Convergence Failure. Typically, a neural network fails to converge when the model loss does not settle down during the training process. In the case of a GAN, a failure to converge refers to not finding an equilibrium between the discriminator and the generator, see Figure 3.3. The likely way that you will identify this type of failure is that the loss for the discriminator has gone to zero or close to zero. In some cases, the loss of the generator may also rise and continue to rise over the same period.

Mode collapse. A mode collapse refers to a generator model that is only capable of generating one or a small subset of different outcomes, or modes. Here, mode refers to an output distribution, e.g. a multi-modal function refers to a probability distribution with more than one local maximum probability peak. With a GAN generator model,

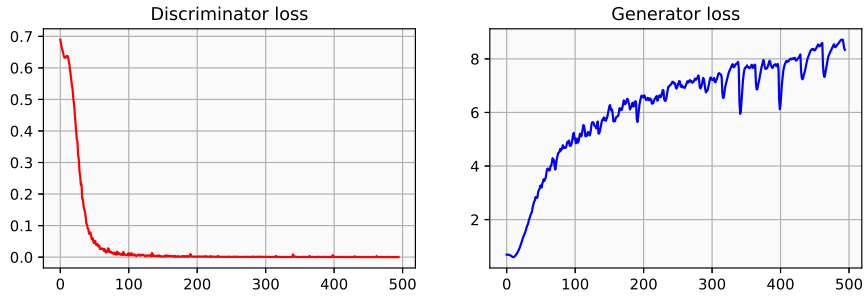


Figure 3.3: Convergence failure refers to not finding an equilibrium between the discriminator and the generator.

a mode failure means that the vast number of points in the input latent space result in one or a small subset of generated samples.

Columns from Figure 3.4 show a heatmap of the generator distribution after increasing numbers of training steps. The final column shows the data distribution (a toy 2D mixture of Gaussians dataset). The top row shows training for a GAN without mode collapse, its generator quickly spreads out and converges to the target distribution. The bottom row shows a GAN training with mode collapse, the generator rotates through the modes of the data distribution. It never converges to a fixed distribution.

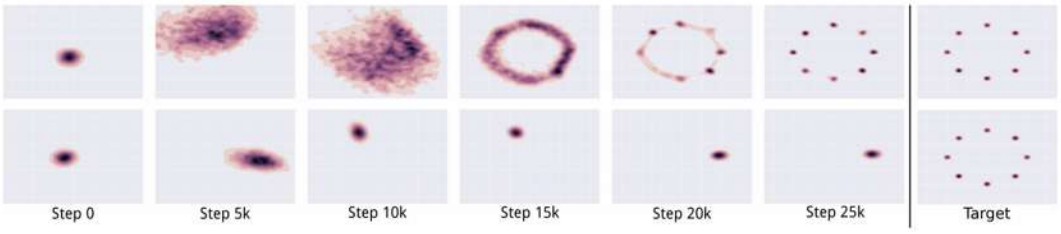


Figure 3.4: Taken from **Unrolled Generative Adversarial Networks** by Luke Metz, *et. al.* 2017.

Vanishing gradients. If the discriminator is too good, then generator training can fail due to vanishing gradients During backpropagation, the gradient flows backward, from the final layer to the first layer. As it flows backward, it gets increasingly smaller. Sometimes, the gradient is so small that the initial layers learn very slowly or stop learning completely. In this case, the gradient does not change the weight values of the initial layers at all, so the training of the initial layers in the network is effectively stopped.

Certain activation functions, like sigmoid function Figure 3.5, encloses a large input space into a small input space between 0 and 1.Therefore, a large change in the input of the sigmoid will cause a small change in the output. Hence, the derivative becomes small.

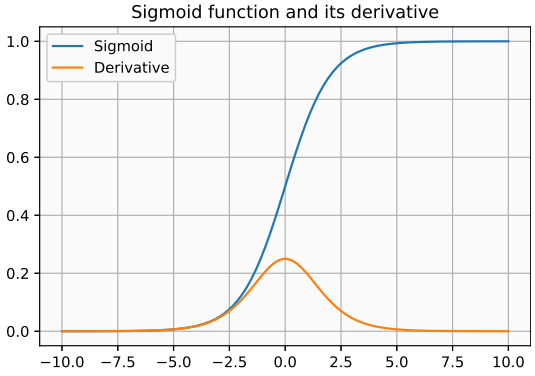


Figure 3.5: The gradient could be very small and does not change the weights of the layers.

3.2 Wasserstein GAN

The Wasserstein GAN (WGAN) was one of the first big steps toward stabilizing GAN training. With a few changes, Arjovsky et al., were able to show how to train GANs that have the following two properties:

- A meaningful loss metric that correlates with the generator's convergence and sample quality.
- Improved stability of the optimization process.

Specifically, the paper introduces a new loss function for both the discriminator and the generator. Using this loss function instead of binary cross entropy results in a more stable convergence of the GAN.

3.2.1 Wasserstein loss

To train the GAN discriminator, we calculate the loss when comparing class estimation for real samples $p_i = D(x_i)$ to the response $y_i = 1$, and estimations for generated samples $p_i = D(G(z_i))$ to the response $y_i = 0$. Therefore for the GAN discriminator, minimizing the loss function (Equation 3.1) can be written as:

$$\max_{\theta_D} \left(\mathbb{E}_{x \sim \gamma} \left[\ln D(x) \right] + \mathbb{E}_{z \sim N(0,1)} \ln \left[1 - D(G(z)) \right] \right) \tag{3.5}$$

To train the GAN generator, we calculate the loss when comparing predictions for generated samples $p_i = D(G(z_i))$ to the response $y_i = 1$. Therefore for the GAN generator, minimizing the loss function (Equation 3.1) can be written as:

$$\min_{\theta_G} \left(\mathbb{E}_{z \sim N(0,1)} \ln \left[D(G(z)) \right] \right) \tag{3.6}$$

The *Wasserstein loss* requires 1 and -1 as labels, rather than 1 and 0. Also, the sigmoid activation was removed from the final layer of the discriminator, so that predictions p_i are no longer constrained to fall in the range $[0, 1]$, but instead it can now be any number in the range $(-\infty, \infty)$. For this reason, the discriminator in a WGAN is usually referred to as a **critic**. The Wasserstein loss function is the defined as follows:

$$\mathcal{L} = -\frac{1}{n}\sum_{i=1}^{n} y_i p_i \tag{3.7}$$

To train the WGAN critic, we calculate the loss when comparing predictions for a real sample $p_i = D(x_i)$ to the response $y_i = 1$ and predictions for generated images $p_i = D(G(z_i))$ to the response $y_i = -1$. Therefore for the WGAN critic, minimizing the loss function can be written as:

$$\max_{\theta_D} \left(\mathbb{E}_{x\sim\gamma} \left[D(x) \right] - \mathbb{E}_{z\sim N(0,1)} \left[D(G(z)) \right] \right) \tag{3.8}$$

To train the WGAN generator, we calculate the loss when comparing predictions for generated samples $p_i = D(G(z_i))$ to the response $y_i = 1$. Therefore for the WGAN generator, minimizing the loss function can be written as:

$$\min_{\theta_G} \left(\mathbb{E}_{z\sim N(0,1)} D\left(G\left(z \right) \right) \right) \tag{3.9}$$

The main differences between Wasserstein loss and Binary Cross-Entropy (BCE) loss is that, the discriminator under the BCE loss outputs a value between 0 and 1, while in W-loss will output any number. Additionally, the structure of the cost function is very similar, but W-loss does not have any logarithms within it, and that is because it is a measure of how far the prediction of the critic for the real ($D(x)$) is from its prediction from the generated $D(G(z))$. The critic is no longer bounded, and just trying to separate the two distributions as much as possible, and as result, the critic is allowed to improve without degrading its feedback back to the generator. This occurs because, it does not have a vanishing gradient problem, and this mitigates against mode collapse, because the generator always gets useful feedback.

3.2.2 The Lipschitz constraint

As the Wasserstein loss does not restrict the output with a sigmoid function, the loss can therefore be very large. The authors of the paper [Arjovsky et al., 2017] show that for the Wasserstein loss function to work, the critic needs to be a 1-Lipschitz continuous

function. We say the critic is 1-Lipschitz continuous if it satisfies the following inequality for any two input samples, x_1 and x_2:

$$\frac{|D(x_1) - D(x_2)|}{|x_1 - x_2|} \leq 1 \tag{3.10}$$

Here, $|x_1 - x_2|$ is the absolute difference between two samples and $|D(x_1) - D(x_2)|$ is the absolute difference between the discriminator predictions. Essentially, we requiere a limit on the rate at witch the predictions of the critic can change between two samples, i.e., the absolute value of the gradient must be at most 1 everywhere.

Figure 3.6 shows an example of a 1D Lipschitz continuous function. Let $f : \mathbb{R} \to \mathbb{R}$ (black curve), $f(\cdot)$ is 1-Lipschitz continuous because for all x in its domain, the norm of its gradient is, at most, 1; graphically we can plot two lines (with slopes $m = 1$ and $m = -1$) and the function remains entirely inside those lines (green area).

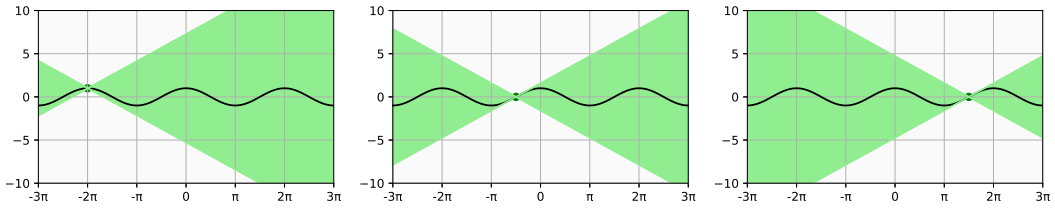


Figure 3.6: There exists a double region (green area) such that wherever it is placed on $f(x)$, the black curve, the function always remains entirely inside.

This condition on the critics neural network is important for W-Loss because it assures that the W-Loss function is not only continuous and differentiable, but also that it does not grow too much and maintain some stability during training.

There are two main methods to enforce one Lipschitz continuity on the critic, namely *weight clipping* and *gradient penalty*.

3.2.3 Weight clipping

With this method the weights of the critics neural network are forced to take values between a fixed interval. After the weights were updated during gradient descent, this method will clip any weights outside of the desired interval.

Basically that means the weights over that interval, either too high or too low, will be set to the maximum or the minimum amount allowed. Arjovsky, et al., show how it is possible to enforce the Lipschitz constraint by clipping the weights of the critic to lie within a small range, $[-0.01, 0.01]$, after each training batch.

This method has a downside: If the clipping parameter is large, then it can take a long time for any weights to reach their limit, thereby making it harder to train the critic till optimality. If the clipping is small, this can easily lead to vanishing gradients when the number of layers is big.

Not only is this trying to do 1-Lipschitz continuity enforcement, this might also limit the critic too much. Or on the other hand, it might actually limit the critic too little if we do not clip the weights enough.

3.2.4 Gradient penalty

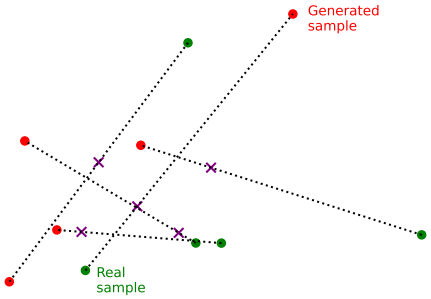
A much softer way to enforce the critic to be 1-Lipschitz continuous is with the gradient penalty. All we need to do is to add a regularization term to the loss function, as indicated by Equation (3.11):

$$\min_{\theta_G} \max_{\theta_C} \underbrace{\mathbb{E}_{x\sim\gamma}\left[D(x)\right] - \mathbb{E}_{z\sim N(0,1)}\left[D(G(z))\right]}_{\text{Original critic loss}} + \underbrace{\lambda \mathbb{E}_{\hat{x}\sim\delta}\left[\left(\left|\left|\nabla_{\hat{x}} D\left(\hat{x}\right)\right|\right|_2 - 1\right)^2\right]}_{\text{Gradient penalty}}. \tag{3.11}$$

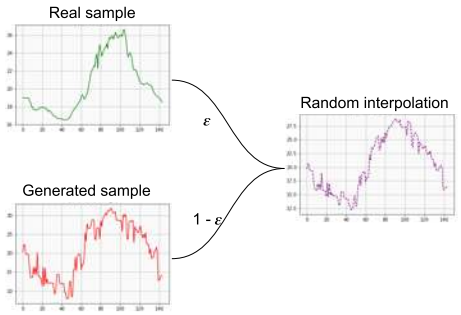
We implicitly define δ sampling uniformly along straight lines between pairs of points sampled from the real data distribution γ and the generator distribution $N(0, 1)$ [Gulrajani et al., 2017]. This is motivated by the fact that the original optimal critic contains straight lines with gradient norm 1 connecting coupled points form γ to $N(0, 1)$.

It is intractable to calculate this gradient everywhere during the training process, so instead the WGAN-GP evaluates the gradient at only a handful of points. To ensure a balanced mix, we use a set of interpolated samples that lie at randomly chosen points along lines connecting the batch of real samples to the batch of fake samples pairwise, as shown in Figure 3.7a.

What the regularization term on Equation (3.11) does to the W-loss function, is



(a) Sampling uniformly along straight lines between pair of points sample from the real distribution and the normal distribution.



(b) Interpolation for one real time series scenario and one generated.

Figure 3.7: Interpolation between real and generated samples.

that it penalizes the critic when its gradient norm is higher than one. λ is just a hyperparameter value of how much to weigh this regularization term against the main loss function, and $\hat{x}$ is an interpolated sample (Figure 3.7b).

The gradient penalty loss measures the squared difference between the norm of the gradient of the predictions with respect to the input samples and 1. The model will naturally be inclined to find weights that ensure the gradient penalty term is minimized, thereby encouraging the model to conform to the Lipschitz constraint (see Algorithm 3.2).

Algorithm 3.2: Gradient_Penalty(Discriminator, real_samples, generated_samples)

1 $\varepsilon = U[0, 1]$
2 interpolated $= \varepsilon \cdot$ real_samples $+ (1 - \varepsilon) \cdot$ generated_samples
3 pred = Discriminator(interpolated)
4 grads = Discriminator.UpdateWeights([pred])
5 norm = Norm(grads)
6 **return** $(\text{norm} - 1)^2$

Algorithm 3.3 shows the implementation of the Wasserstein loss with gradient penalty, where D represents the discriminator (critic) and G represents the generator.

Algorithm 3.3: WGAN_GP(Generator, Discriminator, n_epochs, batch_size, n_critic, λ)

```
for i = 0 to n_epochs do
    // Update the discriminator:
    for j = 0 to n_critic do
        z = Get_Noise_Vectors(batch_size)
        generated = Generator(z)
        real = Real_Samples(batch_size)
        gp = Gradient_Penalty(Discriminator, real, generated)
        loss = Discriminator(generated) - Discriminator(real) + λ gp
    Discriminator.UpdateWeights(loss)
    // Update generator:
    z = Get_Noise_Vectors(batch_size)
    generated = Generator(z)
    G_model.UpdateWeights(Discriminator(generated))
```

3.3 Inception score

Deep generative models are powerful tools that have produced impressive results in recent years, but unlike other deep learning neural networks models that are trained with a loss function until convergence, a GAN generator model is trained using the discriminator. Both the generator and discriminator are trained together to maintain an equilibrium.

As such, there is no objective loss function used to train the GAN generator model and no way to objectively assess the progress of the training and the relative or absolute quality of the model from loss alone.

The advances made by the GANs have been for the most part empirically driven [Barratt and Sharma, 2018], and it is common to periodically generate and save samples during the model training process and use subjective human evaluation on the generated samples in order to both evaluate the quality of the generated samples and to select a final

generator model.

An early adopted example of an objective evaluation method for GAN generated samples is the Inception Score (IS) proposed by Tim Salismans, et al. in their 2016 paper titled "Improved Techniques for Training GANs".

3.3.1 Definition

The Inception Score (IS) is an objetive metric for evaluating the quality of generated samples. It involves the use of a pre-trained neural network model for classification to classify the generated samples. The pre-trained neural network is used to compute the probability of the sample belonging to each class over a large number of generated samples. These predictions are then summarized into the inception score.

The score measures two things simultaneously:

- Diversity: the samples should have variety, and
- Quality: each sample distinctly looks like something specific.

Inception score has a lowest value of 1 and a highest value of the number of classes supported by the classification model; in this thesis the model we will use supports three classes, and as such, the highest inception score on this thesis is three.

3.3.2 Calculation

The inception score is calculated by first using a pre-trained neural network to predict the class probabilities for each generated sample. These are conditional probabilities, e.g. class label conditional on the generated sample. Samples that are classified strongly as one class over all other classes indicate a high quality. As such, the conditional probability of all generated images in the collection should have a low entropy [Salimans et al., 2016].

The entropy is calculated as the negative sum of each observed probability multiplied by the log of the probability. The conditional probability captures our interest in the quality.

$$\text{Entropy} = -\sum_{i=1}^{n} p_i \ln(p_i) \tag{3.12}$$

To capture our interest in a variety of samples, we use the marginal probability. This is the probability distribution of all generated samples. We would prefer the sum of all the values from the marginal probability distribution to have a high entropy.

Combining these two elements by calculating the Kullback-Leibler (KL) divergence, between the conditional and marginal probability distribution for all the generated samples. Finally, the metric proposed at [Salimans et al., 2016] is:

$$IS(G) = \exp\left(\mathbb{E}_{x\sim p_g} KL\big(p\left(y|x\right)||p\left(y\right)\big)\right) \tag{3.13}$$

where $x \sim p_g$ indicates that x is a sample taken from p_g, $KL\left(p(y|x)||p(y)\right)$ is the KL-divergence between the distributions $p(y|x)$ and $p(y)$, $p(y|x)$ is the conditional class distribution, and $p(y)$ is the marginal class distribution. The exp in the expression is there to make the values easier to compare.

3.3.3 Implementation

The samples are split into several groups (Salismans et al., used 10 groups), and the inception score is calculated on each group of images, then the average is reported.

The calculation of the IS on a batch of samples involves first using the neural network to classify and calculate the conditional probability for each sample ($p(y|x)$). The marginal probability is then calculated as the average of the conditional probabilities for the images in the batch ($p(y)$).

The KL divergence is calculated for each sample as the conditional probability multiplied by the log of the conditional probability minus the log of the marginal probability:

$$\text{KL divergence} = P(y|x)\big[\log p(y|x) - \log p(y)\big] \tag{3.14}$$

The KL divergence is then summed over all samples and averaged over all classes and the exponent of the result is calculated to give the final score [Salimans et al., 2016].

Algorithm 3.4 implements the official inception score used when reported in several papers that use the score.

Algorithm 3.4: Inception_Score(p_yx, n_split)

```
scores = list()
n_part = ⌊ p_yx.length[0] / n_split ⌋
n_part = Floor(p_yx.length / n_split)
for i = 1 to n_split do
    subset ← p_yx[i · n_part : (i+1) · n_part]
    p_y ← Mean(p_yx)
    kl_d ← p_yx · (Log(p_yx) - Log(p_y))
    sum_kl ← Sum(kl_d)
    avg ← Mean(sum_kl)
    IS_score ← Exp(avg)
    scores.append(is_score)
end
return Mean(scores)
```

3.4 Chapter conclusions

In this chapter we explored the implementation of the generative adversarial networks [Goodfellow et al., 2014], some of their most common problems, and a some theoretical changes in order to improve the results by enforcing the constraint 1-Lipschitz: weight clipping [Arjovsky et al., 2017] and gradient penalty [Gulrajani et al., 2017]

All GANs are characterized by a generator versus discriminator (or critic) architecture, with the discriminator trying to distinguish between real and fake samples, and the generator aiming to fool the discriminator. By balancing how these two adversaries are trained, the GAN generator can gradually learn how to produce similar samples to those in the training set.

Chapter 4

Experiments and Results

The main goal of a GAN is to solve the question: If we have a data set of similar objects, such as, a collection of images of handwritten 5's, or images of bees, or images of airplanes, can we artificially generate similar objects? [Wang, 2020].

GANs have shown a great potential to replicate the probability distribution of the original data in order to generate new artificial samples that "look like" the original. Figure 4.1 shows samples from digit 5 taken from the MNIST data set [Deng, 2012], bees drawings taken from Google Draws [Jongejan et al., 1999] and airplanes from CIFAR-10 data set [Krizhevsky et al.,]. For each set of images, the five images on the left were taken from the training set, and the three on the right was produced by a GAN .

Data set	Real samples					Generated samples		
MNIST								
Quick Draw								
CIFAR-10								

Figure 4.1: On each of the data set, the five samples from the mid column were taken from the original data set, and the one on the right are synthetic samples made from the generator.

By gathering similar objects, one GAN can learn the probability distribution and generate a sample with the same (or approximately the same) probability distribution. So, will create sets of time series with similar characteristics in order to produce a new sample, this way if a sunny day is forecast for tomorrow we will be able to produce a sample that look like a sunny day.

This chapter provides a detailed overview of the conducted experiments. Section 4.1 describes the time series used in this thesis and how they were group in order to create sets with the same features. Section 4.2 will describe how to create a GAN and how to train it in order to produce resemble data.

4.1 Time series description

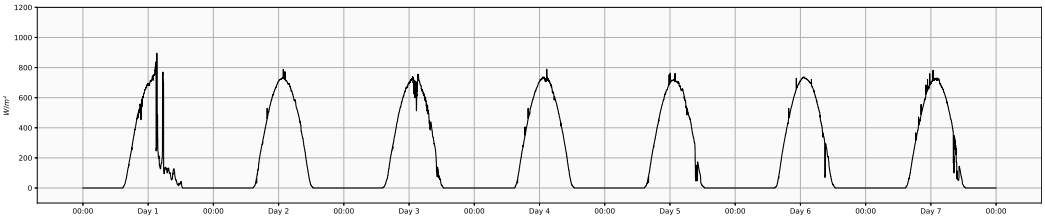
In this thesis we used three different time series: solar irradiance, ambient temperature and wind speed. Figure 4.2 show the data have daily patterns, that lead us to think we can group the data into daily scenarios.

The data from the solar irradiance and ambient temperature were taken form the weather station at Morelia, Michoacán. Both series have 1-minute resolution data, and we have 600 days of data. The data from wind speed was taken from the weather station at Cointzio, Michoacán. This time series has 60-minute resolution data, and we have 357 days of data.

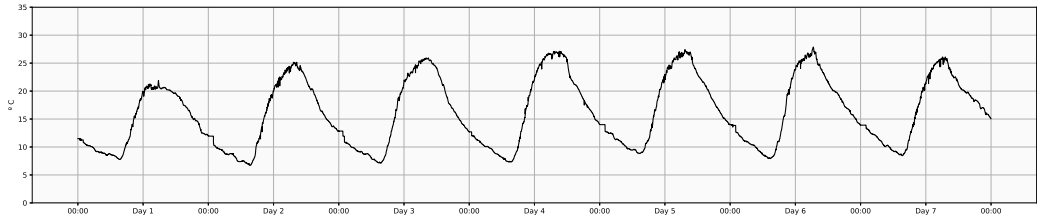
Starting from the whole time series data, we rearrange the time series in order to have daily data shape. This way our data takes into account the fluctuations due to weather conditions for each day. Figure 4.3 shows (on the top) the original data over seven days, and also (on the bottom) show two different daily samples.

The first step in order to group samples, was to fix the time resolution. The original samples, from solar irradiance and ambient temperature, are 1-minute resolution, this means 1440 values per day. Although we could create scenarios with that resolution, we used 10-minute, 30-minute and 60-minute resolution.

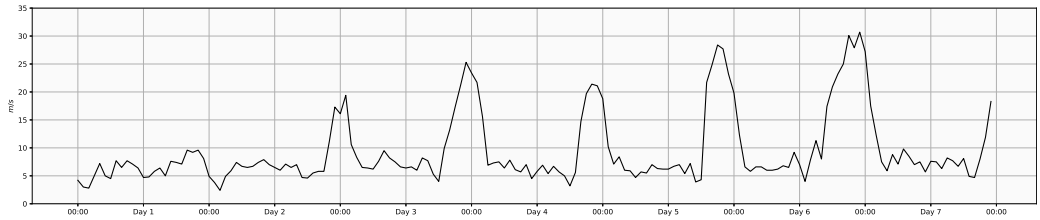
From the daily life it is know there are days with more solar irradiance than others, as well as there are days where the wind speed is lower than others. In this sense, we can



(a) Solar irradiance time series with 1-minute resolution.



(b) Ambient temperature time series with 1-minute resolution.



(c) Wind speed time series with 60-minute resolution.

Figure 4.2: Solar irradiance, ambient temperature and wind speed original time series.

group the daily samples into groups with different values, e.g., low, mid and high values.

From visual inspection, the first day on Figure 4.3 (red curve) have lower solar irradiance than day four (blue curve). This way, we could classify each sample and assign them one label each, e.g., day one could be classify as *low solar irradiance* while day four could be *mid solar irradiance*.

In order to group the data into low, mid and high values, we use the trapezoidal rule, Equation (4.1), to approximate the Area Under the Curve (AUC) , and thus to classify accordingly.

$$\mathcal{A} \approx \frac{1}{2}\left(y_0 + 2\sum_{i=1}^{n-1} y_i + y_n\right) \tag{4.1}$$

By using Equation (4.1) we created three different groups, each with similar characteristics according to the range. However, besides defining groups by AUC, we also used

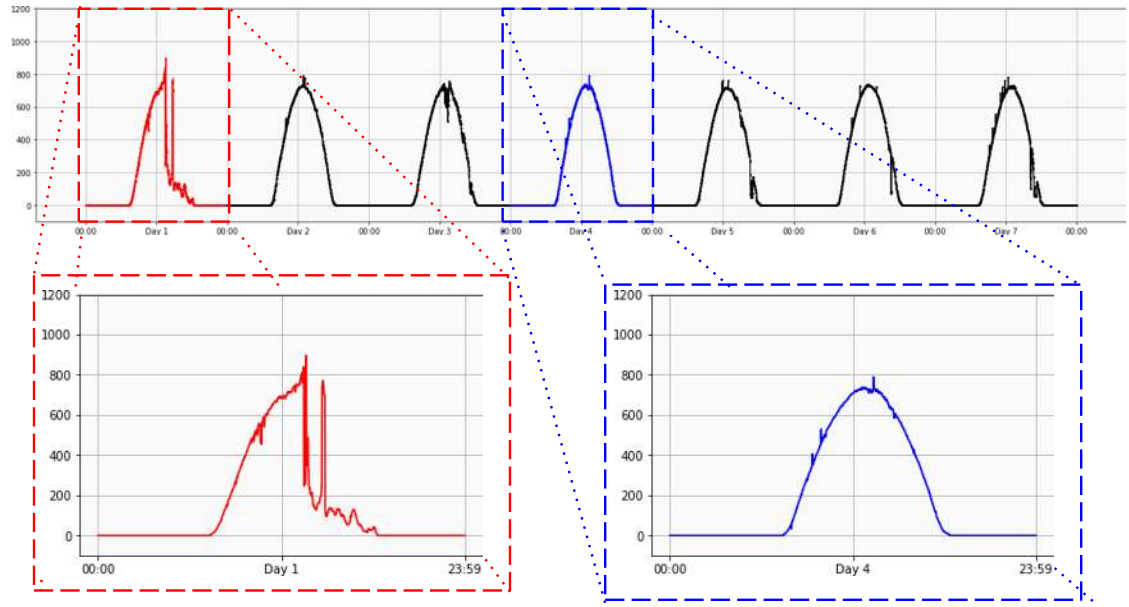


Figure 4.3: The time series data were grouped in order to form daily time series.

the K-means algorithm to create three groups with similar characteristics in the hyper-dimensional space.

Figure 4.4 summarize how we group the time series data. The first step was to fix the time resolution (only for solar irradiance and ambient temperature) in order to perform different experiments. Next, we create the training sets so that each one of them has similar characteristics to each other, by using two criteria to cluster data: area under the curve and the K-means algorithm.

4.2 Experiment description

Once we have group the time series data, we need to construct an adversarial neural network for each group in Figure 4.4 to produce new samples. The generator will transform vectors from the normal distribution into a sequence that should resemble the real samples; and the critic should compute the Wasserstein distance.

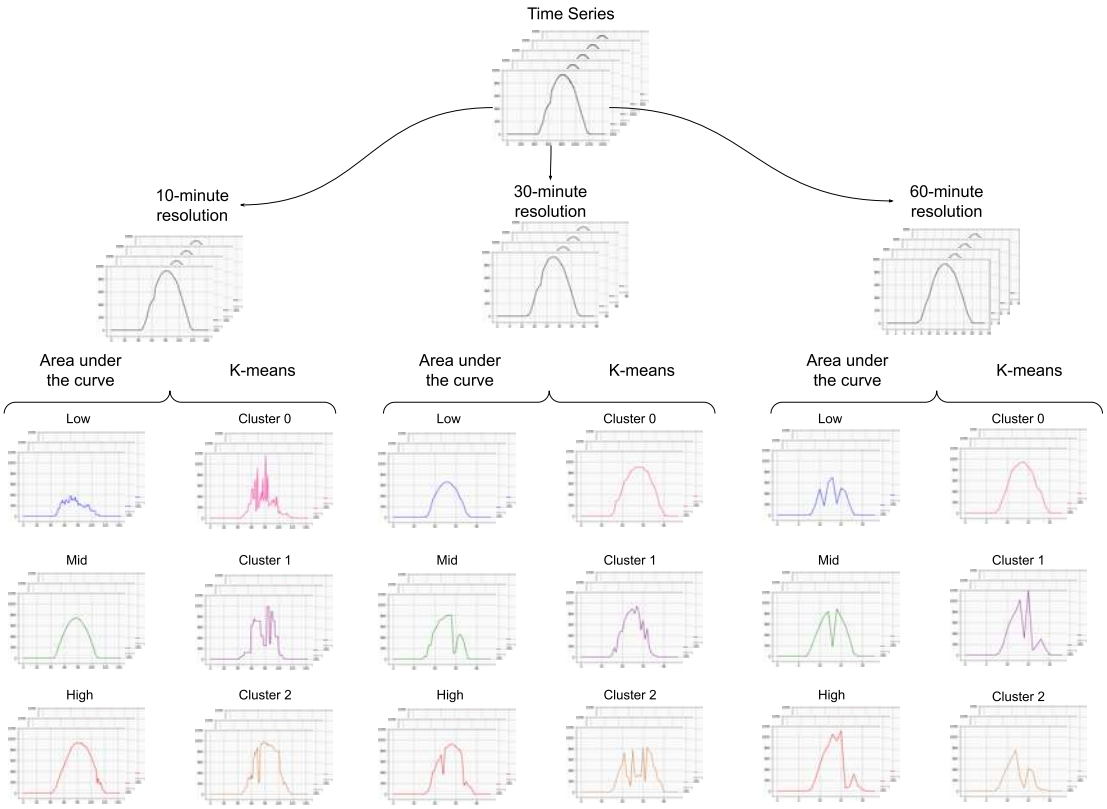


Figure 4.4: The time series data was subsampled with three different time resolutions, and after we use one criterion (area or K-means) to create the scenarios.

From our experiments, we determined that the generator will use a mixture of rectifier linear activations, batch normalization and a linear layer. The generator's inputs is a noise vector sampled from the normal distribution, the hidden layers will be a group of dense, batch normalization and leaky ReLU layers; the output layer is a dense layer with hyperbolic tangent as activation function, and shape equal to the time series resolution, see Figure 4.5.

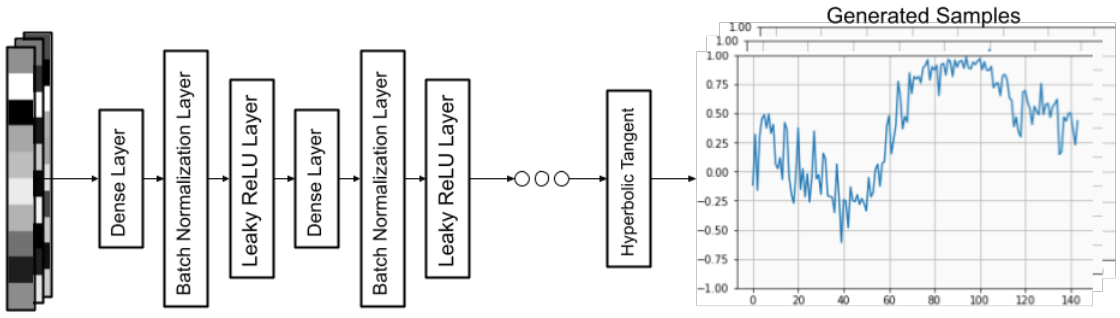


Figure 4.5: The generator receives a noise vector and transform it into a sequence that should have the same characteristics as the real samples.

Instead of using a discriminator to classify, or predict, the probability of generated sample as being real or fake, the Wasserstein GAN changes or the discriminator model with a *critic* that scores the "realness" or "fakeness" of a given sample.

The critic used in our experiments also use a rectifier linear activation and dropout layers, but batch normalization should not be used because batch normalization creates correlation between images in the same batch, which makes the gradient penalty loss less effective [Foster, 2019]. The critic's inputs are samples taken from the real data as well from the generated data; the hidden layers are a stack of dense, leaky ReLU and dropout layer; finally, the output will produce one single real number, see Figure 4.6.

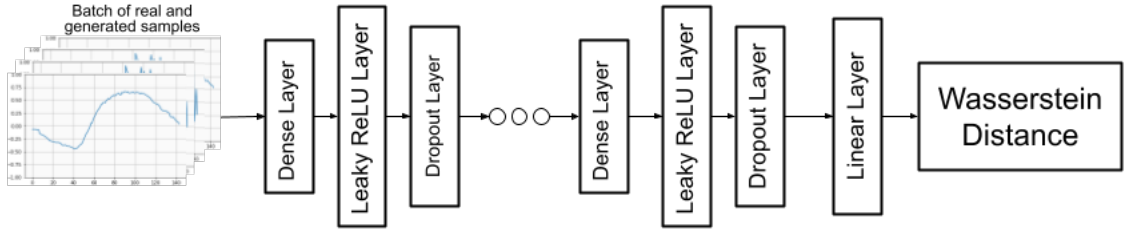


Figure 4.6: The critic on a Wasserstein GAN scores the "realness" of a given sample.

Once we have defined our generator and critic, we implement the WGAN-GP model: For each epoch, we will perform the following stapes as laid out in Algorithm 3.3 from page 27.

1. Train the generator and get the generator's loss,

2. Train the critic and get the critic's loss,

3. Calculate the gradient penalty,

4. Multiply this gradient penalty with a constant weight factor,

5. Add the gradient penalty to the discriminator loss.

4.3 Solar irradiance

By plotting the data as daily samples we can observe there are different levels of solar irradiance, that leads us to think we can create scenarios according to how much solar irradiance we would expect. Figure 4.7 shows six different real days: the two from the left seem to have more area under the curve, while the two from the right seem to have lower.

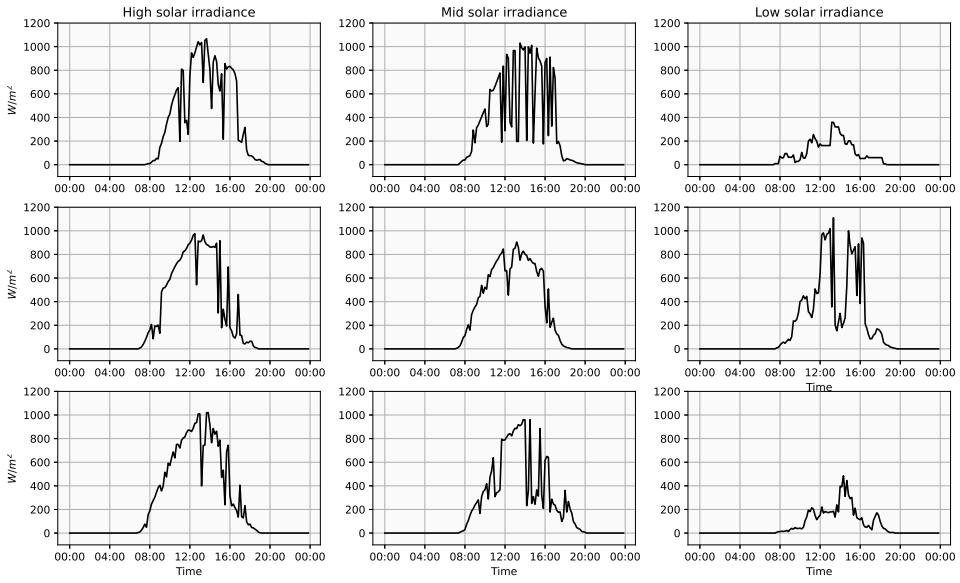


Figure 4.7: The area under the curve of solar irradiance per day varies every day. There are days on which the solar irradiance are greater (plots on the left), and there are days on which is lower (plots on the right).

First, we fixed the time resolution into 10, 30 and 60 minutes, then we will performed two sets of experiments for each time resolution: we used Equation (4.1), to calculate the area under the curve for each day, and thus classify the samples on three categories: high, mid, and low irradiance; and we used K-means to create another three different groups.

4.3.1 10-minute resolution

Subsampling every day into 10-minute resolution produces samples with 144 measurements per day. The results are described below:

Area under the curve. The values of the area under the curve from solar irradiance from the original samples range from 8,298 to 43,906 W/m^2. We created three different scenarios according to low, mid and high solar irradiance.

After training we generated 3,000 samples (1,000 for each group) and calculated the area under the curve from the generated samples and the most of them have area within the correct range. Table 4.1 summarize the results for the solar irradiance data grouped by area under the curve.

Using the inception score for these samples we obtained 2.19, where the maximum possible value is 3. Figure 4.8 shows three examples for each group.

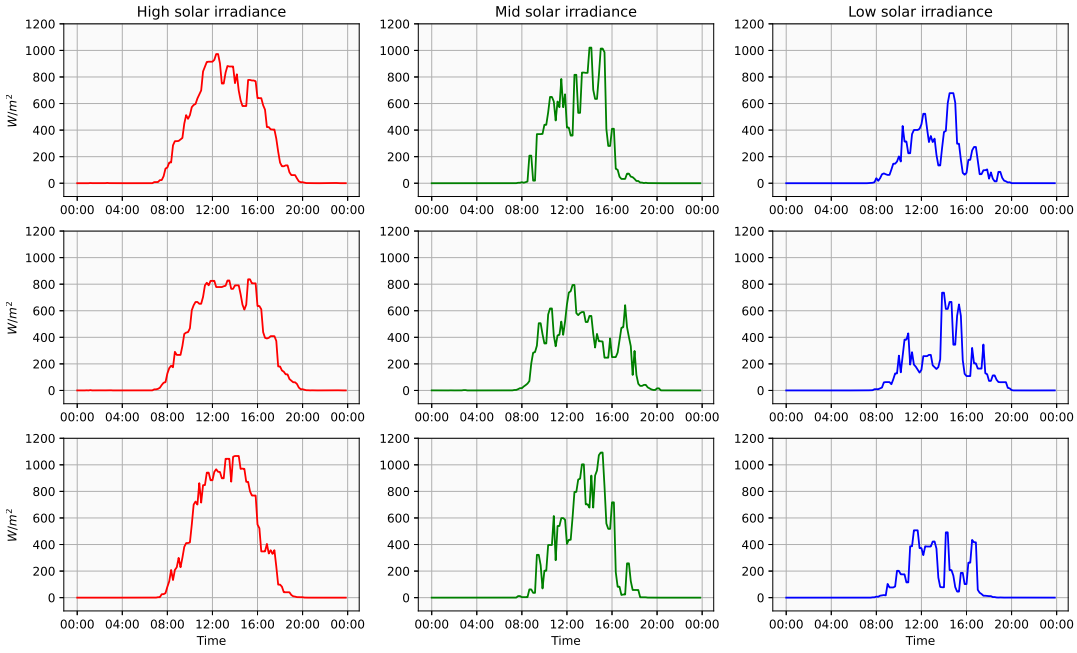


Figure 4.8: Generated samples for solar irradiance 10-minute resolution.

K-means. Grouping the data using K-means algorithm, we created three groups for the time series: classes 0, 1 and 2. We trained a WGAN for each class and then, we use the computed clusters centers to assign the class of the generated samples, and thus verify

if they belong to the correct class. Using the inception score to validate these samples, they score 2.57 over 3. Table 4.1 summarizes the results.

Actual class	Predict class		
	Low	Mid	High
Low	92.5%	7.5 %	0.0%
Mid	13.1%	85.3%	1.6%
High	0.2%	9.3%	90.5%

Area under the curve

Actual class	Predict class		
	Class 0	Class 1	Class 2
Class 0	98.8%	0.76 %	0.44%
Class 1	3.52%	94.3%	2.18%
Class 2	0.86%	2.17%	96.97%

K-means

Table 4.1: Results for 10-minute resolution solar irradiance.

4.3.2 30-minute resolution

Analogously to section 4.3.1, we grouped the samples according to their area under the curve in high, mid, and low solar irradiance; and using the K-means algorithm we created three classes with K-means.

Area under the curve. The values of the area under the curve range from 2,638 to 14,745 W/m^2. The results are summarized in Table 4.2. Figure 4.9 shows three examples for each group. The inception score for the generated samples is 2.73 over 3.

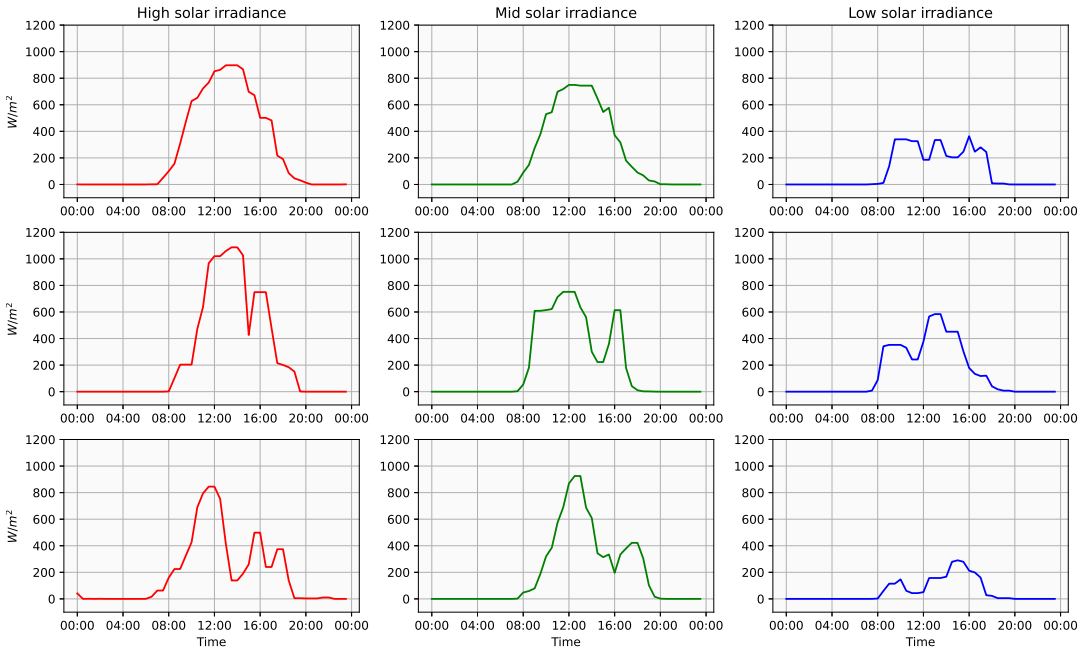


Figure 4.9: Generated samples for solar irradiance 30-minute resolution.

K-means. Using the K-means algorithm we grouped the data into three classes, and calculating the inception score we obtained 2.83 over 3. Table 4.2 summarizes the results after training.

4.3.3 60-minute resolution

The last group of scenarios for solar irradiance were made by subsampling every original sample into 60-minute resolution, i.e., one value for each hour of the day.

Actual class	Predict class		
	Low	Mid	High
Low	96.5%	3.5%	0.0%
Mid	7.1%	92.4%	0.5%
High	0.7%	7.2%	92.1%

Area under the curve

Actual class	Predict class		
	Class 0	Class 1	Class 2
Class 0	97.5%	2.5%	0.0%
Class 1	3.4%	94.3%	2.3%
Class 2	5.2%	3.3%	91.5%

K-means

Table 4.2: Results for 30-minute resolution solar irradiance.

Area under the curve. The values of the area under the curve range from 1,325 to 7,519 W/m^2. The results are summarized in Table 4.3. The inception score for the generated samples is 2.83 over 3; two samples for each group are shown in Figure 4.10.

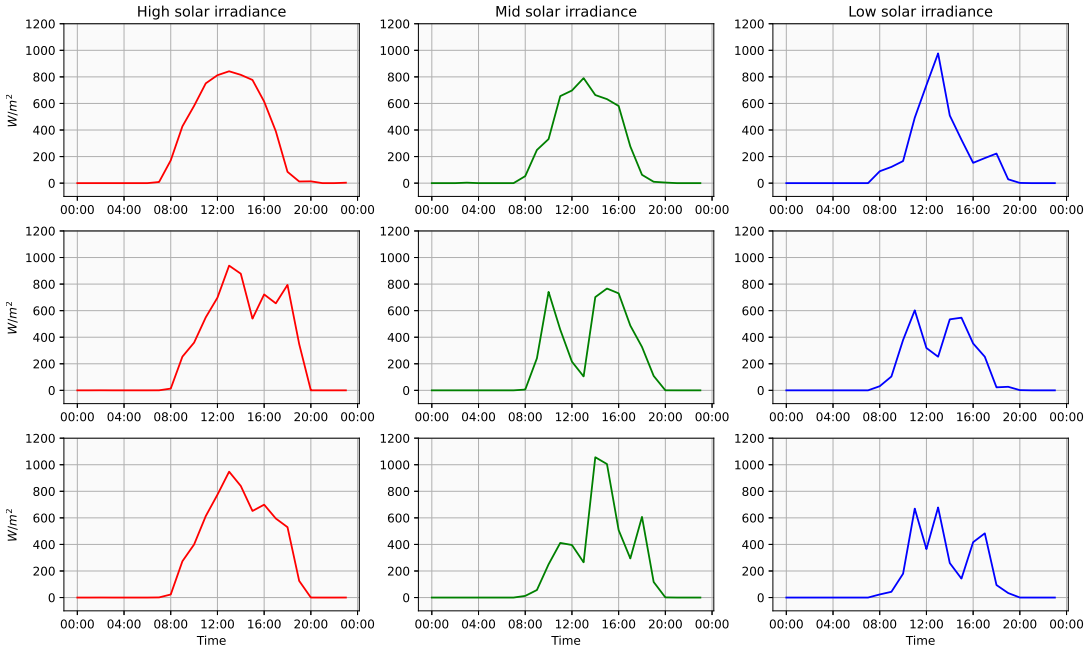


Figure 4.10: Generated samples for solar irradiance 60-minute resolution.

K-means. By grouping using the K-means algorithm, we get an inception score of 2.44 over 3.

Actual class	Predict class		
	Low	Mid	High
Low	96.3%	3.7%	0.0%
Mid	0.1%	92.0%	7.9%
High	0.1%	4.9%	95.0%

Area under the curve

Actual class	Predict class		
	Class 0	Class 1	Class 2
Class 0	96.33%	3.40%	0.27%
Class 1	0.04%	96.44%	3.52%
Class 2	0.41%	5.18%	94.41%

K-means

Table 4.3: Results for 60-minute resolution solar irradiance.

4.4 Ambient temperature

Analogously to section 4.3 Solar irradiance, by plotting the time series we can to group the data into how much warm, or cold, is one day. Figure 4.11 shows two days with high temperature (plots on the left) and two days with low temperature (plots on the right).

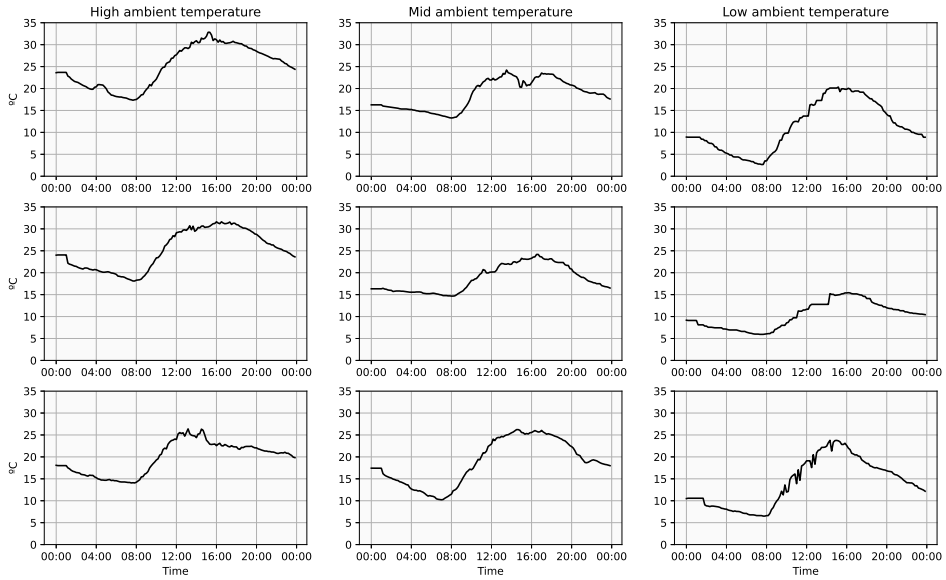


Figure 4.11: The area under the curve per day varies on each day. There are day on which the ambient temperature are greater (plots on the left), and there are day on which is lower (plots on the right).

As well as with the solar irradiance time series, we also create the train sets according to the time resolution (10, 30, and 60 minutes), and after we use the area under

the curve and the K-means algorithm to create the groups (as Figure 4.4).

4.4.1 10-minute resolution

Subsampling every day with 1,440 measurements into 10-minute resolution, will produce samples with 144 measurements per day.

Area under the curve. Figure 4.12 shows three generated samples for each group. The inception score for the generated samples is 2.89 over 3.

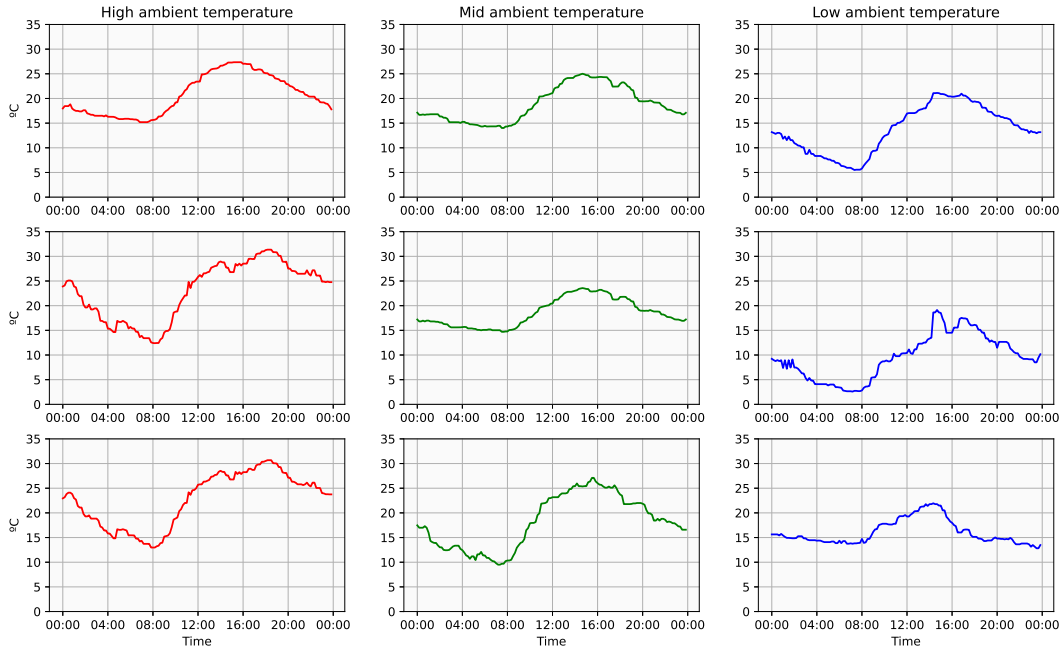


Figure 4.12: Generated samples for ambient temperature 10-minute resolution.

K-means. By grouping using the K-means algorithm, we get an inception score of 2.88 over 3.

4.4.2 30-minute resolution

Subsampling every day with 1440 measurements into 30-minute resolution, will produce samples with 48 measurements per day.

Area under the curve. Figure 4.13 shows three generated samples for each group. The inception score for the generated samples is 2.35 over 3.

Actual class	Predict class		
	Low	Mid	High
Low	99.97%	0.03%	0.0%
Mid	0.0%	98.09%	1.91%
High	0.0%	0.05%	99.95%

Area under the curve

Actual	Predict class		
	Class 0	Class 1	Class 2
Class 0	99.77%	0.23%	0.00%
Class 1	2.78%	97.17%	0.05%
Class 2	0.00%	0.0%	100.0%

K-means

Table 4.4: Results for 10-minute resolution ambient temperature.

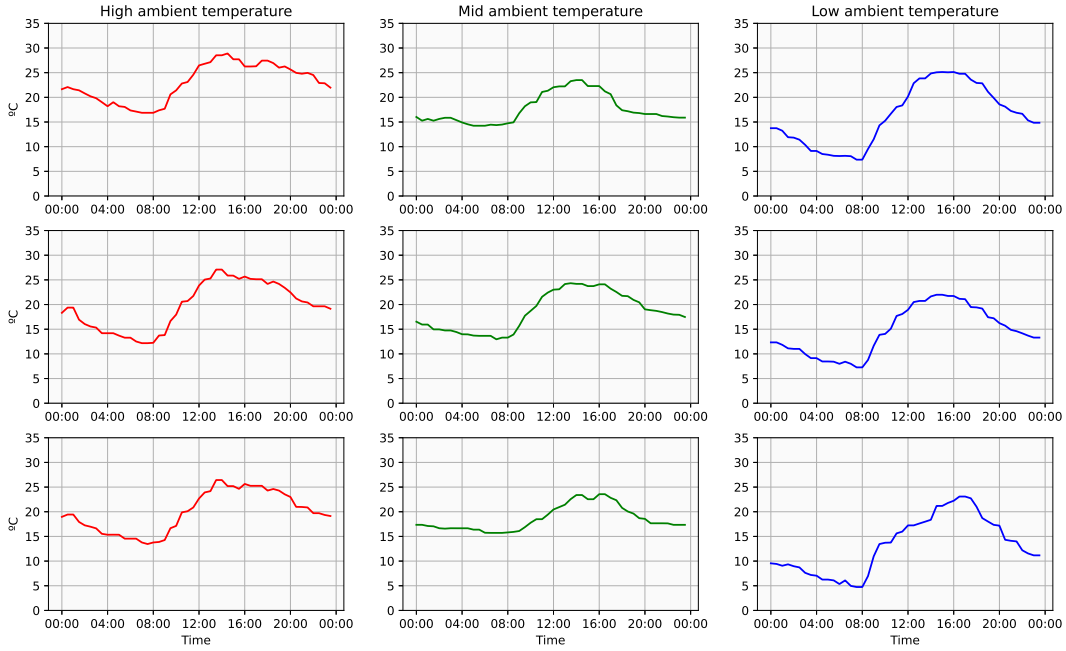


Figure 4.13: Generated samples for ambient temperature 30-minute resolution.

K-means. By grouping using the K-means algorithm, we get an inception score of 2.77 over 3.

4.4.3 60-minute resolution

Subsampling every day with 1440 measurements into 60-minute resolution, will produce samples with 24 measurements per day.

Area under the curve. Figure 4.14 shows three generated samples for each

Actual class	Predict class		
	Low	Mid	High
Low	96.86%	3.13%	0.01%
Mid	0.13%	94.06%	5.80%
High	0.00%	2.20%	97.8%

Area under the curve

Actual class	Predict class		
	Class 0	Class 1	Class 2
Class 0	99.4%	0.6%	0.00%
Class 1	0.62%	97.69%	1.69%
Class 2	0.00%	2.44%	97.56%

K-means

Table 4.5: Results for 30-minute resolution ambient temperature.

group. The inception score for the generated samples is 2.67 over 3.

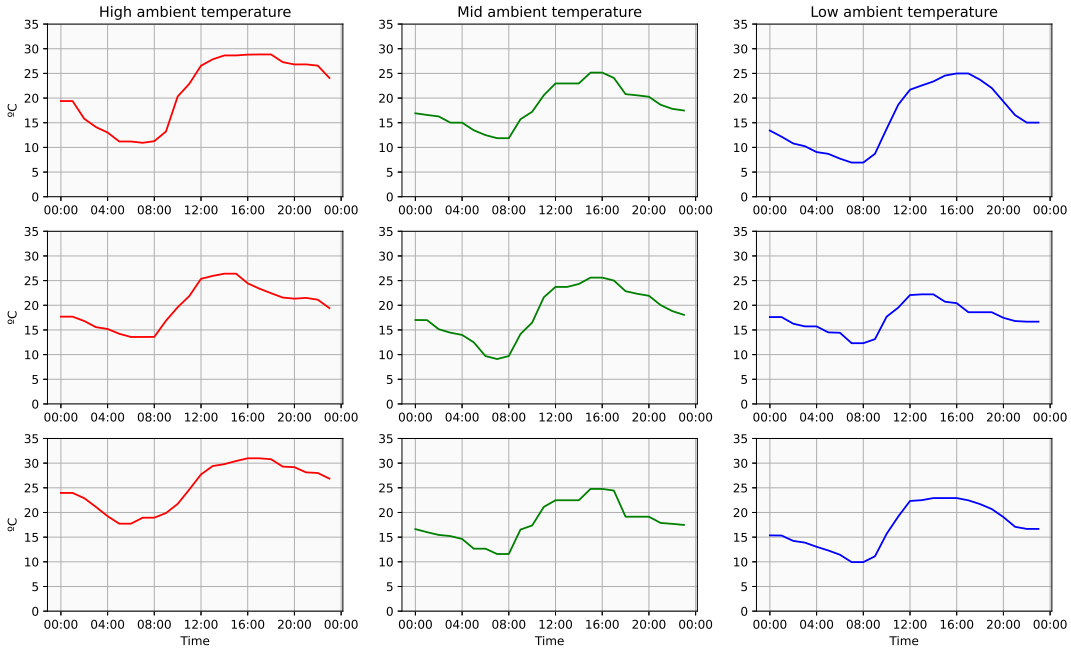


Figure 4.14: Generated samples for ambient temperature 60-minute resolution.

K-means. By grouping using the K-means algorithm, we get an inception score of 2.68 over 3.

Actual class	Predict class		
	Low	Mid	High
Low	99.93%	0.06%	0.00%
Mid	0.00%	100%	0.00%
High	0.00%	0.00%	0.00%

Area under the curve

Actual class	Predict class		
	Class 0	Class 1	Class 2
Class 0	99.95%	0.05%	0.00%
Class 1	0.40%	97.38%	2.22%
Class 2	0.00%	0.00%	100%

K-means

Table 4.6: Results for 60-minute resolution ambient temperature.

4.5 Wind speed

This time series does not have 1-minute resolution, instead have hourly data. Because of this time resolution, we could not make groups with 10 and 30-minute resolution as in sections 4.3 Solar irradiance and 4.4 Ambient temperature. We could only group the data by area under the curve and the K-means algorithm with 60-minute resolution.

Figure 4.15 shows the plots of six samples. As we can observe, the plots on the left have more area under the curve than the plots on the right. As in previous experiments, we group the training sets according to the area under the curve, and then we used K-means to perform another set of experiments.

4.5.1 60-minute resolution

Area under the curve. Figure 4.16 shows two generated samples for each group. The inception score for the generated samples is 2.77 over 3.

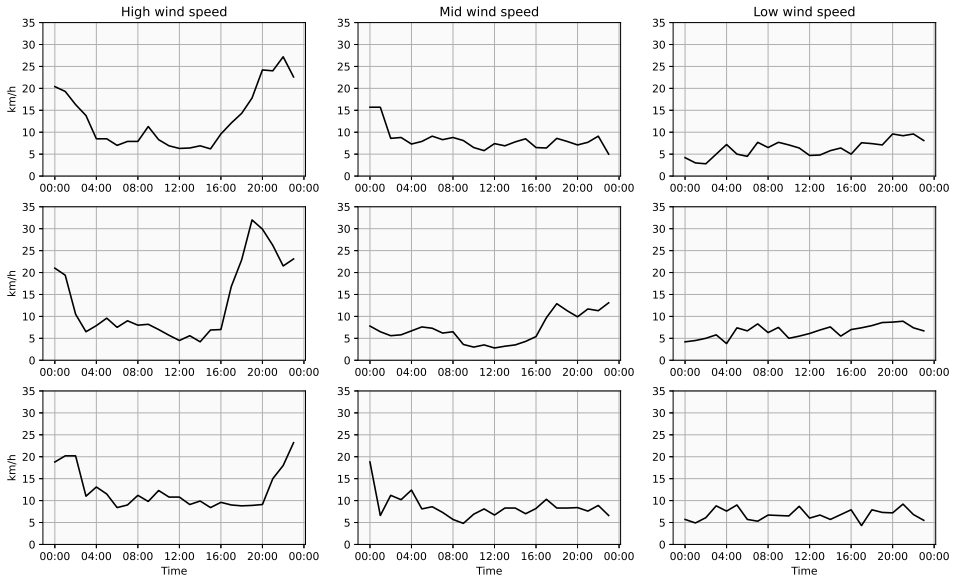


Figure 4.15: Classifying the original samples according to the area under the curve.

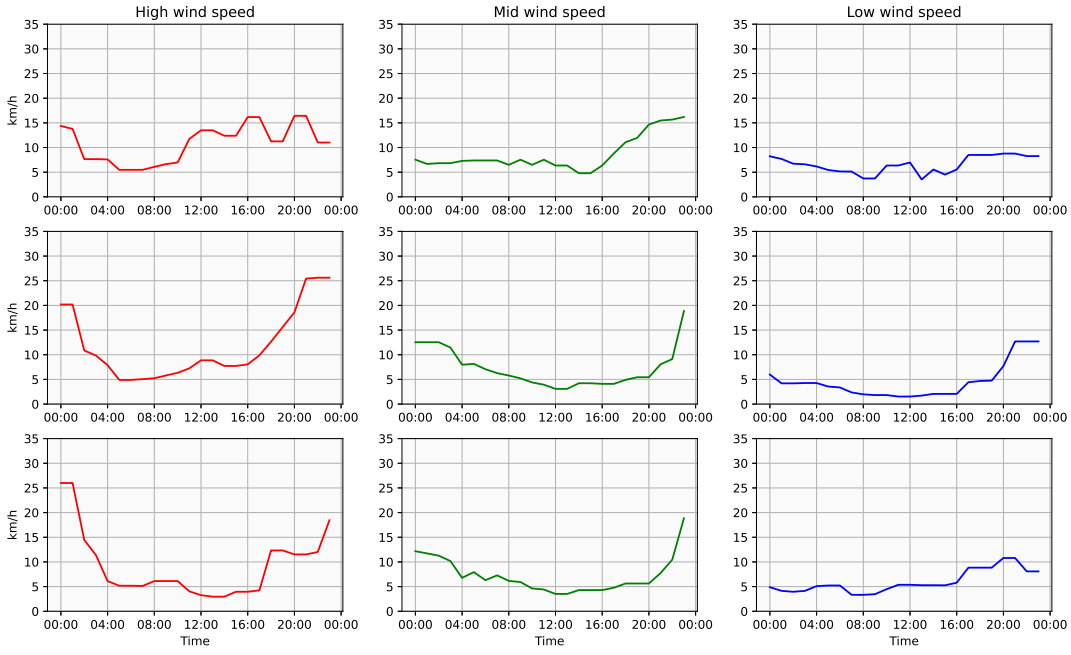


Figure 4.16: Generated samples for wind speed 60-minute resolution.

K-means. By grouping using the K-means algorithm, we get an inception score of 2.77 over 3.

Actual class	Predict class		
	Low	Mid	High
Low	82.26%	17.73%	0.0%
Mid	7.13%	92.13%	0.73%
High	0.20%	10.0%	89.8%

Area under the curve

Actual class	Predict class		
	Class 0	Class 1	Class 2
Class 0	99.52%	0.43%	0.05%
Class 1	0.01%	81.85%	18.14%
Class 2	0.05%	0.27%	99.68%

K-means

Table 4.7: Confusion matrix for 60-minute resolution wind speed.

4.6 Chapter conclusions

After the whole set of experiments we can affirm our WGANs with gradient penalty, can replicate the patterns of the time series. By visual inspection we can validate the generated samples include the fluctuations of weather as the real samples, and thus we can create plausibles scenarios.

Besides the visual inspection, we calculate the inception score (as detailed in section 3.3 from page 27) to really have a quantitative measure to assess the generated samples. Table 4.8 shows the inception score obtain for our scenarios. Note that the score is slightly better when we group the time series by K-means.

Time resolution	Solar irradiance		Ambient temperature		Wind speed	
	AUC	K-means	AUC	K-means	AUC	K-means
10-minute resolution	2.120	2.511	2.898	2.884	–	–
30-minute resolution	2.731	2.835	2.352	2.771	–	–
60-minute resolution	2.834	2.445	2.673	2.688	2.773	2.777

Table 4.8: Results for the Inception Score for all the scenarios.

Chapter 5

Conclusions

This chapter summarizes the characteristics of the data, the results obtained in Chapter 4, interpretation of results as well as some ideas that may serve to conduct future research.

5.1 General conclusions

We started this thesis with three different time series: solar irradiance, ambient temperature, and wind speed. From the data, we created samples for each day and grouped these samples to create a training set with the same characteristics (as shown in Figure 4.4 from page 35).

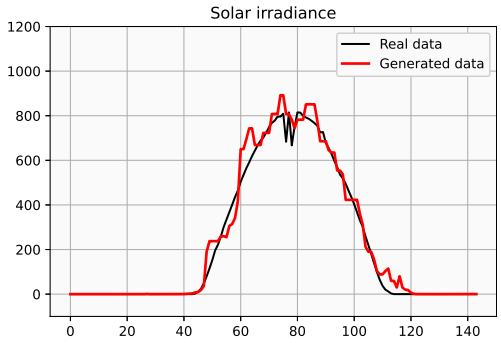
Once we have the different training set, we train a WGAN architecture so the generator will be able to replicate the distribution from the original data.

We were able to produce realistic synthetic days according to different scenarios. Having data at 1-minute resolution, allowed us to subsampling the time series data and create different scenarios for different purposes (although we could also generate a scenario with 1 minute resolution).

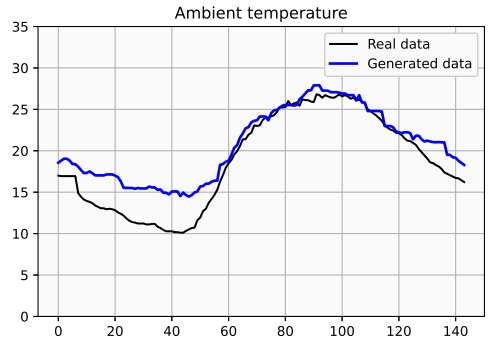
Scenario generation can help model the uncertainties and variations in renewables generation, and it is an essential tool for decision-making in power grids with high penetration of renewables. This thesis proved WGANs performs well for scenario generation for solar irradiance, ambient temperature, and wind speed.

Figure 5.1 shows one scenario for each time series. Figure 5.1a shows one real and one generated scenario, Figure 5.1b the two scenarios are from ambient temperature, and Figure 5.1c shows scenarios for wind speed. From visual inspection we can notice the generated samples are quite similar to the real samples.

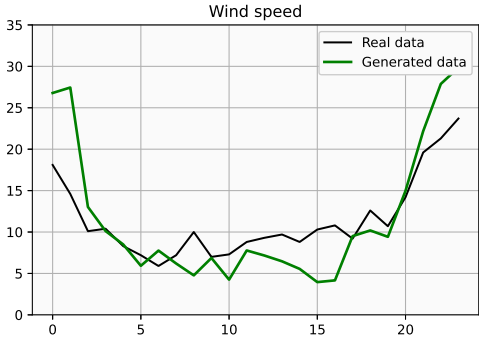
In summary, the results illustrate presented in Chapter 4 shows that generative adversarial networks with gradient penalty can generate new daily scenarios with high quality that capture the intrinsic features of the original data (including range, fluctuations, patterns, etc.), and not to simply memorize the training data.



(a) Solar irradiance scenarios. The black curve is the real sample, and the red one is a generated sample.



(b) Ambient temperature scenarios. The black curve is the real sample, and the blue one is a generated sample.



(c) Wind speed scenarios. The black curve is the real sample, and the green one is a generated sample.

Figure 5.1: Scenario comparison. The curves on black correspond to real samples from solar irradiance, ambient temperature and wind speed, respectively. We can see, the generated scenario has the same shape as the real sample.

5.2 Future work

Finally, we present some ideas that may serve as areas of potential research, and may lead to fruitful results:

5.2.1 Use deep convolutional GANs

We implemented a WGAN with gradient penalty whose generator and critic were simple feed-forward neural networks. Despite the simplicity, many of the generated scenarios that the generator produced after being fully trained were convincing (as shown in Figure 5.1).

We believe we can create more plausible scenarios with a more powerful network architecture. Instead of simple feed-forward networks, both the generator and the critic can be implemented as convolutional neural networks (CNNs, or ConvNets). The resulting GAN architecture is know as *Deep Convolutional GAN* [Foster, 2019].

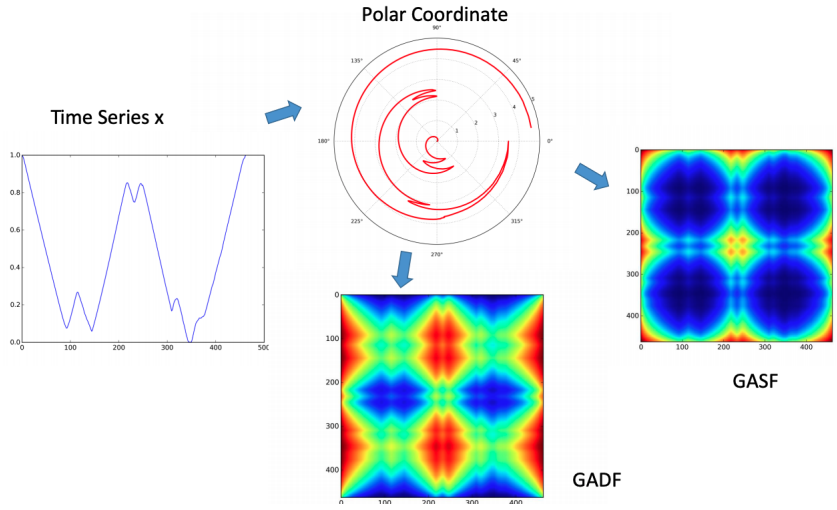
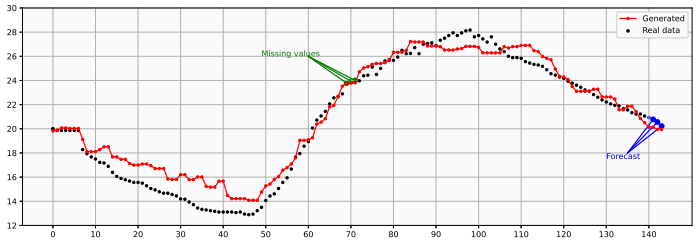


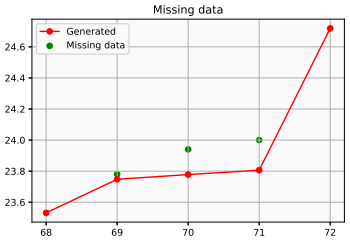
Figure 5.2: Encoding map of Gramian Angular Fields. [Wang and Oates, 2015] transformed a sequence into a polar coordinate system and calculate its Gramian Angular Summation/Difference Fields.

5.2.2 Forecasting and missing data

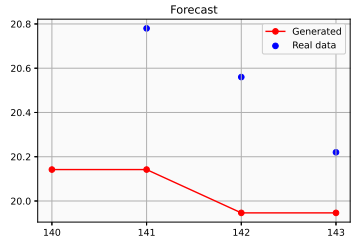
We can use these generated scenarios to obtain missing data and for the forecasting task. If given daily scenario with some missing data, or if we want to forecast the next values, we could generated a scenario similar enough to the real data and use the generated data to find the missing data, or to make predictions, see Figure 5.3.



(a) The generated scenario (red) resembles the true data (black), and we can find a good approximation to the missing values (green) and make a forecast prediction (blue).



(b) A closer view for the missing data. In red, we plot the scenario made by a generator, the green dots are the real values.



(c) From the generated scenario we can forecast the next values of the time series. The blue points shows the actual values.

Figure 5.3: Given a scenario for time series (black dots) we can find missing values and make forecasting prediction.

References

[Adhikari and Agrawal, 2013] Adhikari, R. and Agrawal, R. K. (2013). An introductory study on time series modeling and forecasting. *arXiv preprint arXiv:1302.6613*.

[Ahirwar, 2019] Ahirwar, K. (2019). *Generative Adversarial Networks Projects: Build next-generation generative models using TensorFlow and Keras*. Packt Publishing.

[Antipov et al., 2017] Antipov, G., Baccouche, M., and Dugelay, J.-L. (2017). Face aging with conditional generative adversarial networks. In *2017 IEEE International Conference on Image Processing (ICIP)*, pages 2089–2093.

[Arjovsky et al., 2017] Arjovsky, M., Chintala, S., and Bottou, L. (2017). Wasserstein gan.

[Barratt and Sharma, 2018] Barratt, S. and Sharma, R. (2018). A note on the inception score.

[Bigdeli, 2016] Bigdeli, N.; Salehi Borujeni, M. A. K. (2016). Time series analysis and short-term forecasting of solar irradiation, a new hybrid approach. *Swarm and Evolutionary Computation*.

[Bishop, 2006] Bishop, C. M. (2006). *Pattern Recognition and Machine Learning*. Information science and statistics. Springer, 1st ed. 2006. corr. 2nd printing edition.

[Brock et al., 2019] Brock, A., Donahue, J., and Simonyan, K. (2019). Large scale gan training for high fidelity natural image synthesis.

[Brownlee, 2019] Brownlee, J. (2019). *Generative Adversarial Networks with Python: Deep*

Learning Generative Models for Image Synthesis and Image Translation. Machine Learning Mastery.

[Brundage et al., 2018] Brundage, M., Avin, S., Clark, J., Toner, H., Eckersley, P., Garfinkel, B., Dafoe, A., Scharre, P., Zeitzoff, T., Filar, B., Anderson, H., Roff, H., Allen, G. C., Steinhardt, J., Flynn, C., , Beard, S., Belfield, H., Farquhar, S., Lyle, C., Crootof, R., Evans, O., Page, M., Bryson, J., Yampolskiy, R., and Amodei, D. (2018). The malicious use of artificial intelligence: Forecasting, prevention, and mitigation.

[Conejo, 2010] Conejo, J. M. R. M. A. (2010). A methodology to generate statistically dependent wind speed scenarios. *Applied Energy*, 87.

[Csáji, 2001] Csáji, B. C. (2001). Approximation with artificial neural networks. Master's thesis, Eötvos Loránd University (ELTE), Budapest, Hungary.

[de Oliveira et al., 2000] de Oliveira, K. A., Vannucci, A., and da Silva, E. C. (2000). Using artificial neural networks to forecast chaotic time series. *Physica A: Statistical Mechanics and its Applications*, 284.

[Deng, 2012] Deng, L. (2012). The mnist database of handwritten digit images for machine learning research. *IEEE Signal Processing Magazine*, 29(6):141–142.

[Foster, 2019] Foster, D. (2019). *Generative Deep Learning: Teaching Machines to Paint, Write, Compose, and Play*. O'Reilly Media.

[Geiger et al., 2020] Geiger, A., Liu, D., Alnegeimish, S., Infante, A. C., and Veeramacheneni, K. (2020). Tadgan: Time series anomaly detection using generative adversarial networks.

[Géron, 2017] Géron, A. (2017). *Hands-On Machine Learning with Scikit-Learn and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems*. O'Reilly Media.

[Goodfellow et al., 2016] Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep Learning*. Adaptive Computation and Machine Learning series. MIT Press.

[Goodfellow, 2017] Goodfellow, I. J. (2017). NIPS 2016 tutorial: Generative adversarial networks. *CoRR*, abs/1701.00160.

[Goodfellow et al., 2014] Goodfellow, I. J., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y. (2014). Generative adversarial networks.

[Granger et al., 2014] Granger, C., Newbold, P., and Shell, K. (2014). *Forecasting Economic Time Series*. Elsevier Science.

[Gulrajani et al., 2017] Gulrajani, I., Ahmed, F., Arjovsky, M., Dumoulin, V., and Courville, A. (2017). Improved training of wasserstein gans.

[He et al., 2015] He, K., Zhang, X., Ren, S., and Sun, J. (2015). Deep residual learning for image recognition.

[Hyndman and Athanasopoulos, 2018] Hyndman, R. and Athanasopoulos, G. (2018). *Forecasting: principles and practice*. OTexts.

[Jongejan et al., 1999] Jongejan, J., Rowley, H., Kawashima, T., Kim, J., and Fox-Gieg, N. (1999). Quick, draw! the data.

[Karras et al., 2018] Karras, T., Aila, T., Laine, S., and Lehtinen, J. (2018). Progressive growing of gans for improved quality, stability, and variation.

[Khashei et al., 2008] Khashei, M., Hejazi, S. R., and Bijari, M. (2008). A new hybrid artificial neural networks and fuzzy regression model for time series forecasting. *Fuzzy Sets and Systems*, 159.

[Kirchgassner and Wolters, 2008] Kirchgassner, G. and Wolters, J. (2008). *Introduction to Modern Time Series Analysis*. Springer.

[Krizhevsky et al.,] Krizhevsky, A., Nair, V., and Hinton, G. Cifar-10 (canadian institute for advanced research).

[Langr and Bok, 2019] Langr, J. and Bok, V. (2019). *GANs in Action: Deep learning with Generative Adversarial Networks*. Manning Publications.

[Lemke and Gabrys, 2010] Lemke, C. and Gabrys, B. (2010). Meta-learning for time series forecasting and forecast combination. *Neurocomputing*, 73.

[Luer et al., 2019] Luer, F., Mautz, D., and Bohm, C. (2019). [ieee 2019 international conference on data mining workshops (icdmw) - beijing, china (2019.11.8-2019.11.11)] 2019 international conference on data mining workshops (icdmw) - anomaly detection in time series using generative adversarial networks.

[McCulloch and Pitts, 1943] McCulloch, W. S. and Pitts, W. (1943). A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biology*, 5.

[Palit and Popovic, 2006] Palit, A. and Popovic, D. (2006). *Computational Intelligence in Time Series Forecasting: Theory and Engineering Applications*. Advances in Industrial Control. Springer London.

[Patterson and Gibson, 2017] Patterson, J. and Gibson, A. (2017). *Deep Learning: A Practitioner's Approach*. O'Reilly Media.

[Qiao et al., 2021] Qiao, J., Pu, T., and Wang, X. (2021). Renewable scenario generation using controllable generative adversarial networks with transparent latent space. *CSEE Journal of Power and Energy Systems*, 7(1):66–77.

[Radford et al., 2016] Radford, A., Metz, L., and Chintala, S. (2016). Unsupervised representation learning with deep convolutional generative adversarial networks.

[Reed et al., 2016a] Reed, S., Akata, Z., Mohan, S., Tenka, S., Schiele, B., and Lee, H. (2016a). Learning what and where to draw.

[Reed et al., 2016b] Reed, S., Akata, Z., Yan, X., Logeswaran, L., Schiele, B., and Lee, H. (2016b). Generative adversarial text to image synthesis.

[Rosebrock, 2017] Rosebrock, A. (2017). *Deep Learning for Computer Vision with Python: Starter Bundle*. PyImageSearch.

[Rosenblatt, 1958] Rosenblatt, F. (1958). The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, 65.

[Salimans et al., 2016] Salimans, T., Goodfellow, I., Zaremba, W., Cheung, V., Radford, A., and Chen, X. (2016). Improved techniques for training gans.

[Shirvany et al., 2009] Shirvany, Y., Hayati, M., and Moradian, R. (2009). Multilayer perceptron neural networks with novel unsupervised training method for numerical solution of the partial differential equations. *Applied Soft Computing*, 9.

[Takahashi et al., 2019] Takahashi, S., Chen, Y., and Tanaka-Ishii, K. (2019). Modeling financial time-series with generative adversarial networks. *Physica A: Statistical Mechanics and its Applications*, 527:121261.

[Wang, 2020] Wang, Y. (2020). A mathematical introduction to generative adversarial nets (gan).

[Wang and Oates, 2015] Wang, Z. and Oates, T. (2015). Imaging time-series to improve classification and imputation.

[Xu et al., 2019] Xu, S., Chan, H. K., and Zhang, T. (2019). Forecasting the demand of the aviation industry using hybrid time series sarima-svr approach. *Transportation Research Part E: Logistics and Transportation Review*, 122.

[Zhang, 2003] Zhang, G. (2003). Time series forecasting using a hybrid arima and neural network model. *Neurocomputing*, 50.

[Zhang et al., 2017a] Zhang, H., Xu, T., Li, H., Zhang, S., Wang, X., Huang, X., and Metaxas, D. (2017a). Stackgan: Text to photo-realistic image synthesis with stacked generative adversarial networks.

[Zhang et al., 2017b] Zhang, Z., Song, Y., and Qi, H. (2017b). Age progression/regression by conditional adversarial autoencoder.