



UNIVERSIDAD DE GRANADA

Facultad de Ciencias y Escuela Técnica Superior de Ingenierías
Informática y de Telecomunicación

DOBLE GRADO: INGENIERÍA INFORMÁTICA Y
MATEMÁTICAS

TRABAJO DE FIN DE GRADO

Redes Generativas Adversarias para creación de *deep fakes*

Presentado por:
Silvia Barroso Moreno

Tutor:
Juan Gómez Romero
Departamento de Ciencias de la Computación e Inteligencia Artificial
Francisco Torralbo Torralbo
Departamento de Geometría y Topología

Curso académico 2021-2022

Redes Generativas Adversarias para creación de *deep fakes*

Silvia Barroso Moreno

Silvia Barroso Moreno *Redes Generativas Adversarias para creación de deep fakes .*
Trabajo de fin de Grado. Curso académico 2021-2022.

**Responsable de
tutorización**

Juan Gómez Romero
*Departamento de Ciencias de la Computación
e Inteligencia Artificial*

Francisco Torralbo Torralbo
Departamento de Geometría y Topología

DOBLE GRADO:
INGENIERÍA
INFORMÁTICA Y
MATEMÁTICAS

Facultad de Ciencias y
Escuela Técnica Superior
de Ingenierías Informática
y de Telecomunicación
Universidad de Granada

DECLARACIÓN DE ORIGINALIDAD

D./Dña. Silvia Barroso Moreno

Declaro explícitamente que el trabajo presentado como Trabajo de Fin de Grado (TFG), correspondiente al curso académico 2021-2022, es original, entendida esta, en el sentido de que no ha utilizado para la elaboración del trabajo fuentes sin citarlas debidamente.

En Granada a 19 de junio de 2022

Fdo: Silvia Barroso Moreno

Índice general

Agradecimientos	XI
Resumen	XIII
Summary	XV
Introducción	XIX
1. Contextualización	XIX
2. Descripción del trabajo	XX
3. Estructura del trabajo	XX
4. Bibliografía fundamental	XXI
5. Objetivos del trabajo	XXI
6. Planificación y estimación de costes	XXIII
I. Fundamentos informáticos de las redes generativas adversarias (GANs) y aplicación en la creación de <i>deep fakes</i>	1
1. Redes neuronales	3
1.1. Fundamentos	3
1.1.1. La neurona	3
1.1.2. Arquitectura de la red	4
1.1.3. Funciones de activación	5
1.1.4. Ejemplo de aprendizaje: XOR	5
1.2. Entrenamiento de redes neuronales	6
1.2.1. Función de costo	7
1.2.2. Retropropagación	7
1.2.3. Descenso del gradiente	9
1.3. Mejoras del aprendizaje	10
1.4. Teoremas de Aproximación universal	10
2. Redes convolucionales (CNNs)	13
2.1. Fundamentos	13
2.1.1. CNN y la historia del Deep Learning	13
2.1.2. El operador de convolución	14
2.1.3. Arquitectura de la red	15
2.2. Entrenamiento de CNN	16
2.3. Mejoras del aprendizaje	16
3. Redes Generativas Adversarias (GANs)	17
3.1. Fundamentos	17
3.1.1. Nacimiento de las GANs	17

Índice general

3.1.2. ¿Que son las GANs?	17
3.1.3. Metáforas asociadas a las GANs	18
3.1.4. Funcionamiento	20
3.1.5. ¿Por que estudiar las GANs?	21
3.2. Entrenamiento de GANs	22
3.2.1. Funciones de coste	24
3.2.2. Dificultades del aprendizaje	27
3.3. Tipos de GANs	27
3.3.1. DCGAN	28
3.3.2. Versiones principales de la configuración de las GANs	28
3.3.3. ProGAN	30
3.3.4. SGAN	31
3.3.5. CGAN	32
3.3.6. CycleGAN	32
3.3.7. StyleGAN	34
3.3.8. Resumen	35
4. Creación y manipulación de Deepfakes	39
4.1. Síntesis de toda la cara	39
4.2. Intercambio de identidad	41
4.3. Manipulación de atributos	45
4.4. Intercambio de expresiones	46
4.5. Otras manipulaciones faciales	48
II. Fundamentos matemáticos de las redes generativas adversarias (GANs)	49
5. Conceptos previos	51
5.1. Teoría de la medida	51
5.1.1. Concepto abstracto de medida y espacio de medida	51
5.1.2. Integral asociada a una medida	52
5.1.3. Medida elemental y medida de Jordan	56
5.1.4. Medida de Lebesgue	58
5.1.5. Teorema Radon Nikodyn	59
5.1.6. Medida de probabilidad	60
5.2. Probabilidad	61
5.2.1. Variable aleatoria y vector aleatorio	61
5.2.2. Variables aleatorias continuas	63
5.2.3. Cambio de variable multidimensional	65
5.2.4. Esperanza matemática	65
6. Introducción matemática para la formulación de las GANs	67
6.1. Comprensión del problema matemático	67
6.2. Enfoque básico de las redes generativas adversarias	68
6.3. Problema min-max	69
6.3.1. Ejemplo de problema min-max	70
6.4. Formulación min-max del enfoque básico de las redes generativas adversarias	71
6.4.1. Medidas de similitud entre dos distribuciones de probabilidad	72

6.4.2. Solución al problema minmax	76
6.5. Algoritmo de entrenamiento de las redes generativas adversarias	77
6.6. Formulación minmax para el marco general de las redes generativas adversarias	79
III. Experimentación y resultados	85
7. Experimentación y resultados	87
7.1. Metodología	87
7.1.1. Descripción	87
7.1.2. Datos	96
7.1.3. Configuración de las redes neuronales	97
7.1.4. Herramientas	103
7.2. Resultados	103
7.2.1. MNIST	104
7.2.2. CelebA	110
IV. Conclusiones y trabajo futuro	115
8. Conclusiones y Trabajo futuro	117
A. Otras aproximaciones de tipo generativo	119
A.1. Transferencia de estilo	119
A.1.1. Representación del contenido semántico	119
A.1.2. Representación del estilo	120
A.1.3. Algoritmo de transferencia de estilo	120
A.2. Autoencoders	121
A.3. Otros	122
A.3.1. Redes generativas estocásticas(GSN)	122
A.3.2. GPT-3	122
A.3.3. Dall-E	122
Bibliografía	123

Agradecimientos

A mi familia que han estado apoyándome en mi día a día y creyendo ciegamente en mi.

A mis tutores, gracias a ellos he podido desarrollar este trabajo en el que me han ayudado y guiado en todo momento.

A mis compañeros y amigos que siempre me han animado a perseguir mis sueños.

Y por último quiero mostrar mi gratitud a toda la Universidad de Granada que estos últimos seis años de mi vida han sido mi familia acogiendo a esta gaditana y dividiendo su corazón entre la bahía y la sal de Cádiz y el embrujo y la sierra de Granada.

Resumen

Las redes generativas adversarias, llamadas habitualmente GANs, son modelos de aprendizaje profundo especializados en la generación de contenidos sintéticos realistas, como imágenes, audios y vídeos, a partir de ejemplos reales. Este trabajo se centra en el estudio de los fundamentos matemáticos e informáticos de las GANs, así como en su aplicación para la generación de ultrafalsificaciones de imágenes (*deep fakes*).

Las GANs son un tipo de redes neuronales con dos partes diferenciadas, una red generadora y otra red discriminadora, que compiten en un juego de suma cero. La red generadora produce muestras, con la intención de engañar a la red discriminadora para que crea que son muestras reales, y la red discriminadora debe distinguir las muestras reales de las producidas por el generador. Ambas redes se entrenan de forma simultánea con el objetivo de construir un buen generador que sea capaz de producir ejemplos sintéticos indistinguibles de los reales. Según su creador Ian Goodfellow, “es un proceso donde cada una de las redes va mejorando y aprende de su oponente”.

El procesamiento en las GANs tiene diferencias relevantes respecto a las redes neuronales habituales. Inicialmente, disponemos de un conjunto de entrenamiento con ejemplos reales y de vectores de ruido aleatorio con la misma estructura. Para cada vector de ruido, la red generadora construye un ejemplo falso. A continuación, estos ejemplos falsos, junto a los ejemplos reales, constituyen la entrada de la red discriminadora, cuya misión es asignar a cada ejemplo la probabilidad de que pertenezca al conjunto de ejemplos de entrenamiento; esto es, de que sea verdadero. La formulación y la implementación de las GANs presentan características de interés, que se estudian en la parte de este trabajo más cercana al ámbito de la informática.

En cuanto al entrenamiento, las dos redes neuronales se ajustan al mismo tiempo aplicando gradiente descendente y retropropagación. La clave del entrenamiento de las GANs es determinar cuándo finalizar este entrenamiento. Puesto que se trata de un juego de suma cero, también conocido como problema minmax, el objetivo teórico es alcanzar un equilibrio de Nash, esto es, un punto donde ningún jugador mejore su situación aunque el entrenamiento continúe. No obstante, este equilibrio es muy difícil de alcanzar en la práctica y presenta un problema para las GANs. En la parte de matemática del trabajo, se analiza la formulación de este problema minmax y de sus implicaciones en la convergencia del entrenamiento de las GANs, así como las diferentes alternativas dependiendo de la formulación de la función de pérdida de las redes basada en teoría de la probabilidad.

Finalmente, se realizan diversos experimentos con varios tipos de GANs (arquitectura y función de pérdida) en dos problemas de procesamiento de imágenes: la generación de dígitos, basada en el conjunto de datos MNIST, y la ultrafalsificación de caras, con el conjunto de datos CelebA. Este segundo caso de uso es especialmente relevante en la actualidad, ya que las GANs se utilizan cada vez más para la manipulación de contenidos audiovisuales con el propósito de propagar desinformación. Por ello, previamente se revisa el estado del arte de las *deep fakes* con GANs en este contexto.

En conclusión, este trabajo proporciona una introducción formal a las GANs, describiendo los principios fundamentales matemáticos que subyacen a esta técnica y explicando su

Resumen

materialización en una implementación con herramientas de aprendizaje profundo. La experimentación demuestra que es posible generar contenidos falsos y potencialmente realistas

Palabras claves: GAN, deepfakes, aprendizaje profundo, redes neuronales, minmax, modelo generativo, probabilidad, equilibrio Nash

Summary

Generative Adversarial Networks, known usually as GANs, are a deep learning model specialized in the generation of realistic synthetic contents, such as images, audio or video, based on real examples. The following work focuses on the study of the mathematics and computer science fundamentals of GANs, as well as on their application for the generation of *deep fakes*.

GANs are a kind of neural networks consisting of two well differentiated parts, a generative and a discriminative network, which compete against each other in a zero-sum game. The generative network produces samples with the intention of deceiving the discriminative network so that it believes they are real samples. For its part, the discriminative network must distinguish between real and produced samples. Both networks are trained simultaneously with the objective of creating a good generator which is capable of producing synthetic examples indistinguishable from the real ones. In words of its creator, Ian Goodfellow, “it is a process where each network improves and learns from its adversary progressively”.

Processing in GANs has relevant differences with respect to conventional neural networks. At the beginning we have a training set with real examples as well as a set of vectors of random noise which have the same structure. For each vector of noise, the generative network produces a fake example. Next, those fake examples, together with the real examples, form the input for the discriminative network, whose responsibility is to assign to each example the probability that it belongs to the set of training examples ; that is, that it is real. The formulation and implementation of GANs have characteristics of interest, which will be studied in detail in the part of this work centered in the field of computer science.

With regard to training, both neural networks are trained at the same time by applying gradient descent and backpropagation. The key to the training of GANs is to determine when to stop it. Since it is a zero-sum game, also known as the minmax problem, the theoretical goal is to find a Nash equilibrium, that is, a point where neither player improves his situation even though training could continue. Nevertheless, in practice it is very difficult to reach that equilibrium, what suppose a problem for GANs. In the mathematical part of this work, the formulation of the minmax problem and its implications in the convergence of training will be analyzed, as well as the different alternatives depending on the formulation of the loss function based on probability theory.

Finally, several experiments are carried out with different types of GANs (architecture and loss function) on two image processing problems: the generation of digits, based on the MNIST dataset, and *deep fakes*, with the set of CelebA data. This second use case is particularly important nowadays, since GANs are being used increasingly for the manipulation of audio-visual content with the purpose of spreading misinformation. Therefore, the state-of-the-art of *deep fakes* with GANs in this context will be reviewed previously.

In conclusion, this work presents a formal introduction to GANs, describing the mathematical fundamentals that underlie this technique and explaining its materialization through an implementation with deep learning tools. Experimentation shows that it is possible to create fake and potentially realistic contents.

Summary

Computer Science fundamentals of GANs and its application for the creation of deep fakes

The first section of this work will cover the computer science related aspects needed to carry out this project. Theory about neural networks and convolutional neural networks (CNNs) will be explained, since our model always work with two neural networks: the generator and the discriminator. Once neural networks have been studied, the functioning, training and learning process of a generator or a discriminator can be easily imagined by the reader, but the way they work in conjunction in order to generate images from scratch is still unknown. It must be taken into account that the generative network needs the discriminative network to know how to create a so realistic synthetic image that the discriminator is not able to distinguish it from a real image. Therefore, the functioning, training and learning process of a GAN model will be studied later. Likewise, the most common types of GANs used today will be introduced. Moreover, the state-of-the-art of four types of facial manipulations will be reviewed in order to show the dangers that the ability to generate fake faces, both in images and videos, entails. Some examples are the manipulation for creating images of non-existent faces, known as *Entire Face Synthesis*, the manipulation that replaces in a video one person's face with another's person face, known as *Identity Swap*, manipulation that modifies or edits the face of a person in an image, known as *Attribute Manipulation*, and the manipulation that replaces one person's facial expression with another's in a video, known as *Expression Swap*.

Mathematical Fundamentals of GANs

In the second section some concepts of measure theory and probability theory will be introduced. Indeed, one needs to know the explained concepts from measure theory to understand probability theory. The mathematical basis of a GAN model includes probability distributions and probability measures. Let us imagine that we have a dataset X, consisting of samples generated from the same probability distribution, then our goal is to obtain a GAN model that generates similar data to that in X. In other words, we need a model that generates new data that is close to the original dataset. That similarity between the actual and the fake data is achieved by determining the probability distribution of the data generated by the GAN model and forcing that distribution to approximate or to be the same as the probability distribution of the data in X. A basic approach of GANs will be considered where the neural networks are modeled as functions and a minmax problem for the zero-sum game will be presented. The basic approach will study the minmax problem when the probability distributions are continuous and a general framework will be formulated that will try to solve the minmax problem in the case where the probability distributions are not continuous. Moreover, a generalization of the basic approach to a general framework, known mathematically as f-GAN, will be introduced. Additionally, the training algorithm of a GAN model will be studied in more detail.

Experimentation and results

The third section will show the images generated by the implementation of different proposed experiments for a DCGAN model and a WGAN-GP model. First of all, the MNIST database was used with a view to comparing both models and it could be clearly seen that

the WGAN-GP model obtained better results. In addition, different experiments were carried out for each model in order to familiarize ourselves with the learning process and to try to improve it if possible. Secondly, a test with the database CelebA and the WGAN-GP model was performed where the utility of being able to generate fake faces of celebrities was shown.

Keywords: GAN, deep fakes, deep learning, neural networks, minmax problem, generative model, probability, Nash equilibrium, probability distribution, CNN, probability measure, zero-sum game, f -GAN, CelebA, MNIST, DCGAN, WGAN-GP

Introducción

Este capítulo trata de contextualizar, describir y estructurar todo el trabajo realizado, así como marcar los objetivos alcanzados.

La contextualización se encargará de explicar la importancia de las deepfakes en nuestro día a día, ya que acarrean numerosas amenazas, junto con una breve descripción del algoritmo GANs que nos permite su creación. A continuación, se mostrará una descripción del trabajo junto con la estructura sintetizada en los distintos capítulos. Además, se citarán las principales fuentes consultadas y los objetivos alcanzados. A modo de cierre se realizará una planificación y estimación de costes del trabajo.

1. Contextualización

Generalmente, las deepfakes se refiere a vídeos o imágenes en las que la cara y/o la voz de una persona ha sido manipulada utilizando software de inteligencia artificial de forma que el vídeo alterado o la imagen alterada parezca auténtica. Estas falsificaciones son una fuente de preocupación a día de hoy.

Se basan en algoritmos donde tenemos un primer modelo que genera imágenes y un segundo modelo que adivina si las imágenes son reales o falsas. Estos algoritmos pueden ser codificadores automáticos o redes generativas adversarias(GAN), en nuestro estudio nos centraremos en las GANs.

Las GANs enfrentan dos modelos de inteligencia artificial uno contra el otro, el primero conocido como el generador, recibe un ruido aleatorio y construye una imagen falsa que se agregará con todas las imágenes reales para introducirse como entrada al segundo modelo, conocido como discriminador. Al principio esta imagen falsa no se parece a una cara pero al repetir el proceso indefinidas veces, se generarán imágenes de caras falsas indistinguibles de las caras reales. Por tanto, se habrán generado deepfakes.

Las deepfakes tuvo su desarrollo inicial a finales de la década de 1990 pero los avances más relevantes se realizaron a finales de la década de 2010.

Como se ha comentado anteriormente la creación de deepfakes puede crear una sociedad llena de amenazas donde las personas no puedan distinguir lo que es verdad de lo falso. Esta tecnología podría generar dudas sobre eventos falsos, por ejemplo en los tribunales de justicia, o plantear riesgos de seguridad personal ya que estas falsificaciones podrían crear datos biométricos y engañar a los sistemas de reconocimiento de rostro, voz, ...

Esta tecnología puede presionar a los periodistas que luchan por filtrar las noticias reales de las falsas, amenazan la seguridad nacional ya que puede difundir propaganda e interferir en las elecciones y plantean numerosos problemas de ciberseguridad.

Su mayor impacto recae en el acoso a mujeres al introducir caras falsas en videos pornográficos y en las dudas que aparecen en documentos políticos al pensar que puedan ser falsos. Al existir las deepfakes muchas veces no puedes creer lo que ves. En definitiva, presenta una serie de impactos negativos en cuanto al acoso, las falsificaciones en política y la posibilidad de sembrar la duda.

2. Descripción del trabajo

Nuestro trabajo se enfoca en estudiar la creación de deepfakes haciendo uso de las redes generativas adversarias(GANs).

En el primer bloque se estudiaron los fundamentos informáticos de las GANs y su utilidad para la creación de deepfakes. Inicialmente, se recuerdan los conceptos de redes neuronales y CNN que he aprendido a lo largo mi formación para luego adentrarnos en el estudio de las GANs. Aquí se estudiarán los fundamentos de las redes generativas adversarias: su nacimiento, modelo de computación, funcionamiento, algoritmos de entrenamiento e implementación eficiente y tipología de GANs. Luego, se analizará la literatura sobre generación de contenido sintético realista en imágenes y/o videos con este tipo de redes.

En el segundo bloque se trataran los fundamentos matemáticos de las GANs junto un recordatorio de conceptos previos estudiados a lo largo de mi formación en la carrera. Primero se estudiará los conceptos necesarios acerca de teoría de la medida, en particular para medidas finitas (probabilidad) para describir de forma rigurosa los fundamentos matemáticos de las redes generativas adversarias. A continuación, se estudiará bajo qué condiciones el problema minmax asociado a dichas redes tiene solución y que el método iterativo propuesto para su resolución en base a un conjunto de muestra inicial converge a la medida buscada. Finalmente, se verá que estando en un marco general, sin imponer condiciones, nuestro problema minmax asociado a dichas redes tiene solución.

En el tercer bloque se desarrollan distintos ejemplos ilustrativos haciendo uso de la base de datos CelebA y MNIST. Concretamente, se implementarán dos tipos de GANs, DCGAN y WGAN-GP, y se probaran distintos experimentos con objeto de generar contenido falso que evidencie el alto potencial de los modelos GANs.

Finalmente, se presentará un cuarto bloque con las conclusiones obtenidas y con un análisis de los posibles trabajos futuros. Asimismo, se puede consultar un apéndice con información complementaria sobre otros modelos generativos.

3. Estructura del trabajo

Anteriormente se ha comentado un poco la organización de nuestro trabajo, se puede apreciar que disponemos de cuatro bloques y cada uno de ellos se estructurará en capítulos.

1. El primer bloque se estructura en los capítulos 1, 2, 3 y 4. El capítulo 1 recuerda los fundamentos y el entrenamiento de las redes neuronales, así como las mejoras que se les puede introducir en su aprendizaje y los teoremas de aproximación universal. Los capítulos 2 y 3 persiguen el mismo esquema que el capítulo 1, esto es, presentan una red distinta donde se explican sus fundamentos, su entrenamiento y las mejoras que pueden realizarse en su aprendizaje. Concretamente, el capítulo 2 nos presenta las CNN y el capítulo 3 las GANs. Este bloque enfocará toda su atención en los capítulos 3 y 4, en el capítulo 3 podemos observar como se detallan claramente los fundamentos y el entrenamiento de una GANs, y finalmente se estudiarán distintos tipos de GANs. Por otro lado, el capítulo 4 realizará la síntesis de las distintas técnicas que existen de manipulaciones faciales y creación de deepfakes.

2. El segundo bloque se divide en dos capítulos. El capítulo 5 expone conceptos de teoría de la medida y de probabilidad que nos serán de gran utilidad para la comprensión de todo los fundamentos matemáticos. Estos conceptos nos permiten realizar adecuadamente todos los razonamientos expuestos en el capítulo 6 donde presentaremos la formulación matemática de las GANs, su enfoque minmax, demostraremos la existencia de solución para dicho problema y la convergencia del algoritmo de entrenamiento de una GANs.
3. El tercer bloque se divide en un capítulo con dos secciones. En la primera sección del capítulo 7 se muestra todos los experimentos realizados junto con explicaciones de la implementación realizada para cada tipo de GANs, las bases de datos usadas, las configuraciones de las redes neuronales del generador y discriminador y también se comentarán las herramientas usadas en nuestro código. La segunda sección nos ilustra los resultados obtenidos con los experimentos propuestos.
4. El cuarto bloque contempla el capítulo 8 donde se tratará de explicar las conclusiones que se ha obtenido después del desarrollo de este trabajo junto con algunas aspiraciones futuras.

4. **Bibliografía fundamental**

Señalamos la bibliografía que nos ha servido como base para desarrollar todo nuestro trabajo.

- GANs In Action, Vladimir Bok.
- Deep Learning, Ian Goodfellow.
- Deep Learning with Python, Francois Chollet.
- DeepFakes and Beyond: A Survey of Face Manipulation and Fake Detection. Ruben Tolosana, Ruben Vera-Rodriguez, Julian Fierrez, Aythami Morales, and Javier Ortega-Garcia, 2020.
- An introduction to measure theory, Terence Tao.
- A mathematical introduction to generative adversarial nets (GAN), 2020, Yang Wang.
- Generative adversarial nets, Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio.

5. **Objetivos del trabajo**

Nuestro objetivo es comprender las redes generativas adversarias y ver su aplicación en la generación de contenidos multimedia sintéticos. Para ello, se describirán los fundamentos matemáticos en que se basa dicha técnica de aprendizaje automático y se demostrará su validez. A modo de cierre se desarrollarán programas que ilustren la capacidad de esta técnica de crear deepfakes.

1. Informática: El objetivo principal es describir los fundamentos informáticos y realizar una experimentación donde se pruebe a crear deepfakes haciendo uso del modelo GAN.

- a) Recordar toda la información necesaria sobre las redes neuronales y las CNN.
 - b) Estudiar los fundamentos de las GANs donde incluiremos su nacimiento, a que modelos pertenecen, como funcionan, algunas metáforas que clarifican su gran potencial y por qué se estudian estos modelos a día de hoy. A continuación, se pretende explicar el entrenamiento de las GANs, detallando las funciones de costes y las dificultades que se presentan en su aprendizaje. Finalmente, se mostrarán los distintos tipos de GANs.
 - c) Realizar una síntesis de las distintas técnicas de manipulación de imágenes faciales o vídeos. En cada manipulación se estudiará su técnica, las bases de datos usadas y su estado de arte en técnicas de detección a día de hoy.
 - d) Llevar a cabo distintos experimentos con la base de datos CelebA y MNIST. Implementar distintos modelos GANs y ver cómo son capaces de generar contenido sintético a partir de las bases de datos propuestas.
2. Matemáticas: El objetivo principal es describir los fundamentos matemáticos de las redes generativas adversarias(GANs).
- a) Estudiar los conceptos de *teoría de la medida*: ¿qué es una medida? ¿cuándo dos medidas son similares? ¿qué es una medida absolutamente continua?, medidas finitas, etc. y *teoría de probabilidad*: distribuciones de probabilidad, densidad de probabilidad, esperanza de una variable aleatoria, etc.
 - b) Nuestra GAN generará de forma artificial objetos similares a los existentes en una colección de objetos con cierto grado de consistencia. Por tanto, necesitamos precisar qué entendemos por colección de objetos con cierto grado de consistencia y definir qué queremos decir con objetos similares. Así matematizaremos nuestro objetivo en términos de distribuciones de probabilidad y realizamos una comprensión del problema matemático planteado.
 - c) Mostrar el enfoque básico de las GANs donde el discriminador y el generador se modelan como funciones G y D , ya que aplicando el Teorema de Aproximación Universal tenemos que cualquier red neuronal se puede aproximar por funciones.
 - d) Presentar el problema minmax: puesto que la estrategia desarrollada en GAN consiste en plantear un problema de minmax será necesario desarrollar este tipo de problemas.
 - e) Realizar una formulación minmax del enfoque básico GAN para ver bajo qué condiciones tienen solución, qué propiedades verifica la solución, utilidades y aplicaciones. Además, habrá que demostrar por qué el procedimiento de minmax converge teóricamente a la solución buscada. También, se estudiarán las medidas de similitud entre dos distribuciones de probabilidad ya que nos será de utilidad en determinados razonamientos.
 - f) Convergencia del algoritmo de entrenamiento de una GAN básica.
 - g) Estudiar la formulación minmax para el marco general donde el enfoque GAN admite siempre solución sin tener que imponer condiciones sobre las distribuciones de probabilidad. Se demostrará cómo generalizar nuestro enfoque básico a un marco general conocido como f -GAN.

Veremos como todos los objetivos se han quedado cumplidos en su totalidad y en la siguiente tabla se presentan las asignaturas que mayor influencia han tenido en nuestro trabajo.

Matemáticas	Informática
Estadística Descriptiva e Introducción a la probabilidad Topología I Análisis matemático I Análisis matemático II Probabilidad	Inteligencia Artificial Aprendizaje Automático Visión por Computador

6. Planificación y estimación de costes

Se van a enumerar por orden las tareas que conforman la planificación de este proyecto junto con el tiempo que se le dedicará a cada tarea.

La planificación del proyecto seguirá las siguientes etapas.

1. **Análisis y organización del problema:** esta etapa se encargará de analizar el contenido del trabajo, así como de organizar los distintos capítulos. Esta tarea se desarrollará durante 8 días a finales de Enero, ya que se necesitará tener todos los contenidos bien estructurados para continuar posteriormente con su desarrollo.
2. **Estudio e investigación del problema:** esta etapa se corresponderá con el estudio a fondo de toda la bibliografía junto con el repaso de las asignaturas necesarias para el desarrollo de este trabajo. Primero, se comenzará con la investigación del contenido informático y una vez se familiarice con las GANs se procederá a investigar de forma paralela el contenido matemático, para ello se necesitarán los 28 días del mes de Febrero.
3. **Experimentación:** se implementarán experimentos en programas donde se evidencie la capacidad de una GAN para generar contenido sintético. Esta tarea se iniciará justo después del estudio e investigación del problema ya que, justo en ese momento, dispondrá de conocimientos para afrontar la experimentación. Esta tarea se desarrollará durante los meses de Marzo, Abril y Mayo, concretamente en 21 días, al mismo tiempo se redactará la memoria.
4. **Redacción de la memoria:** en esta etapa se elaborará todo el documento. Necesitará aproximadamente 60 días, distribuidos entre los meses de Marzo, Abril y Mayo donde se podrá exponer todos los conocimientos estudiados en mi investigación.
5. **Revisiones finales:** en esta última etapa se realizarán cambios y correcciones a lo largo de 15 días con objeto de concluir este documento final. Su desarrollo tendrá lugar a finales de Mayo y principios de Junio.

Tarea	Días	Meses
Análisis y organización del problema	8	Enero
Estudio e investigación del problema	28	Febrero
Experimentación	21	Marzo, Abril y Mayo
Redacción de la memoria	60	Marzo, Abril y Mayo
Revisiones finales	15	Mayo y Junio

Introducción

A continuación, se calculará la estimación de costes basadas en las horas dedicadas al trabajo. Concretamente, se va a calcular el sueldo bruto de un trabajador. De media se planea trabajar unas 40 horas semanales desde el inicio del desarrollo de este trabajo a finales de Enero. Haciendo una estimación de costes en función de las horas podemos elaborar el coste que se le pagaría a un trabajador por el desarrollo de este proyecto.

Para estimar los costes debemos de considerar que el salario de un ingeniero informático promedio en España es de 26.450€ al año o 13,56€ por hora. Por tanto, si se va a dedicar 40 horas semanales a lo largo de cinco meses, aproximadamente, entonces el gasto total estimado para este proyecto debe ser de 10848€. Si consideramos el coste del Colab+(aportado por el proyecto IBERIFIER (www.iberifier.eu)) que se ha usado para ejecutar nuestra experimentación, en tal caso debemos de añadirle 50€ y finalmente, el gasto total sería 10898€.

Parte I.

**Fundamentos informáticos de las redes
generativas adversarias (GANs) y aplicación en la
creación de *deep fakes***

1. Redes neuronales

Este capítulo describirá diversos conceptos de aprendizaje automático que nos serán de gran utilidad para los próximos capítulos. En primer lugar, introduciremos el concepto de red neuronal junto con un ejemplo simple. A continuación, abordaremos los fundamentos del aprendizaje basado en gradientes y su materialización en los algoritmos de retropropagación y descenso del gradiente. Luego comentaremos algunas ideas claves a tener en cuenta para la optimización de redes neuronales y concluiremos exponiendo los teoremas de aproximación universal. [Goo16]

1.1. Fundamentos

Las redes neurales de propagación hacia adelante, también llamadas redes neuronales *feedforward*, son los modelos de aprendizaje profundo por excelencia. Su objetivo es aproximar una función f^* ; por ejemplo, en un clasificador, la función $y = f^*(x)$, que asigna a una entrada x una categoría y . Una red neuronal define una función $f(x; \theta)$ y, aplicando un proceso de optimización, es capaz de *aprender* el valor de los parámetros θ que consiguen la mejor aproximación de la función f^* .

- Se denominan *feedforward* porque la información fluye hacia adelante: a partir de una entrada x , se realizan diversos cálculos intermedios f dependientes de θ y, finalmente, se obtiene la salida.
- Se consideran *redes* porque f se obtiene como la composición de varias funciones. Por ejemplo, podríamos tener tres funciones $f^{(1)}, f^{(2)}, f^{(3)}$ conectadas en cadena para formar $f(x) = f^{(3)}(f^{(2)}(f^{(1)}(x)))$. En este caso, $f^{(1)}$ es la primera capa de la red, $f^{(2)}$ la segunda capa, y así sucesivamente. A este tipo de arquitectura se la denomina perceptrón-perceptrones multicapa (*multi-layer perceptron*, MLP). La longitud total de la cadena determina la profundidad del modelo, de ahí el calificativo de aprendizaje *profundo* cuando la red tiene una gran cantidad de capas. Las capas intermedias se denominan capas ocultas, mientras que la capa final se llama capa de salida.
- Se llaman *neuronales* porque están inspiradas en la neurociencia. Cada unidad se asemeja a una neurona en el sentido de que recibe información de muchas otras unidades y calcula su valor escalar de salida ponderando estas entradas y modulando el resultado con una activación. Formalmente, cada neurona se representa con el vector de sus pesos de entrada y la función que usa como activación. El proceso de aprendizaje consiste en ajustar estos pesos para acercarse a la salida deseada.

1.1.1. La neurona

La neurona es la unidad básica de procesamiento de una red neuronal. Una neurona recibe una serie de valores de entrada provenientes de la capa anterior y los usa para operar con sus parámetros internos con objeto de devolver un único valor de salida. Más concretamente,

1. Redes neuronales

la neurona se encargará de realizar una combinación lineal de las entradas con los pesos sumándole un valor de sesgo b , y a continuación, aplicará una transformación no lineal de este resultado.

En adelante, utilizaremos la siguiente notación para una neurona: x el vector de valores de entrada, N es el número de entradas de la neurona, w es el vector de pesos y b es el valor del sesgo. Así, el cálculo dentro de la neurona es el siguiente:

$$z = w^T x + b = \sum_{j=1}^N w_j \cdot x_j + b$$

Adicionalmente, la neurona aplica una función de activación no lineal sobre la salida anterior. Podemos darnos cuenta de que por mucho que se intenten enlazar neuronas una detrás de otra, no se obtendría una mayor ventaja que con el uso de una sola neurona. Esto ocurre porque la composición de funciones lineales es una función lineal. Para darle utilidad a la idea de enlazar neuronas y para garantizar que se pueda obtener un comportamiento global no lineal de la red, surge el concepto de función de activación. Las funciones de activación son funciones no lineales h que toman como entrada el valor z y devuelven $h(z)$ como salida final $f^{(i)}$ de la neurona i . En la sección 1.1.3 comentaremos diversas funciones de activación.

$$f^{(i)} = h(z)$$

1.1.2. Arquitectura de la red

El número de capas ocultas, el número de unidades por capa y la forma en que se conectan las neuronas definen la arquitectura de una red. Una vez que nosotros fijemos la arquitectura, tendremos que calcular los parámetros w mediante algoritmos de aprendizaje.

La arquitectura más simple es la del perceptrón multicapa, o MLP, que se muestra en la Figura 1.1. En esta arquitectura, primero se define una capa de entrada con n neuronas, donde la neurona i introduce el valor x_i correspondiente, siendo $x = [x_1, x_2, \dots, x_n]$ la entrada de la red.

Estas entradas serán enviadas a cada neurona de la primera capa oculta de la red. Desde ese momento estas neuronas se encargarán de calcular una transformación afín $z = w^T x + b$ y luego le aplican a z la función no lineal, función de activación, para obtener la salida de cada neurona i , $f^{(i)} = h(z)$. Este valor emitido por la neurona i será enviado a todas las neuronas de la capa siguiente. Este mecanismo lo realizan todas las neuronas de las capas ocultas. Finalmente, cuando llegamos a la capa de salida nos encontramos con tantas neuronas como salidas tendrá la red.

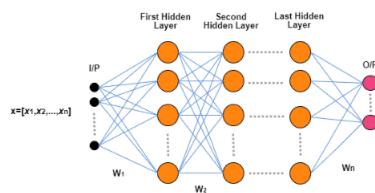


Figura 1.1.: Perceptrón multicapa [Kim17]

En capítulos siguientes veremos como funcionan distintas arquitecturas especializadas como son las CNN y las GANs.

1.1.3. Funciones de activación

Veamos las funciones de activación más relevantes:

- **Unidad lineal rectificada(ReLU)**

Su funcionamiento consiste en no realizar ninguna operación para las entradas positivas o nulas y devolver cero cuando la entrada es negativa.

$$h(x) = \begin{cases} x & \text{si } x \geq 0 \\ 0 & \text{si } x < 0 \end{cases}$$

- **Tangente hiperbólica**

Consiste en una función acotada entre -1 y 1 .

$$h(x) = \tanh(x)$$

- **Sigmoide o logística**

Consiste en una función acotada entre 0 y 1 , comúnmente usada en las capas de salida de las redes de clasificación binarias.

$$h(x) = \frac{1}{1+\exp^{-x}}$$

- **Softmax**

Consiste en una función acotada entre 0 y 1 , comúnmente usada en las capas de salida de las redes de clasificación donde tenemos n clases.

$$h(x_i) = \frac{e^x}{\sum_{i=1}^n e^{x_i}} \quad \forall i \in \{1, \dots, n\}$$

1.1.4. Ejemplo de aprendizaje: XOR

Las redes neuronales son una generalización del perceptrón que usa una transformación de características que aprende de los datos. El perceptrón no puede implementar una clasificación simple de funciones, como puede ser un XOR.

Para hacer más concreta la idea de una red neuronal, comenzaremos con un ejemplo de una red que aprende la función $f=\text{XOR}$. Para ello, descomponemos f en dos perceptrones simples y luego combinamos la salida de estos dos perceptrones, h_1, h_2 , de una manera simple: $f = h_1 \bar{h}_2 + h_2 \bar{h}_1$

Además, se puede probar que un etiquetado complicado, el cual esta compuesto por perceptrones, es una disjunción(OR) de conjunciones(AND) aplicado a los perceptrones componentes. A continuación, veamos como implementar esta idea.

$$\begin{aligned} OR(x_1, x_2) &= \text{sign}((+1)x_1 + (+1)x_2 + (+1.5)) \\ AND(x_1, x_2) &= \text{sign}((+1)x_1 + (+1)x_2 + (-1.5)) \end{aligned}$$

1. Redes neuronales

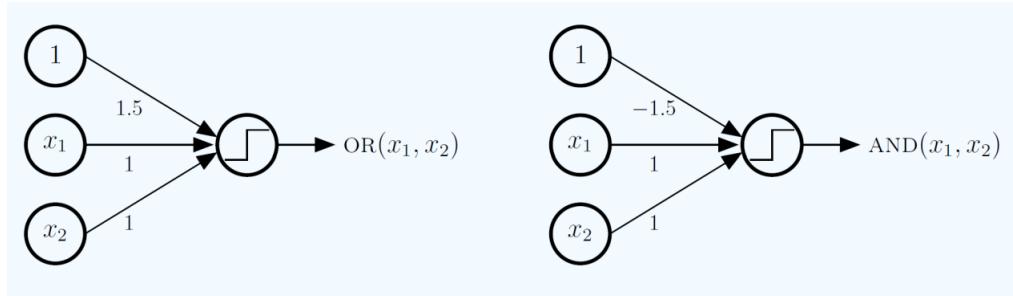


Figura 1.2.: OR/AND [CS]

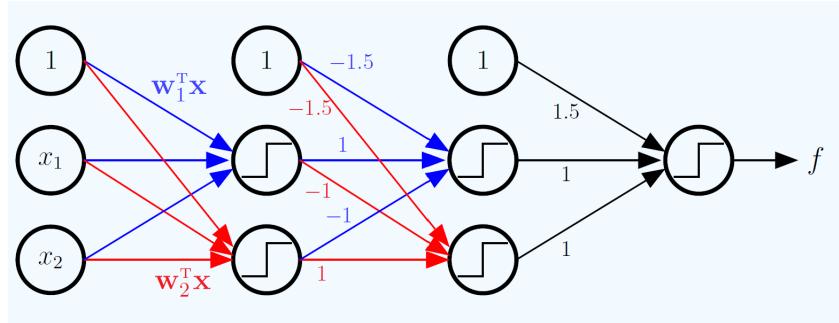


Figura 1.3.: XOR [CS]

Asumiendo el punto (x_1, x_2) y los vector de pesos (w_1, w_2, w_3) y (v_1, v_2, v_3) tenemos:

- Capa 1:

$$h_1(x_1, x_2) = \text{sign}(w_0 + x_1 w_1 + x_2 w_2)$$

$$h_2(x_1, x_2) = \text{sign}(v_0 + x_1 v_1 + x_2 v_2)$$
- Capa 2:

$$h_3(x_1, x_2) = \text{sign}(-1.5 + (-1) \cdot h_1(x_1, x_2) + (+1) \cdot h_2(x_1, x_2))$$

$$h_4(x_1, x_2) = \text{sign}(-1.5 + (+1) \cdot h_1(x_1, x_2) + (-1) \cdot h_2(x_1, x_2))$$
- Capa salida:

$$f := \text{sign}(1.5 + (+1) \cdot h_3(x_1, x_2) + (+1) \cdot h_4(x_1, x_2))$$

Acabamos de ver que existe una red neuronal capaz de implementar la función XOR.

1.2. Entrenamiento de redes neuronales

La mayor diferencia entre los modelos lineales y las redes neuronales es que la no linealidad de una red neuronal hace que las funciones de pérdida más interesantes sean no convexas. Esto significa que las redes neuronales generalmente se entrena mediante el uso de optimizadores iterativos basados en gradientes que tratan de disminuir la función de costo a un valor pequeño. La optimización convexa converge a partir de cualquier parámetro inicial mientras que el descenso de gradiente estocástico aplicado a funciones de pérdida no convexas no tiene tal garantía de convergencia y es sensible a los valores de los parámetros.

iniciales. Para las redes neuronales es importante inicializar todos los pesos a pequeños valores aleatorios, es bastante común usar una Gaussiana($0, \sigma^2_\epsilon$) que cumpla $\sigma^2_\epsilon \max_{x_i} \|x_i\|^2 \ll 1$.

Para calcular el gradiente veremos el algoritmo de retropropagación y para entrenar nuestra red usaremos algoritmos que se basan en usar el gradiente para descender la función de costo de una forma u otra. Estos algoritmos son mejoras del algoritmo de descenso de gradiente estocástico. Ahora nos centraremos en escoger la función de costo que usaremos en el algoritmo retropropagación. También cabe resaltar que la elección de la función de costo está estrechamente relacionada con la elección de la función de activación. La mayoría de veces usamos la entropía cruzada entre la distribución de los datos y la distribución del modelo. La elección de cómo representar la salida determina la forma de la función de entropía cruzada.

1.2.1. Función de costo

Un aspecto importante del diseño de una red neuronal es la elección de la función de costo. En la mayoría de los casos, nuestro modelo paramétrico define una distribución $p(y|x; \theta)$ y utilizamos el principio de máxima verosimilitud. Esto significa que usamos como función de costo la entropía cruzada entre los datos de entrenamiento y las predicciones del modelo. La función de costo utilizada finalmente para entrenar la red neuronal a menudo combinará la función que presentaremos aquí con algún término de regularización. Una de las estrategias de regularización más populares es el enfoque de weight decay ya que ha sido utilizado en modelos lineales y es directamente aplicable a redes neuronales profundas.

La mayoría de las redes neuronales modernas se entranan usando la máxima verosimilitud, esto significa que la función de costo es simplemente la verosimilitud logarítmica negativa, descrita como la entropía cruzada entre la distribución de los datos de entrenamiento(p_{data}) y la distribución de modelo (p_{model}).

$$E(\theta) = -\mathbb{E}_{x,y \sim p_{data}} \log p_{model}(y|x)$$

Si $p_{model}(y|x) = N(y; f(x; \theta), I)$ entonces se obtiene el costo del error cuadrático medio :

$$E(\theta) = -\mathbb{E}_{x,y \sim p_{data}} \|y - f(x : \theta)\|^2 + const$$

1.2.2. Retropropagación

Para desarrollar el algoritmo vamos a usar la siguiente notación:

- Las **capas** están etiquetadas por $l = 0, 1, 2, \dots, L$, donde $l = 0$ es la capa de entrada, $l = L$ es la capa de salida y las demás son las capas ocultas. Cada capa tiene **dimensión** $d^{(l)}$, esto es, tiene $d^{(l)}+1$ nodos
- **Señal de entrada** $s^{(l)}$, es el vector de entrada de dimensión $d^{(l)}$
- El **peso** asociado al nodo j de la capa l que proviene del nodo i de la capa anterior lo denotamos como $w_{ij}^{(l)}$. Si le aplicamos los pesos a la entrada se obtiene la señal que va a ese nodo, Por ejemplo, la entrada al nodo j de la capa l se obtendría

$$s_j^l = \sum_{i=0}^{d^{l-1}} w_{ij}^l x_i^{l-1}$$

1. Redes neuronales

En definitiva, podemos denotar $s^{(l)}$ al **vector de entrada a la capa l**, llamando $W^{(l)}$ a la matriz donde en la fila j almacena los pesos necesarios para construir la señal de entrada del nodo j.

$$s^{(l)} = (W^{(l)})^T x^{(l-1)}$$

El **vector de salida de la capa l**, $x^{(l)}$, se construye a partir de la función de activación θ .

$$x^{(l)} = \begin{pmatrix} 1 \\ \theta(s^{(l)}) \end{pmatrix}$$

$$x = x^{(0)} \xrightarrow{W^{(1)}} s^{(1)} \xrightarrow{\theta} x^{(1)} \xrightarrow{W^{(2)}} s^{(2)} \xrightarrow{\theta} x^{(2)} \dots \dots \dots \xrightarrow{W^{(L)}} s^{(L)} \xrightarrow{\theta} x^{(l)} = h(x)$$

■ Algoritmo de propagación hacia delante

La siguiente imagen se ha extraido de los apuntes de asignaturas cursadas en el grado.

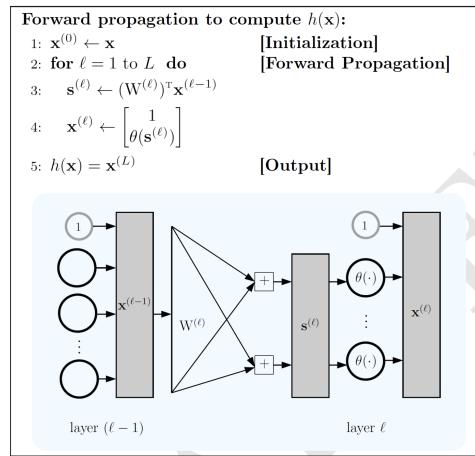


Figura 1.4.: Algoritmo

Retropropagación es un algoritmo especial que calcula el gradiente de forma eficiente. Nuestra idea es calcular el gradiente del error dentro de la muestra $\{(x_1, y_1), \dots, (x_N, y_N)\}$, usando la notación definida anteriormente tenemos

$$E_{in}(w) = \frac{1}{N} \sum_{n=1}^N e_n = \frac{1}{N} \sum_{n=1}^N e(h(x_n), y_n)$$

$$\frac{\partial E_{in}}{\partial W^{(l)}} = \frac{1}{N} \sum_{n=1}^N \frac{\partial e_n}{\partial W^{(l)}}$$

Retropropagación se basa en aplicar varias veces la regla de la cadena para calcular derivadas parciales en la capa l utilizando derivadas parciales en la capa $l+1$.

$$s^{(l)} = (W^{(l)})^T x^{(l-1)} \rightarrow \frac{\partial e}{\partial W^{(l)}} = \frac{\partial e}{\partial s^{(l)}} \frac{\partial s^{(l)}}{\partial W^{(l)}}$$

Denotamos la sensibilidad de el error respecto a la señal de entrada en la capa l como $\delta^{(l)} = \frac{\partial e}{\partial s^{(l)}}$. La idea es propagar el error hacia atrás, de la última capa a la penúltima capa y así sucesivamente hasta llegar a la primera capa. Por tanto, tenemos la siguiente recurrencia $\delta^{(1)} \leftarrow \delta^{(2)} \leftarrow \dots \leftarrow \delta^{(L)}$, que nos servirá para ir calculando:

$$\frac{\partial e}{\partial W^{(l)}} = \frac{\partial e}{\partial s^{(l)}} \frac{\partial s^{(l)}}{\partial W^{(l)}} = x^{(l-1)} [\delta^{(l)}]^T$$

A continuación, veamos como calcular $\delta^{(l)}$ (\odot denota la multiplicación elemento a elemento)

$$\begin{aligned} \frac{\partial e}{\partial s^{(l)}} &= \frac{\partial e}{\partial x^{(l)}} \frac{\partial x^{(l)}}{\partial s^{(l)}} = \frac{\partial e}{\partial s^{(l+1)}} \frac{\partial s^{(l+1)}}{\partial x^{(l)}} \frac{\partial x^{(l)}}{\partial s^{(l)}} = \delta^{(l+1)} W^{(l+1)} \frac{\partial x^{(l)}}{\partial s^{(l)}} \longrightarrow \delta^{(l)} = \frac{\partial e}{\partial s^{(l)}} = \\ &\theta'(s^{(l)}) \odot [W^{(l+1)} \delta^{(l+1)}]_1^{d(l)} \end{aligned}$$

Para arrancar la recurrencia necesitamos conocer el valor $\delta^{(L)}$:

$$\delta^{(L)} = \frac{\partial e}{\partial s^{(L)}} = \frac{\partial (x^L - y)^2}{\partial s^L} = 2(x^L - y) \frac{\partial x^L}{\partial s^L} = 2(x^L - y) \theta'(s^{(L)})$$

Una vez conozcamos todos los valores de $\delta^{(l)} \forall l \in \{1, 2, \dots, L\}$ conoceremos las derivadas $\frac{\partial e_n}{\partial W^{(l)}} \forall l \in \{1, 2, \dots, L\}$ y por tanto el valor del gradiente que buscamos, $\frac{\partial E_{in}}{\partial W^{(l)}} \forall l \in \{1, 2, \dots, L\}$.

1.2.3. Descenso del gradiente

El entrenamiento de una red neuronal empieza con la inicialización aleatoria de los parámetros de la red, posteriormente se seguirá un proceso iterativo del cálculo del gradiente y actualización de los parámetros. El descenso del gradiente definirá como se usará el gradiente para actualizar los parámetros de la red. A continuación, podemos apreciar el proceso de actualización donde μ se conoce como tasa de aprendizaje y está relacionado con la velocidad de convergencia hacia el mínimo de la función de coste.

$$w(t+1) = w(t) + \mu \nabla E_{in}(w(t))$$

1.2.3.1. Descenso del gradiente estocástico (SGD)

Es un enfoque muy eficiente del descenso del gradiente ya que no necesita recorrer todos los datos de entrenamiento, solo necesita trabajar con una muestra del conjunto de entrenamiento. Por lo tanto, el método de descenso de gradiente estocástico puede actualizar los parámetros del modelo con un solo dato de entrenamiento, lo que acelera enormemente el entrenamiento.

1.3. Mejoras del aprendizaje

De todos los problemas de optimización involucrados en el aprendizaje profundo, el más difícil es el entrenamiento de redes neuronales. Debido a esto se han desarrollado distintas técnicas de optimización donde se pretende encontrar los parámetros θ que optimizan la función de costo $E(\theta)$ así como determinar los términos de regularización adicionales.

- Una diferencia muy importante entre la optimización general y la optimización como la usamos para entrenar algoritmos es que los algoritmos de entrenamiento no suelen detenerse en un mínimo local. Se detienen cuando se cumple un criterio de convergencia basado en **EarlyStopping**. Esta técnica está diseñada para hacer que el algoritmo se detenga cada vez que comience a producirse sobreajuste.
- Otra diferencia relevante es que la función objetivo generalmente se descompone como una suma sobre los ejemplos de entrenamiento. Por tanto, podemos calcular esta función tomando muestras al azar de una pequeña cantidad de conjunto de datos y luego calculando la media. Así reducimos el coste de evaluar la función en todos los datos de entrenamiento. De aquí surge la idea de los algoritmos que trabajan con minibatch en lugar de con todo el batch completo, como por ejemplo **SGD(Descenso de gradiente estocástico)**
- El descenso de gradiente estocástico y sus variantes son los algoritmos de optimización más utilizados para el aprendizaje automático y para el aprendizaje profundo. El parámetro crucial es la **tasa de aprendizaje** que antes comentamos pero ahora en lugar de ser una tasa fija, se irá disminuyendo con el tiempo.
- A veces el aprendizaje haciendo uso de SGD puede ser lento y en estos casos se recomienda utilizar la técnica **Momentum**. Este método está diseñado para acelerar el aprendizaje, especialmente frente a curvaturas altas, gradientes pequeños pero constantes o gradientes ruidosos. Se encarga de introducir un nuevo parámetro que desempeña el papel de velocidad y almacena la velocidad a la que se mueven los parámetros en el espacio de parámetros.
- La técnica **Dropout** es una técnica de regularización donde eliminamos un porcentaje de neuronas de cada capa de la red neuronal. Esta eliminación se realiza de forma aleatoria, consiguiendo que las neuronas no se especialicen en ninguna tarea muy concreta.

1.4. Teoremas de Aproximación universal

Para finalizar este capítulo nos preguntamos si pueden las redes neuronales comportarse como cualquier función matemática([[Cyb89](#)]). La respuesta es que si aunque sería más correcto decir que se pueden aproximar.

Si tenemos una función $f(x)$ que es aproximada por una red neuronal $f^*(x)$, siempre podemos encontrar una red que cumpla $|f(x) - f^*(x)| < \epsilon$ para cualquier ϵ que fijemos.

Hay que detallar que las funciones discontinuas con saltos no podrán ser aproximadas por redes neuronales ya que las funciones calculadas por una red son continuas.

Por otra parte, también tenemos que aclarar que las redes neuronales no son el único aproximador universal que existe, de hecho uno de los más usados son las funciones polinómicas.

Teorema 1.1. *Una red feedforward con una sola capa es suficiente para representar cualquier función, pero la capa puede ser inviablemente grande y puede fallar al aprender y generalizar correctamente.*

2. Redes convolucionales (CNNs)

Este capítulo tratará de recordar el funcionamiento básico de las redes convolucionales y la relevancia que han tenido en el aprendizaje profundo.

2.1. Fundamentos

Las redes convolucionales (LeCun, 1989) , también conocidas como redes neuronales convolutivas o simplemente CNNs, son un tipo de red neuronal diseñada para procesar datos que tienen una topología similar a una cuadrícula conocida. Su nombre indica que la red emplea una operación matemática lineal llamada convolución. Las redes convolucionales son simplemente redes neuronales que utilizan la convolución en lugar de la multiplicación general de matrices en al menos una de sus capas.

2.1.1. CNN y la historia del Deep Learning

Las redes convolucionales han jugado un papel importante en la historia del aprendizaje profundo. Fueron algunos de los primeros modelos profundos en funcionar bien, mucho antes de que los modelos profundos arbitrarios se consideraran viables. Las redes convolucionales también fueron algunas de las primeras redes neuronales en resolver importantes aplicaciones comerciales y permanecen a la vanguardia de las aplicaciones comerciales de aprendizaje profundo en la actualidad. Por ejemplo, en la década de 1990, el grupo de investigación de redes neuronales de ATT desarrolló una red convolucional para leer cheques (LeCun et al., 1998b). A fines de la década de 1990, este sistema implementado por NEC estaba leyendo más del 10 por ciento de todos los cheques en los Estados Unidos. Posteriormente, Microsoft implementó varios sistemas de OCR y reconocimiento de escritura a mano basados en redes convolucionales (Simard et al., 2003).

Las redes convolucionales también se utilizaron para ganar muchos concursos. La intensidad actual del interés comercial en el aprendizaje profundo comenzó cuando Krizhevsky et al. (2012) ganó el desafío de reconocimiento de objetos de ImageNet, pero las redes convolucionales se había utilizado para ganar otros concursos de aprendizaje automático y visión por computadora con menos impacto años antes.

Asimismo, las redes convolucionales fueron algunas de las primeras redes profundas de trabajo entrenadas con retropropagación y se consideraron más eficientes computacionalmente que las redes totalmente conectadas.

2. Redes convolucionales (CNNs)

2.1.2. El operador de convolución

La operación llamada convolución se define de la siguiente forma y se identifica normalmente con un asterisco.

$$s(t) = \int x(a)w(t-a)da$$

$$s(t) = (x * w)(t)$$

El primer argumento x hace referencia a la entrada de la red, el segundo argumento w se corresponde con el kernel y la salida s se le conoce como mapa de características.

Si asumimos que x y w están definidos solo en números enteros t , podemos definir la convolución discreta:

$$s(t) = (x * w)(t) = \sum_{a=-\infty}^{\infty} x(a)w(t-a)$$

La entrada suele ser una matriz multidimensional de datos y el kernel una matriz multidimensional de parámetros que se adaptan mediante el algoritmo de aprendizaje. Estas matrices multidimensionales se conocen como tensores. Además, debemos de asumir que estas funciones son cero excepto en un conjunto finito de puntos, esto significa que podemos implementar la suma infinita como una suma finita.

Si usamos imágenes bidimensionales, tendremos kernels bidimensionales.

$$S(i, j) = (I * K)(i, j) = \sum_m \sum_n I(m, n)K(i - m, j - n)$$

Al invertir el núcleo se obtiene la propiedad commutativa de la convolución y la anterior expresión se puede escribir de forma equivalente

$$S(i, j) = (K * I)(i, j) = \sum_m \sum_n I(i - m, j - n)K(m, n)$$

Finalmente, aclaramos con un ejemplo donde se muestra la operación de convolución que básicamente consiste en ir paseando el kernel sobre la capa de entrada e ir calculando el operador por cada casilla.

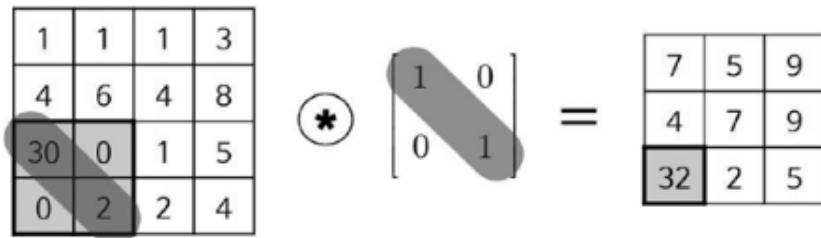


Figura 2.1.: Ejemplo del operador de convolución [Kim17]

2.1.3. Arquitectura de la red

Una CNN consta de dos partes, una primera parte conocida como **extracción de características** que se encarga de transformar la imagen de entrada y resumirla en un vector a partir de las transformaciones de tensores que se llevan a cabo en las distintas capas que conforman una CNN(Capas de convolución,pooling,activación....) y una segunda parte donde actua el **clasificador** para pasar del vector de características a la clase mediante conexiones densas (fully connected) y haciendo uso de SoftMax.

Veamos un resumen de las capas que pueden aparecer en un bloque convolucional.

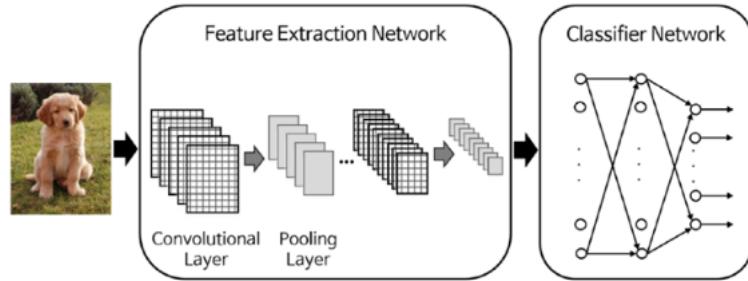


Figura 2.2.: CNN [Kim17]

1. Capa convolucional

Esta capa se encarga de realizar las transformaciones locales del tensor de entrada haciendo uso del operador de convolución que hemos descrito anteriormente. Vamos aplicando el kernel por el tensor de entrada y produciendo un tensor de salida. Si el tensor de entrada presenta unas dimensiones de $W_1 \times H_1 \times D_1$ y tenemos k filtros que aplicar, el tamaño del kernel es $F \times F$, se le aplica un stride S y zero-padding P , entonces las dimensiones del tensor de salida son $W_2 \times H_2 \times D_2$ donde $W_2 = \frac{W_1 - F + 2P}{S} + 1$, $H_2 = \frac{H_1 - F + 2P}{S} + 1$, $D_2 = K$.

Para estos valores se introducen $F \times F \times D_1$ pesos por filtro y por tanto para K filtros tenemos $(F \times F \times D_1) \times K$ pesos que aprender.

Stride me permite ir saltando convoluciones y zero-padding es un mecanismo para llenar con ceros las posiciones fuera del tensor.

2. Capa Activación

Esta capa se encarga de aplicar una función de activación por cada elemento, sin la capa de activación no se produce aprendizaje

3. Capa de reducción o Pooling

Se encarga de reducir el tamaño del tensor y existen distintos mecanismos como puede ser AveragePooling o MaxPooling. Cabe resaltar que MaxPooling a pesar de proporcionar invariancia espacial no es adecuado para modelos generativos.

4. Capa de UnSampling o Transposed

Esta capa solo se utiliza cuando sea necesaria porque las dimensiones del tensor sean

2. Redes convolucionales (CNNs)

demasiado pequeñas y se quiera realizar una interpolación para construir un tensor de mayor tamaño.

5. Capa Full Connected(FC)

Son las capas introducidas para operar con el vector de características.

2.2. Entrenamiento de CNN

Se realiza de forma análoga al entrenamiento de una red neuronal que podemos ver en la sección [1.2](#). Como las CNN son un tipo de red neuronal podemos aplicar los algoritmos de BackPropagation y descenso del gradiente citados en dicha sección.

2.3. Mejoras del aprendizaje

Vamos a introducir una serie de capas adicionales que son usada con el objetivo de mejorar el funcionamiento de una CNN.

Por una parte sería óptimo que todas las variables estuviesen en una misma escala, se pretende normalizar las imágenes, es decir, que con el batch de entrenamiento se obtenga una media de 0 y desviación típica de 1. Al ejecutar todas las capas anteriormente citadas vamos modificando las propiedades de los datos y por tanto vamos modificando la media y la desviación típica. De ahí surge la idea de porque no introducimos una capa que normalice los datos que van por la red. Así surgió la **capa de BatchNormalization**. Esta capa funciona con 4 parámetros por cada neurona y aprende para cada neurona del tensor el valor que debe restarle y dividirle para que la neurona obtenga una media de 0 y desviación de 1. Además de estos valores, también se aprende la escala y la fase. Hubo una gran discusión sobre donde colocar esta capa, los autores defendieron colocarla antes que la capa de activación pero luego se probó que después también funcionaba.

Al igual que sucedía con las redes neuronales la técnica de Dropout aplicada a las capas full connected aportó muy buenos resultados y se permitió el uso de la **Capa de Dropout** con objeto de optimizar el funcionamiento de la CNN ya que se encarga de la regularización.

Además de introducir estas capas, también se estudió otros mecanismos como aumentar la profundidad del modelo, el estudio de redes densas, el procesado de múltiples escalas de forma simultánea.

3. Redes Generativas Adversarias (GANs)

En este capítulo se expondrá una aproximación de un modelo generativo para la generación de contenido sintético con redes neuronales. Los modelos generativos presentan diversos enfoques y dominios de aplicación, en este trabajo nos centraremos en imágenes y en redes generativas adversarias, GAN.

3.1. Fundamentos

3.1.1. Nacimiento de las GANs

En 2014 un estudiante de doctorado en la universidad de Montreal llamado Ian Goodfellow inventó las redes generativas adversarias. Mas concretamente, el lunes 5 de diciembre de 2016 a las 14:30 horas, Ian Goodfellow de Google Brain presentó un tutorial titulado "Generative Adversarial Networks" a los delegados de la conferencia Neural Information Processing Systems(NIPS) en Barcelona. Esta técnica ha permitido a los ordenadores generar datos realistas usando no una, sino dos redes neuronales separadas. Las GANs han logrado resultados muy buenos e impensables, como la capacidad de generar imágenes falsas con una calidad que se asemeja a la del mundo real, convertir un garabato en una imagen similar a una fotografía o convertir imágenes de video de un caballo en una cebra corriendo, todo a partir de una pequeña cantidad de datos de entrenamiento etiquetados. Un ejemplo relevante que demostró la alta potencia de las GANs fue la síntesis de rostros humanos. En 2014 las máquinas que usaban las GANs solo eran capaces de mostrar un rostro borroso, e incluso eso se convirtió en un gran éxito. Tres años después, en 2017, los avances de las GAN permitieron construir rostros falsos cuya calidad era tan alta como las buenas fotografías de retratos.

3.1.2. ¿Qué son las GANs?

Las redes generativas adversarias o GAN son una aproximación de tipo generativo basado en redes generadoras diferenciables. Se basan en el planteamiento seguido en teoría de juegos donde la red generadora debe competir contra su adversario. La red generadora produce muestras $x = G(z; \theta^{(G)})$ y su adversario, la red discriminadora, intenta distinguir entre las muestras extraídas del conjunto de entrenamiento y muestras construidas por el generador. El discriminador se encarga de asignarle una probabilidad a la entrada x , $D(x; \theta^{(D)})$ indica la probabilidad de que x sea un ejemplo de entrenamiento real en lugar de una muestra falsa (los parámetros $\theta^{(G)}, \theta^{(D)}$ serán aprendidos en el entrenamiento de la GAN).

Por tanto, las GANs son una clase de técnicas de aprendizaje automático que consisten en dos modelos que se entranan de forma simultanea: la red del Generador entrenada para generar datos falsos y la red del Discriminador entrenada para distinguir las muestras falsas de muestras reales.

3. Redes Generativas Adversarias (GANs)

- El término GENERATIVO se refiere al objetivo central del modelo que es generar nuevos datos
- El término ADVERSARIO apunta a la competitividad que existe en este planteamiento de teoría de juegos donde compiten los dos modelos: el Generador y el Discriminador. El objetivo del Generador es crear nuevos datos que no se puedan distinguir de los datos reales del conjunto de entrenamiento. Las dos redes intentan siempre competir entre ellas: cuanto mejor se vuelve el Generador para crear nuevos datos similares a los datos del conjunto de entrenamiento, mejor debe ser el Discriminador para distinguir los datos reales de los falsos.
- El término REDES indica la clase de modelos de aprendizaje automático que más se usa para representar al Generador y al Discriminador, las redes neuronales.

3.1.3. Metáforas asociadas a las GANs

3.1.3.1. Ganimals

Esta metáfora ha sido extraída del libro [Fos19] y relata muy bien la competitividad y funcionamiento de las dos redes.

Una tarde, mientras camina por la jungla local, Gene ve a una mujer hojeando una serie de fotografías en blanco y negro, con cara de preocupación. Se acerca a preguntar si puede ayudar. La mujer se presenta como Di, una entusiasta exploradora, y explica que está buscando al escurridizo ganimal, una criatura mítica que se dice que vaga por la jungla.

Dado que la criatura es nocturna, solo tiene una colección de fotos nocturnas de la bestia que una vez encontró tirada en el suelo de la jungla, arrojadas por otro entusiasta de los ganimales. Di gana dinero vendiendo las imágenes a coleccionistas, pero está empezando a preocuparse, ya que en realidad nunca ha visto a las criaturas y le preocupa que su negocio se tambalee si no puede producir más fotografías originales pronto.

Gene decide ayudar a Di. Él acepta buscar él mismo al ganimal y darle las fotografías de la bestia nocturna que logra tomar. Dado que Gene nunca ha visto un ganimal, no sabe cómo producir buenas fotos de la criatura y, además, dado que Di solo ha vendido las fotos que encontró, ni siquiera puede notar la diferencia entre una buena foto de un ganimal y una foto de nada en absoluto.

A partir de este estado de ignorancia, ¿cómo pueden trabajar juntos para garantizar que Gene sea eventualmente capaz de producir impresionantes fotografías de ganimales?

Se les ocurre el siguiente proceso. Cada noche, Gene toma 64 fotografías, cada una en un lugar diferente con diferentes lecturas aleatorias de la luz de la luna, y las mezcla con 64 fotografías de animales de la colección original. Luego, Di mira este conjunto de fotos e intenta adivinar cuáles fueron tomadas por Gene y cuáles son originales. Basándose en sus errores, actualiza su propia comprensión de cómo discriminar entre los intentos de Gene y las fotos originales. Luego, Gene toma otras 64 fotos y se las muestra a Di. Di le da a cada foto una puntuación entre 0 y 1, indicando cómo realista que ella cree que es cada foto. Basándose en estos comentarios, Gene actualiza la configuración de su cámara para asegurarse de que la

próxima vez tome fotos que es más probable que Di califique alto.

Inicialmente, Gene no recibe ningún comentario útil de Di, después de algunas semanas le puede brindar mejores comentarios a Gene para que pueda ajustar su cámara. Esto hace que la tarea de Di sea más difícil, ya que ahora las fotos de Gene no son del todo tan fácil de distinguir de las fotos reales.

Con el tiempo, Gene mejora cada vez más en la producción de fotos de ganimales, hasta que, finalmente, Di se resigna una vez más al hecho de que no puede distinguir la diferencia entre las fotos de Gene y las originales. Llevan las fotos generadas por Gene a la subasta y los expertos no pueden creer la calidad de los nuevos avistamientos: son tan convincentes como los originales.

Las aventuras de Gene y Di cazando escurridizos ganimals nocturnos son una metáfora de uno de los avances de aprendizaje profundo más importantes de los últimos años: GANs.

3.1.3.2. Pinturas de Leonardo da Vinci

Imaginemos que queremos una GAN que genere imágenes que se parezcan a las de Leonardo da Vinci para ello usaríamos un conjunto de datos de entrenamiento de la obra de arte de da Vinci. El objetivo del Generador es crear ejemplos que no se puedan distinguir de los datos de entrenamiento, esto significa producir pinturas que se parezcan a las de da Vinci. Mientras que el objetivo del Discriminador es distinguir los ejemplos falsos producidos por el generador de los ejemplos reales provenientes del conjunto de datos de entrenamiento. En este ejemplo, el Discriminador desempeña el papel de un experto de arte que evalúa la autenticidad de las pinturas que se cree que son de da Vinci.

Asimismo, el experto de arte le da al falsificador información sobre lo que hace que una pintura sea de da Vinci y así el falsificador se vuelve a su estudio para preparar nuevas falsificaciones. A medida que pasa el tiempo, el falsificador se vuelve cada vez más competente para imitar el estilo de da Vinci y el comerciante de arte se vuelve cada vez más experto en detectar falsificaciones. Finalmente, tenemos unas excelentes obras de Leonardo da Vinci falsas.

Acabamos de ver como el falsificador de arte(el Generador) intenta engañar a un experto de arte(el Discriminador). Cuanto más convincentes sean las pinturas falsas que hace el falsificador, mejor debe ser el experto en arte para determinar su autenticidad. Esto también es cierto en la situación inversa: cuanto mejor sea el experto en arte para saber si una pintura en particular es genuina, más debe mejorar el falsificador para evitar que lo atrapen in fraganti. Esta metáfora ha sido extraída de [Bok19] y la usan muchos libros para entender el funcionamiento básico de una GANs.

3.1.3.3. Criminal-Detective

Otra metáfora que se usa frecuentemente para describir las GANs y que el propio Ian Goodfellow suele usar a menudo es la de un criminal y un detective, [Goo16]. El criminal toma el papel del Generador para falsificar dinero y el detective juega el papel del Discriminador para intentar atraparlo. Cuanto más auténticos se vuelven los billetes falsos, mejor debe ser

3. Redes Generativas Adversarias (GANs)

el detective para detectarlos, y viceversa.

3.1.3.4. Dígitos escritos a mano

Imaginemos ahora que nuestro objetivo es enseñar a una GAN a producir dígitos escritos a mano que parezcan realistas. Este sería otra buena metáfora extraída de [Bok19] para entender el funcionamiento de las GANs donde el generador construye dígitos escritos a mano falsos y el discriminador trata de distinguirlos de los verdaderos. Finalmente, se obtienen un conjunto de dígitos falsos con una gran semejanza a los verdaderos.

3.1.4. Funcionamiento

Nos basamos en el ejemplo de los dígitos escritos a mano para explicar el sistema de acción de las GANs. Vamos a estudiar los detalles enumerados del siguiente diagrama extraído de [Bok19].

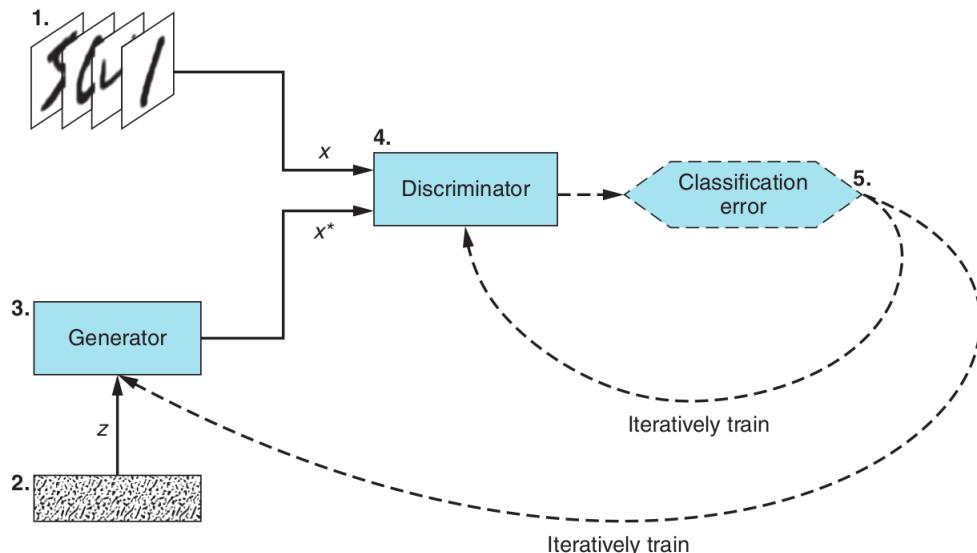


Figura 3.1.: Funcionamiento de una GANs[Bok19]

1. Conjunto de datos de entrenamiento

Es el conjunto de datos de ejemplos reales que queremos que el Generador aprenda para que genere muestras falsas, x^* , de alta calidad. Sirve como entrada a la red Discriminadora y en este caso el conjunto de datos consta de imágenes de dígitos escritos a mano.

2. Vector de ruido aleatorio

Es la entrada a la red del Generador, z , y consiste en un vector de números aleatorios que el Generador usa como punto de partida para sintetizar ejemplos falsos.

3. Red del generador

El Generador recibe como entrada un vector de números aleatorios, z , y genera ejem-

plos falsos, x^* . Su objetivo es que no se puedan distinguir los ejemplos falsos que produce de los ejemplos reales del conjunto de datos de entrenamiento.

4. Red del discriminador

El Discriminador recibe como entrada un ejemplo real, x , proveniente del conjunto de conjunto de entrenamiento o un ejemplo falso, x^* , producido por el Generador. En cada ejemplo, el Discriminador determina y genera la probabilidad de que el ejemplo sea real.

5. Entrenamiento/ajuste iterativo

Para cada una de las predicciones del Discriminador, determinamos los aciertos y errores de las probabilidades que asigna a cada muestra, como lo haríamos con un clasificador normal, y usamos los resultados para ajustar iterativamente las redes del Discriminador y del Generador a través de la retropropagación:

- Los pesos y sesgos del Discriminador se actualizan para maximizar su accuracy de clasificación. Así maximiza la probabilidad de predicción correcta: x como real y x^* como falso.
- Los pesos y sesgos del Generador se actualizan para maximizar la probabilidad de que el Discriminador clasifica erróneamente, esto es, intenta minimizar su accuracy para predecir x^* como real.

Básicamente, se entrena las dos redes mediante retropropagación utilizando la pérdida del Discriminador. El Discriminador se esfuerza por minimizar la pérdida tanto para los ejemplos reales como para los falsos, mientras que el Generador intenta maximizar la pérdida del Discriminador para los ejemplos falsos que produce.

Resumiendo el funcionamiento en dos ideas básicas, tenemos que primero vamos a desarrollar un modelo Generador que convierte un vector en una imagen candidata y a continuación, un modelo Discriminador toma como entrada una imagen candidata(real o sintética) y la clasifica en una de las dos clases.

3.1.5. ¿Por qué estudiar las GANs?

Las GANs han sido una de las innovaciones más importantes en el aprendizaje profundo. Yann LeCun, director de investigaciones de IA en facebook, llegó a decir que las GAN y sus variaciones son la mejor idea de aprendizaje profundo de los últimos 20 años.

Las GANs han capturado la imaginación de los investigadores y del público en general. Han sido incluidos en New York Times, the BBC, Scientific American y muchos otros medios de comunicación destacados.

Lo más notable es la capacidad de las GAN para crear imágenes hiperrealistas, pueden sintetizar imágenes con calidad fotorrealista.

Otro logro notable de las GAN es la traducción de imagen a imagen, por ejemplo convertir una imagen de un caballo en una imagen de una cebra.

3. Redes Generativas Adversarias (GANs)

Los casos de uso de GAN más prácticos son igual de fascinantes. Amazon está experimentando con el uso de GAN para recomendaciones de moda y en la investigación médica, las GAN se utilizan para aumentar el conjunto de datos con ejemplos sintéticos para mejorar la precisión de diagnóstico.

Por otra parte la capacidad de generar nuevos datos e imágenes hace que las GAN puedan ser peligrosas ya que el gran potencial para crear imágenes falsas creíbles es inquietante. Nos centraremos en estudiar esto con más detalle en los siguientes capítulos.

3.2. Entrenamiento de GANs

El algoritmo de entrenamiento GAN tiene dos partes principales. Estas dos partes, el entrenamiento del discriminador y el entrenamiento del generador, representan la misma red GAN en diferentes instantes de tiempo en las etapas correspondientes al proceso de entrenamiento. El siguiente diagrama extraído del libro [Bok19] detalla muy bien el entrenamiento de una GANs cuya base de datos son dígitos escritos a mano pero podría abstraerse a cualquier base de datos de imágenes.

A continuación, veamos de forma esquematizada el proceso de entrenamiento.

Por cada iteración del entrenamiento hacer

1. Entrenar al discriminador

- a) Tomar un ejemplo real aleatorio x del conjunto de datos de entrenamiento.
- b) Obtenga un vector de ruido z , y usando la red del Generador generamos un ejemplo falso x^* .
- c) Usamos la red del Discriminador para clasificar x y x^* .
- d) Calcule el error de clasificación y retropropague el error para actualizar los parámetros entrenables del Discriminador, buscando **minimizar** el error del Discriminador.

2. Entrenar al generador

- a) Obtenga un vector de ruido z , y usando la red del Generador sintetizamos un ejemplo falso x^* .
- b) Usamos la red del Discriminador para clasificar x^* .
- c) Calcule el error de clasificación y retropropague el error para actualizar los parámetros entrenables del Generador, buscando **maximizar** el error del Discriminador.

Fin

Es muy importante tener en cuenta que hay que configurar el discriminador para que se congelen sus pesos durante el entrenamiento del generador. Ya que si los pesos del discriminador pudieran actualizarse durante este proceso, entonces estaría entrenando al discriminador para que siempre clasifique como imagen real.

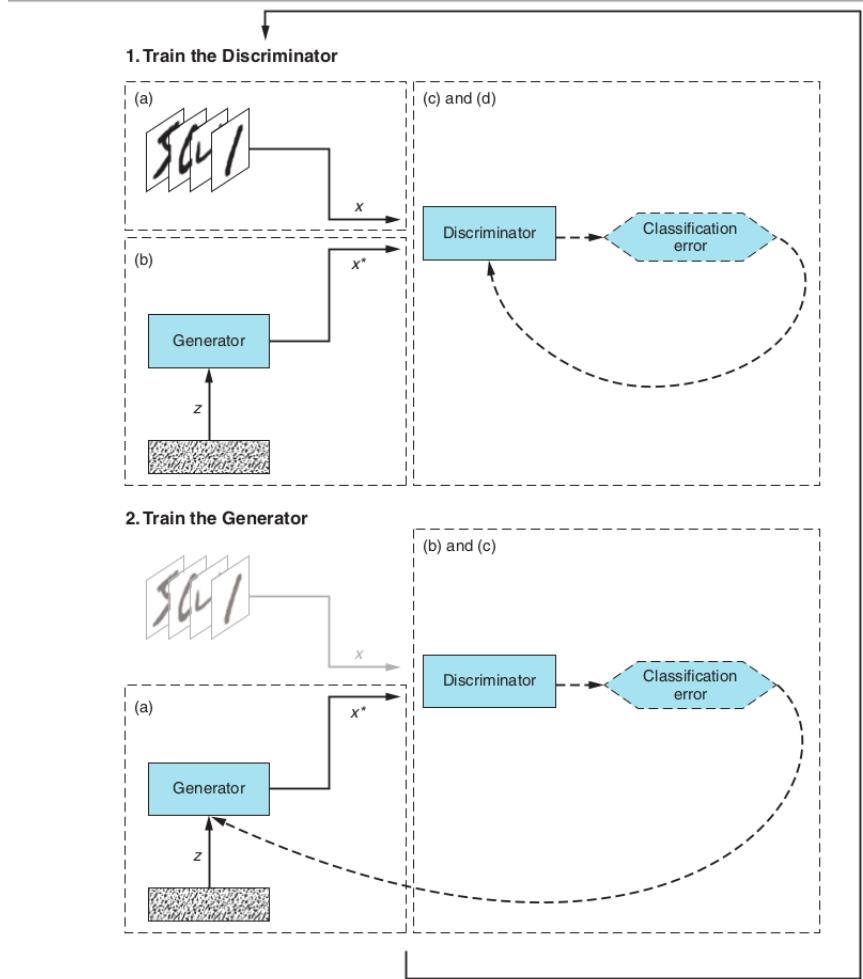


Figura 3.2.: Entrenamiento de una GANs [Bok19]

En este momento quizás se esté preguntando cuando debe detenerse el ciclo de entrenamiento GAN, ¿cómo sabemos cuando una GAN está completamente entrenada para que podamos determinar el número de iteraciones de entrenamiento? Con una red neuronal normal tenemos un objetivo específico, por ejemplo con un clasificador medimos el error de clasificación en los conjuntos de entrenamiento y validación, y detenemos el proceso cuando el error de validación comienza a empeorar para evitar que se produzca sobreajuste. En una GAN las dos redes tienen objetivos contrapuestos: cuando una red mejora, la otra empeora. Entonces nos preguntamos cómo determinamos cuando parar el entrenamiento.

En teoría de juegos puede conocerse esta configuración como un juego de suma cero y todos estos juegos tienen un equilibrio de Nash, un punto en el que ningún jugador puede mejorar su situación o recompensa cambiando sus acciones.

Imaginemos que los dos jugadores en el juego están representados por dos funciones, cada una de las cuales es diferenciable tanto con respecto a sus entradas como con respecto a

3. Redes Generativas Adversarias (GANs)

sus parámetros. El discriminador es una función D que toma x como entrada y usa $\theta^{(D)}$ como parámetros. Asimismo, el generador está definido por una función G que toma z como entrada y utiliza $\theta^{(G)}$ como parámetros. Ambos jugadores tienen funciones de costos que se definen en término de los parámetros de ambos jugadores. El discriminador desea minimizar $J^{(D)}(\theta^{(D)}, \theta^{(G)})$ y debe hacerlo controlando solo $\theta^{(D)}$. De igual forma, el generador desea minimizar $J^{(G)}(\theta^{(D)}, \theta^{(G)})$ y debe hacerlo controlando solo $\theta^{(G)}$. Debido a que el costo de cada jugador depende de los parámetros del otro jugador, pero cada jugador no puede controlar los parámetros del otro jugador, este escenario es más sencillo de describir como un juego que como un problema de optimización. La solución a un problema de optimización es un mínimo (local) pero la solución de un juego es un equilibrio Nash, esto es, una tupla $(\theta^{(D)}, \theta^{(G)})$ que es un mínimo local de $J^{(D)}$ con respecto a $\theta^{(D)}$ y un mínimo local de $J^{(G)}$ con respecto a $\theta^{(G)}$.

Las GANs alcanza el equilibrio de Nash cuando se cumplan las siguientes condiciones:

- El Generador produce ejemplos falsos que son indistinguibles de los ejemplos reales en el conjunto de datos de entrenamiento.
- El Discriminador puede adivinar aleatoriamente si un ejemplo en particular es real o falso.

Pensemos que cuando se alcanza el equilibrio, el discriminador no tiene ninguna herramienta para poder distinguir un ejemplo falso de uno verdadero. Por tanto debido a que la mitad de los ejemplos que recibe son reales y la otra mitad falsos, lo mejor que puede hacer el Discriminador es lanzar una moneda al aire y clasificar cada ejemplo como real o falso con 50 % de probabilidad. De esta forma, se dice que la GAN ha convergido pero en la práctica es casi imposible encontrar el equilibrio Nash debido a las complejidades que tienen la convergencia en juegos no convexos.

3.2.1. Funciones de coste

Siguiendo la notación estándar presentada anteriormente, sea $J^{(G)}$ la función de costo del Generador y $J^{(D)}$ la función de costo del Discriminador. Los parámetros entrenables de las dos redes los denotamos como: $\theta^{(G)}$ para el generador, $\theta^{(D)}$ para el discriminador. Por tanto, tenemos

- Función de costo del Generador: $J^{(G)}(\theta^{(D)}, \theta^{(G)})$
- Función de costo del Discriminador: $J^{(D)}(\theta^{(D)}, \theta^{(G)})$

El equilibrio de Nash se puede matematizar siguiendo esta notación, esto es, el equilibrio de Nash ocurre cuando el Generador minimiza la función $J^{(G)}(\theta^{(D)}, \theta^{(G)})$ con respecto a los parámetros entrenables del generador $\theta^{(G)}$ y, simultáneamente, la función de costo del discriminador $J^{(D)}(\theta^{(D)}, \theta^{(G)})$ se minimiza con respecto a los parámetros entrenables del discriminador $\theta^{(D)}$.

Con esta terminología el algoritmo de entrenamiento queda de la siguiente forma:

Por cada iteración del entrenamiento **hacer**

1. Entrenar al discriminador

- a) Tomar un mini batch de ejemplos reales: x
- b) Tome un mini batch de vectores de ruido aleatorio z y genere un mini batch de ejemplos falsos: $G(z) = x^*$
- c) Calcule la pérdida de clasificación para $D(x)$ y $D(x^*)$, y retropropague el error total para actualizar $\theta^{(D)}$ para **minimizar** la pérdida de clasificación.

2. Entrenar al generador

- a) Tome un mini batch de vectores aleatorios de ruidos z y genere un mini batch de ejemplos falsos: $G(z) = x^*$.
- b) Calcule la pérdida de clasificación para $D(x^*)$, y retropropague la pérdida para actualizar $\theta^{(G)}$ para **maximizar** la pérdida de clasificación.

Fin

El proceso de entrenamiento consta de dos SGD simultáneos. En cada paso, se muestran dos mini batch: un mini batch de valores x del conjunto de datos y un mini batch de valores z . Luego se realizan dos pasos del algoritmo de gradiente descendente de forma simultánea: uno que actualiza $\theta^{(D)}$ para reducir $J^{(D)}$ y otro que actualiza $\theta^{(G)}$ para reducir $J^{(G)}$. En ambos casos usar Adam suele ser una buena opción. Por otra parte, muchos autores recomendaron ejecutar más pasos de un jugador que del otro jugador pero finalmente en 2016 se afirmó que el protocolo que mejor funciona en la práctica es el descenso de gradiente simultáneo, con un paso para cada jugador.

A continuación, procedemos a especificar las funciones de costo de las dos redes. Se pueden usar varias funciones de costos diferentes dentro del marco de las GAN.

1. Función de coste del Discriminador

Todos los diferentes juegos diseñados para GAN hasta ahora usan el mismo costo para el discriminador, $J^{(D)}$. Difieren solo en términos del costo utilizado para el generador, $J^{(G)}$. El costo utilizado para el discriminador es:

$$J^{(D)}(\theta^{(D)}, \theta^{(G)}) = -\frac{1}{2}\mathbb{E}_{x \sim P_X} \log D(x) - \frac{1}{2}\mathbb{E}_{z \sim P_Z} \log(1 - D(G(z)))$$

Esta función es justo el costo de la *entropía cruzada* que se minimiza cuando se entrena un clasificador binario estándar con una salida sigmoidea.

2. Minmax

La versión más simple del juego es un juego de suma cero, en el que la suma de todos los costos del jugador son siempre cero. Por tanto, la función de **costo del generador** sería

$$J^{(G)} = -J^{(D)}$$

3. Redes Generativas Adversarias (GANs)

Debido a que $J^{(G)}$ está ligado directamente a $J^{(D)}$, podemos resumir todo el juego con un valor de función que especifica el pago del discriminador:

$$v(\theta^{(D)}, \theta^{(G)}) = -J^{(D)}(\theta^{(D)}, \theta^{(G)})$$

Los juegos de suma cero también se denominan juegos minmax porque su solución implica la minimización en un bucle externo y la maximización en un bucle interno:

$$(\theta^{(D)}, \theta^{(G)})^* = \arg \min_{\theta^{(D)}} \max_{\theta^{(G)}} v(\theta^{(D)}, \theta^{(G)})$$

3. Motivación heurística

El costo utilizado para el generador en el juego minmax es útil para el análisis teórico, pero no funciona especialmente bien en la práctica.

En el juego minmax, el discriminador minimiza la entropía cruzada, pero el generador maximiza la misma entropía cruzada. Esto es desafortunado para el generador, porque cuando el discriminador rechaza con éxito las muestras del generador con alta confianza, el gradiente del generador se desvanece.

Para resolver este problema, un enfoque es continuar utilizando la minimización de entropía cruzada para el generador. En lugar de cambiar el signo del costo del discriminador para obtener un costo para el generador, cambiamos el objetivo utilizado para construir el costo de la entropía cruzada. El **costo para el generador** entonces se convierte en:

$$J^{(G)} = -\mathbb{E}_{z \sim P_Z} \log D(G(z))$$

En el juego minmax, el generador minimiza la probabilidad logarítmica de que el discriminador sea correcto. En este juego, el generador maximiza la probabilidad logarítmica de que el discriminador se equivoque.

Esta versión del juego está motivada heurísticamente, en lugar de estar motivada por una preocupación teórica. La única motivación para esta versión del juego es asegurar que cada jugador tenga un gradiente fuerte cuando ese jugador esté perdiendo el juego.

En esta versión del juego, el juego ya no es de suma cero y no puede ser descrita con una función de un solo valor.

3.2.2. Dificultades del aprendizaje

El aprendizaje de GAN puede ser difícil en la práctica cuando G, D son redes neuronales y $\max_D v(\theta^{(G)}, \theta^{(D)})$ no es convexo. De hecho, la falta de convergencia puede hacer que las GANs no se ajusten bien. En general, no se garantiza que el descenso simultáneo del gradiente en los costos de dos jugadores alcance un equilibrio. Considere, por ejemplo, la función $v(a, b) = ab$, donde un jugador controla una función de costo ab mientras que el otro jugador controla el costo $-ab$. Si imaginamos a cada jugador haciendo pasos de gradiente infinitesimalmente pequeños, cada jugador reduciendo su propio costo a expensas del otro jugador, entonces a y b entran en una órbita circular estable, en lugar de llegar al punto de equilibrio en el origen. Tenga en cuenta que los equilibrios para un juego minmax no son mínimos locales de v . En cambio, son puntos que son simultáneamente mínimos para los costos de ambos jugadores. Esto significa que son punto silla de v que son mínimos locales con respecto a los parámetros del primer jugador y máximos locales con respecto a los parámetros del segundo jugador. Este ejemplo se estudiará con mayor detalle en el [Capítulo 6](#).

En experimentos realistas, la formulación de mejor rendimiento es una formulación que no es de suma cero. En esta formulación (3) el generador tiene como objetivo aumentar la probabilidad logarítmica de que el discriminador cometa un error, en lugar de intentar disminuir la probabilidad logarítmica de que el discriminador haga la predicción correcta.

La estabilización del aprendizaje GAN sigue siendo un problema abierto. Afortunadamente, el aprendizaje GAN funciona bien cuando la arquitectura del modelo y los hiperparámetros se seleccionan cuidadosamente.

3.3. Tipos de GANs

Vamos a presentar distintos tipos de GANs estudiados a través de las siguientes fuentes [[Bok19](#)], [[Fos19](#)] y [[BGA⁺22](#)]. Primero, vamos a presentar un tipo de GAN que incluye toda la potencialidad de las CNN en su arquitectura. A continuación, en la sección 3.3.2 cambiando las funciones de pérdida de nuestro generador y discriminador se obtienen distintos tipos de GANs: NS-GAN, WGAN y WGAN-GP. En las demás secciones, vemos como el modelo ProGAN permite generar imágenes de calidad full HD y así se podrán generar imágenes fotorrealistas, además este modelo tiene una versión mejorada conocida como StyleGAN. Por otra parte, la SGAN aporta un discriminador que se comporta de forma análoga a un clasificador semisupervisado que alcanza resultados tan buenos como los de un clasificador supervisado. Finalmente, el modelo CGAN cubre la necesidad de controlar la producción de imágenes ya que permite orientar la generación de nuevas imágenes a través de condicionar cada imagen con información adicional.

3. Redes Generativas Adversarias (GANs)

3.3.1. DCGAN

Tanto nuestro generador como nuestro discriminador se implementarán como redes neuronales convolucionales(CNN). Esta idea fue presentada en 2016 por Alec Radford, Luke Metz y Soumith Chintala [RMC15] y mostró una de las innovaciones más importantes en la técnica de las GAN iniciada dos años antes. Fue la primera vez que se logró incorporar CNN a un modelo GAN ya que anteriormente se intentó aprovechar su utilidad por ejemplo con el enfoque de LAPGAN. Esta técnica utilizaba las redes convolucionales dentro de una pirámide laplaciana pero modificaba la arquitectura GAN subyacente. Esta alternativa apareció para no alterar el entrenamiento de la GAN ya que si incluimos la CNN sin modificar la arquitectura de la GAN aparecían muchas dificultades en su entrenamiento.

LAPGAN demostró el gran potencial que tenía el uso de CNN en las GANs. Por consiguiente, Radford y sus colaboradores decidieron introducir las técnicas necesarias para incluir CNN dentro de la GAN sin modificar su arquitectura subyacente y sin tener que cambiarla a un modelo más complejo como sucedía con LAPGAN. Por ejemplo, una de las técnicas que utilizó fue BatchNormalization.

BatchNormalization se encarga de reducir las variaciones en las distribuciones de valores de entrada entre las capas durante el entrenamiento y esto lo consigue mediante la normalización de la salida de cada capa antes de que esta pase como entrada a la siguiente capa. De esta forma, disminuye cualquier interdependencia no deseada entre los parámetros a través de las capas. Esto ayuda a acelerar el proceso de entrenamiento de la red y a aumentar su robustez. Además, la normalización por batch ha demostrado ser esencial para muchas otras arquitecturas del aprendizaje profundo.

DCGAN demostró la versatilidad del modelo GAN: sabemos que el discriminador y el generador pueden aproximarse mediante cualquier función diferenciable y en este caso se aproxima por una tan compleja como una red convolucional multicapa.

Finalmente, tenemos que resaltar que el modelo DCGAN, construido como una red GAN con redes neuronales convolucionales como generador y discriminador, obtiene un rendimiento superior en tareas de procesamiento de imagen.

3.3.2. Versiones principales de la configuración de las GANs

El enfoque de Min-Max (también conocido como MMGAN) trae bastantes problemas, entre ellos cabe destacar la lenta convergencia del discriminador. Por consiguiente, aparecieron nuevas configuraciones como NS-GAN que pierde muchas garantías matemáticas con respecto a la versión min-max pero funcionó mucho mejor o la versión de Wasserstein que tiene una buena base teórica y obtiene un rendimiento superior.

3.3.2.1. NS-GAN

Esta versión fue propuesta en el paper original de las GAN y su funcionamiento difiere en que en lugar de tener dos funciones de pérdida compitiendo entre sí, tenemos dos funciones de pérdida independientes. Su motivación heurística se vió en 3.

$$J^{(G)} = -\mathbb{E}_{z \sim P_Z} [\log D(G(z))]$$

$$J^{(D)} = \mathbb{E}_{x \sim P_{\mathcal{X}}}[\log D(x)] + \mathbb{E}_{z \sim P_{\mathcal{Z}}}[\log(1 - D(G(z)))]$$

El objetivo de Non-Saturating GAN(NS-GAN) es que los gradientes no se saturen, no se acerquen a 0, ya que esto conllevaría a una convergencia lenta por que las actualizaciones de los pesos con Backpropagation serían 0 o muy pequeñas. De hecho, tanto usando la máxima verosimilitud como el enfoque Min-Max los gradientes tienden a 0.

No tenemos garantía de que esta configuración converja al equilibrio de Nash y puede que su entrenamiento no converja pero empíricamente hay pruebas que demuestran que funciona mejor que la versión min-max.

Como el Generador aprende más rápido también se obtiene que el Discriminador aprende más rápido y por tanto se obtiene lo deseado.

3.3.2.2. WGAN

Wasserstein GAN(WGAN), propuesto en [ACB17], obtiene mejores resultados y propone una mejor función de pérdida. Utiliza la distancia del movimiento de la tierra como función de pérdida ya que se correlaciona con la convergencia del generador y con la calidad de la muestras generadas. Además, se obtiene una convergencia más estable de la GAN. La función de pérdida es

$$W(P_{\mathcal{X}}, P_{\mathcal{Z}}) = \mathbb{E}_{x \sim P_{\mathcal{X}}} [F_w(x)] - \mathbb{E}_{z \sim P_{\mathcal{Z}}} [F_w(G_{\theta}(z))]$$

- El discriminador trata de maximizar $\mathbb{E}_{x \sim P_{\mathcal{X}}} [F_w(x)] - \mathbb{E}_{z \sim P_{\mathcal{Z}}} [F_w(G_{\theta}(z))]$
- El generador trata de minimizar $\mathbb{E}_{x \sim P_{\mathcal{X}}} [F_w(x)] - \mathbb{E}_{z \sim P_{\mathcal{Z}}} [F_w(G_{\theta}(z))]$

Tenemos una pérdida sin logaritmos y así se consigue un entrenamiento más ajustable, donde la función F_w debe ser una función continua $1 - Lipschitz$, $F_w \in Lip_1(\mathbb{R}^n)$. La función F_w toma el rol del discriminador donde se ajustan sus parámetros w al entrenar el modelo y la función G_{θ} toma el rol del generador cuyos parámetros a aprender son θ .

En cuanto al entrenamiento de WGAN tenemos que destacar las siguientes diferencias con respecto a la GAN estandar:

- Utiliza la pérdida de Wasserstein
- Se entrena usando etiquetas de 1 y -1, en lugar de 1 y 0.
- Eliminamos la activación sigmoidea de la capa final del discriminador para que las predicciones no estén en el rango $[0, 1]$
- Para que se cumpla la restricción de $1 - Lipschitz$ se recortan los pesos del Discriminador de forma que se encuentren en un rango pequeño y esto se realiza después de cada batch de entrenamiento
- Entrene al discriminador varias veces entre cada actualización del generador.

3. Redes Generativas Adversarias (GANs)

3.3.2.3. WGAN-GP

Esta versión fue presentada en [GAA⁺17], aquí el generador de WGAN-GP se define y compila exactamente igual que el generador WGAN. La diferencia recae en el discriminador que debemos de incluirle los siguientes cambios:

- En la función de pérdida del discriminador debemos añadir un término de penalización del gradiente.
- No vamos a recortar los pesos del Discriminador
- No utilizar capas de BatchNormalization

3.3.3. ProGAN

Progressive GAN, también conocida como PGGAN o ProGAN, es un tipo de configuración de la GAN que ha logrado generar imágenes fotorrealistas en Full HD. Se presentó en una de las principales conferencias de machine learning, the International Conference on Learning Representations (ICLR), en 2018.

El espacio latente va a tener una serie de propiedades relevantes, podemos encontrar los desplazamientos vectoriales que permiten realizar pequeñas modificaciones en imágenes. Por ejemplo, si conocemos el desplazamiento que introduce gafas en una imagen de una cara, entonces ese mismo desplazamiento introducirá gafas en nuevas imágenes que se generen. Asimismo, podemos tomarnos dos vectores aleatorios, luego irnos moviendo de forma incremental e ir produciendo las imágenes hasta llegar a la imagen que se corresponde con el segundo vector. Por tanto, estamos usando la interpolación en nuestro espacio latente.

Un equipo finlandés de NVIDIA publicó un paper, [KALL17], donde se explicó las cuatro innovaciones clave para el entrenamiento en las que se fundamentan los modelos ProGAN:

- **Crecimiento progresivo y suavizado en capas de mayor resolución**

Vamos a explicar esta mejora del entrenamiento de una GANs a partir del siguiente ejemplo. Imaginemos que tenemos un alpinista que se encuentra entre montañas y quiere alcanzar un valle, este es el funcionamiento de SGD. Cuando el alpinista se vaya acercando a un valle vamos a aumentar la complejidad aplicándole zoom al terreno. Esta idea permite realizar optimizaciones en la búsqueda del valle para que el alpinista llegue antes a su destino. Aumentar la resolución del terreno a medida que avanza el alpinista es lo que se conoce como **crecimiento progresivo**.

Aumentar la resolución del terreno puede que progresivamente introduzcamos más complejidad ya que en nuestro ejemplo aparecerán objetos que antes no veíamos. Volviendo a nuestra GAN, con el crecimiento progresivo lo que se logra es pasar de pocas capas convolucionales de baja resolución(pocos parámetros que aprender) a muchas capas convolucionales de alta resolución. Este paso se realiza a medida que se va entrenando.

- **Desviación estándar de minibatch**

Vamos a usar la desviación estándar de todos los píxeles del minibach para determinar si las muestras que se están generando son variadas o no. Por ejemplo, para la base de

datos de MNIST no nos gustaría que nuestra GANs generase una gran proporción de 8 y pocos de los demás números.

Con esta innovación lo único que necesita aprender el Discriminador es que si la desviación estándar es baja en las imágenes del batch entonces es probable que sea falsa. Por tanto, el generador deberá aumentar la variabilidad de las muestras generadas para así engañar al discriminador. Nos podemos dar cuenta que la implementación es bastante sencilla y solo afecta al discriminador

- **Learning rate igualados**

Consiste en usar una inicialización normal estándar simple y luego ir escalando los pesos por capa en tiempo de ejecución. Con esta técnica podemos tomar learning rates de tamaños apropiados en cada dirección del gradiente.

- **Normalización vectorial por píxeles**

No usamos BatchNormalization porque consume mucha memoria y por eso surge esta innovación que toma cada píxel de la salida de la capa de activación justo antes de que se le pase como entrada a la siguiente capa y lo normaliza.

3.3.4. SGAN

Vamos a recordar las diferencia clave entre aprendizaje supervisado, semisupervisado y no supervisado. En el aprendizaje supervisado tenemos una etiqueta y para cada dato x del conjunto de datos, mientras que en el aprendizaje no supervisado no se utilizan etiquetas y solo tenemos los datos x del conjunto de datos de entrenamiento. El aprendizaje semisupervisado tienen etiquetas solo un pequeño subconjunto del conjunto de datos de entrenamiento, entonces tenemos datos con etiquetas y datos sin etiquetas.

En 2016, Tim Salimans, Ian Goodfellow y compañeros de OpenAI [SGZ⁺16] demostrarón que el modelo de GAN semisupervisado obtuvo un accuracy muy cercano al obtenido con una GAN supervisada.

La GAN semisupervisada, también conocida como SGAN, es una GAN cuyo discriminador es un clasificador multiclase. Antes solo se distinguía si la imagen era verdadera o falsa, ahora debemos de distinguir si es verdadera el discriminador nos dirá su etiqueta(si tenemos N clases, nos dirá un número del 1 al N) o si es falsa entonces su etiqueta será $N + 1$. Por tanto, nuestro discriminador distinguirá entre $N + 1$ clases. Esta tarea de distinguir entre $N + 1$ clases en lugar de 2 clases(verdadero/falso) le agregará complejidad a la arquitectura de SGAN y a su entrenamiento.

El generador de SGAN es el mismo que el de una GAN original pero el discriminador es diferente ya que puede recibir tres tipos de entrada: un ejemplo real etiquetado $,(x,y)$, un ejemplo real sin etiqueta $,x$, o un ejemplo falso $,x^*$. La misión del nuevo discriminador es clasificar la entrada en su clase si es real o rechazarla si es falsa.

En el proceso de entrenamiento debemos de calcular la función de pérdida para los tres tipos de entrada: $D((x,y))$, $D(x)$ y $D(x^*)$. De hecho, podemos referirnos a $D(x)$ y $D(x^*)$ como la pérdida supervisada y a $D((x,y))$ como la pérdida no supervisada.

3. Redes Generativas Adversarias (GANs)

Los modelos GAN vistos hasta ahora centran su estudio en la red del generador ya que el objetivo de la red discriminadora ha sido ayudar al generador a mejorar la calidad de su producción de imágenes. En cambio en una SGAN nos interesamos más en el discriminador ya que queremos convertirlo en un clasificador semisupervisado que alcance resultados tan buenos como un clasificador supervisado. Ahora es el generador el que ayuda al discriminador a cumplir su objetivo.

3.3.5. CGAN

Este modelo surgió de la necesidad de controlar la producción de nuevas imágenes, por ejemplo si nuestra GAN sintetiza dígitos escritos a mano, nos gustaría controlar que genere el número 7 en lugar del 9 o otro ejemplo sería controlar que nuestra GAN genere un rostro masculino o femenino. De manera análoga, imagina que somos detectives y queremos resolver un asesinato, tenemos a un testigo que describe al asesino, entonces sería de gran utilidad que esta descripción se le pudiese introducir a una GAN para que genere rostros que se asemejen al asesino. Todos estos ejemplos nos han mostrado una clara intuición del objetivo de CGAN.

El modelo Conditional GAN, también conocido como CGAN, fue uno de los primeros modelos que permitieron la generación de datos específicos. Fue introducido en 2014 por el estudiante de doctorado de la Universidad de Montreal Mehdi Mirza y el arquitecto Simon Osindero, [MO14]. CGAN es una red cuyo generador y discriminador se condicionan durante el entrenamiento mediante el uso de información adicional, que podría ser una etiqueta de clase, un conjunto de etiquetas o una descripción.

Se diferencia de una SGAN porque el discriminador no aprende a identificar la clase a la que corresponde la entrada sino mas bien se encarga de *aceptar* pares (x, y) coincidentes, esto es el dato x coincide con su etiqueta o información adicional y , y *rechazar* los pares (x, y) que no coinciden o los pares donde la imagen x es falsa incluso si coinciden. Por tanto, el generador debe no solo producir datos realistas sino también debe producir datos que coincidan con sus etiquetas para así engañar al discriminador. Nos damos cuenta que añadir información adicional esta obligando al generador a sintetizar un determinado tipo de salida y al discriminador a aceptar solo los ejemplos reales que coincidan con su información adicional.

- Si G es nuestra función del generador, z un vector de ruido y y la etiqueta , entonces la salida es $G(z, y) = x^*|y$.
- Si D es nuestra función del discriminador, entonces acepta como entrada ejemplos reales con su etiqueta (x, y) , y ejemplos falsos con la etiqueta que hemos usado para generarlos $(x^*|y, y)$.

3.3.6. CycleGAN

Las GAN permiten la traducción de imagen a imagen para ello cambiamos la entrada del generador para que en lugar de recibir un vector de ruido reciba una imagen y genere una nueva imagen. Es decir, estamos mapeando o traduciendo de la imagen de entrada a la imagen de salida del generador.

Un grupo de investigadores de UC Berlekey propuso la pérdida de consistencia de ciclo, [ZPIE17], donde necesitamos dos generadores, uno que traduzca del dominio 1 al dominio 2 llamado G_{12} y otro que traduzca la vuelta del dominio 2 al 1 G_{21} . La pérdida de consistencia de ciclo consiste en medir la pérdida a nivel de píxel que existe entre una imagen x y $G_{21}(G_{12}(x))$.

Al tener dos generadores tenemos dos discriminadores D_1 y D_2 . La pérdida adversarial es la pérdida que estudia como de real es la imagen que se genera al hacer la traducción.

Además, tenemos otra pérdida llamada pérdida de identidad que se encarga de que la CycleGAN conserve su estructura de color de la imagen. Para ello introducimos un término de regularización que nos permita mantener el color de la imagen generada similar al color de la imagen original. La utilidad de esto es poder recuperar la imagen original a partir de la imagen generada. Hay que destacar que esta pérdida no es estrictamente necesaria para que nuestra CycleGAN funcione aunque si proporciona mejores resultados si se incluye.

Por tanto, la función de pérdida esta formada por tres componentes: $\mathcal{L}_1(G_{12}, D_{P_X}), \mathcal{L}_2(G_{21}, D_{P_Y})$, $\mathcal{L}_{cycle}(G_{12}, G_{21})$.

$$\mathcal{L}(G_{12}, G_{21}, D_{P_X}, D_{P_Y}) := \mathcal{L}_1(G_{12}, D_{P_X}) + \mathcal{L}_2(G_{21}, D_{P_Y}) + \lambda \mathcal{L}_{cycle}(G_{12}, G_{21})$$

donde

- $\mathcal{L}_1(G_{12}, D_{P_X}) = \mathbb{E}_{x \sim P_X} [\log(D_{P_X}(x))] + \mathbb{E}_{y \sim P_Y} [\log(1 - D_{P_X}(G_{12}(y)))]$
- $\mathcal{L}_2(G_{21}, D_{P_Y}) = \mathbb{E}_{y \sim P_Y} [\log(D_{P_Y}(y))] + \mathbb{E}_{x \sim P_X} [\log(1 - D_{P_Y}(G_{21}(x)))]$
- $\mathcal{L}_{cycle}(G_{12}, G_{21}) = \mathbb{E}_{y \sim P_Y} [\|G_{21}(G_{12}(y)) - y\|_1] + \mathbb{E}_{x \sim P_X} [\|G_{12}(G_{21}(x)) - x\|_1]$

Nuestro problema minmax sería

$$\min_{G_{12}, G_{21}} \max_{D_X, D_Y} \mathcal{L}(G_{12}, G_{21}, D_{P_X}, D_{P_Y})$$

Su arquitectura esta formada por dos CGAN unidas o también puede verse como un Auto-encoder ([Sección A.2](#)) donde el espacio latente tiene la misma dimensión que la entrada y la salida. Tenemos dos caminos, uno que pasa del dominio 1 al dominio 2 y otro del dominio 2 al 1. En cada camino se siguen estos dos pasos:

1. La imagen x se le proporciona al Discriminador correspondiente con el dominio de x (si x pertenece al dominio 1 entonces sería D_1 , en caso contrario D_2) para que nos diga si es real o falsa.
2. La imagen x se le proporciona al Generador(si x pertenece al dominio 1 entonces sería G_{12} , en caso contrario G_{21}) que la traduce en la imagen \tilde{x} y luego se le pasa al Discriminador correspondiente con el dominio de \tilde{x} (si x pertenece al dominio 1 entonces sería D_2 , en caso contrario D_1) para que la evalúe y nos diga si es real o falsa.

En cuanto a la arquitectura del generador, se suelen usar capas convolucionales con skip connection en el codificador, mientras que en la arquitectura del decodificador se usan capas deconvolucionales con una última capa convolucional que sintetiza la imagen de salida con el mismo tamaño que la imagen de entrada.

3. Redes Generativas Adversarias (GANs)

En la arquitectura del discriminador notamos que se obtienen un conjunto de valores que se consideran como un conjunto de minidiscriminadores que luego se promediarán para obtener la salida correspondiente.

En resumen, este modelo se encarga de realizar traducciones de imagen a imagen a partir de tres funciones de pérdida: la pérdida de ciclo para medir la diferencia entre la imagen original y la generada, la pérdida adversarial que nos asegura imágenes realistas y la pérdida de identidad para preservar el color. Los dos generadores suelen usar la arquitectura U-Net y los dos discriminadores se diseñan con una arquitectura basada en PatchGAN.

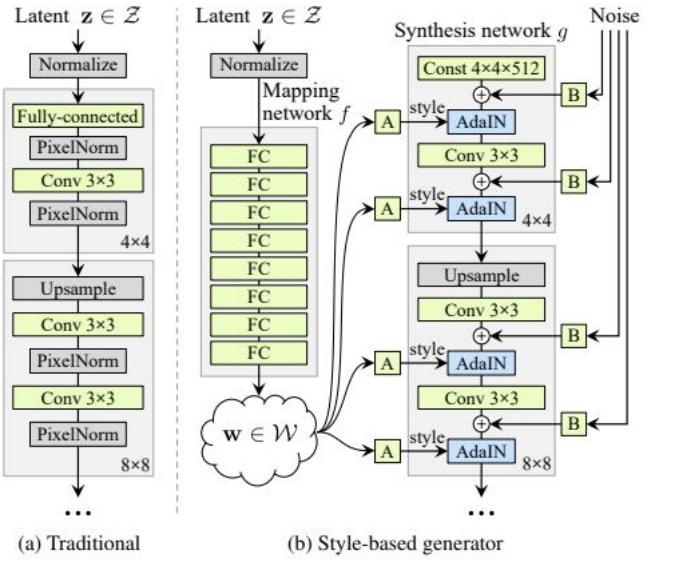
3.3.7. StyleGAN

Fue introducido por Karras et.al. 2019 como una versión mejorada de ProGAN y se ha convertido en el estándar de oro para la edición de imágenes faciales ya que presenta un generador de última generación para las imágenes de alta calidad. Este modelo se centra en controlar las diferentes características visuales a través de las capas progresivas ProGAN y las divide en tres tipos de características: gruesas (con una resolución de hasta 82, por ejemplo forma de la cara), medianas (resolución de 162 a 322, por ejemplo peinados, ojos abiertos o cerrados) y finas (resolución de 642 a 10241, por ejemplo color de ojos, cabello, piel). No está supervisado y sin embargo, su espacio latente se comporta sorprendentemente bien, de hecho admite aritmética latente lineal. Por ejemplo, admite agregar un vector, que representa una determinada edad, a un conjunto de códigos latentes, produciendo como resultado que las imágenes generadas a partir de esos códigos latentes donde se le ha agregado el vector presenten a individuos de esa determinada edad.

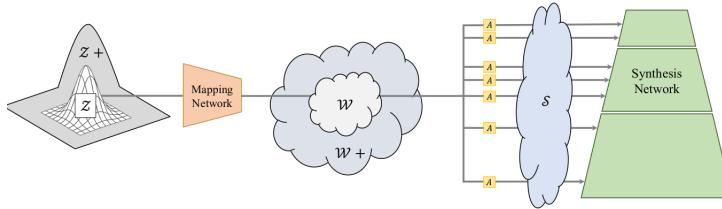
Primero, se propuso una arquitectura de StyleGAN formada por capas de modulación de estilo (AdaIN) que están diseñadas con objeto de controlar el estilo de las imágenes generadas ajustando las estadísticas de los mapas de características a lo largo de la ruta generativa. El generador parte de una constante aprendida C , que representa el epicentro de la distribución, y toda la información y potencia generativa de la red se inyecta a través del estilo y de un vector de ruido aleatorio adicional n . Las capas de inyección de estilo utilizaron el mecanismo de normalización de instancia adaptativa (AdaIN).

Las capas AdaIn no fue el único cambio que se propuso, otro cambio fue que en vez de usar un código latente z muestreado de alguna distribución gaussiana \mathcal{Z} , StyleGAN introducía una red de mapeo que convertía estos códigos en vectores de otro espacio latente intermedio \mathcal{W} . Esta arquitectura inicial puede verse en la siguiente ilustración extraída de [BGA⁺22] donde se comparan las diferencias de un modelo GAN tradicional con el modelo StyleGAN.

Más adelante, se propusieron versiones mejoradas donde las arquitecturas abordaron problemas como la adherencia de textura. Una cuestión importante en este modelo es que disponemos de más de un espacio latente y es muy común trabajar con extensiones de estos espacios. Veamos en la siguiente imagen extraída de [BGA⁺22] el funcionamiento básico que consiste en partir de un código latente aleatorio z muestreado por la distribución normal \mathcal{Z} , este código es transformado al espacio latente aprendido \mathcal{W} a través de una red MLP. Al resultado se le aplican distintas transformaciones afines aprendidas, denotadas por A y así se obtiene el espacio S . Las transformaciones afines se aprenden durante el entrenamiento de cada capa de la red de síntesis y también tenemos que aprender la distribución del espacio


 Figura 3.3.: Arquitectura StyleGAN[BGA⁺22]

latente \mathcal{W} . El espacio S a diferencia de \mathcal{W} contiene varios códigos latentes para una sola imagen, uno por cada bloque de transformación afín mientras que en \mathcal{W} solo teníamos un código latente. Es importante resaltar que a partir de 2019 se propuso trabajar con espacios latentes extendidos $\mathcal{W}+$ y $\mathcal{Z}+$ para que en dichos espacios en vez de contener un único código contuviesen tantos códigos latentes como S .


 Figura 3.4.: Arquitectura StyleGAN y sus espacios latentes[BGA⁺22]

3.3.8. Resumen

En esta sección podemos observar todos los tipos de GANs junto con unas descripciones básicas de su funcionamiento.

3. Redes Generativas Adversarias (GANs)

Tipo	Funcionamiento	Función de pérdida
DCGAN	-CNN -BatchNormalization	Entropía cruzada
NS-GAN	-Funciones de pérdida independientes	$J^{(G)} = -\mathbb{E}_{z \sim P_Z} [\log D(G(z))]$ $J^{(D)} = \mathbb{E}_{x \sim P_X} [\log D(x)] + \mathbb{E}_{z \sim P_Z} [\log(1 - D(G(z)))]$
WGAN	-Pérdida de Wasserstein - $f_\omega \in 1 - Lipschitz$, para ello recortar pesos de D -Etiquetas de -1 y 1 -Entrenar D varias veces entre cada actualización de G	$J^{(D)} = \mathbb{E}_{x \sim P_X} [F_w(x)] - \mathbb{E}_{z \sim P_Z} [F_w(G_\theta(z))]$ $J^{(G)} = -\mathbb{E}_{z \sim P_Z} [F_w(G_\theta(z))]$
WGAN-GP	Difiere de WGAN en la implementación Discriminador (GP=penalización gradiente) -Añadir GP a la pérdida de D -No recortar los pesos de D -No usar capas BatchNormalization	$J^{(D)} = \mathbb{E}_{x \sim P_X} [F_w(x)] - \mathbb{E}_{z \sim P_Z} [F_w(G_\theta(z))] + GP$ $J^{(G)} = -\mathbb{E}_{z \sim P_Z} [F_w(G_\theta(z))]$
CycleGAN	-Traducción de imagen a imagen, G recibe de entrada una imagen -Tiene tres funciones de pérdidas -Generadores-U-Net -Discriminadores-PatchGAN	$\mathcal{L}_1(G_{12}, D_{P_X}) + \mathcal{L}_2(G_{21}, D_{P_Y}) + \lambda \mathcal{L}_{cycle}(G_{12}, G_{21})$ $\mathcal{L}_1(G_{12}, D_{P_X}) = \mathbb{E}_{x \sim P_X} [\log(D_{P_X}(x))] + \mathbb{E}_{y \sim P_Y} [\log(1 - D_{P_X}(G_{12}(y)))]$ $\mathcal{L}_2(G_{21}, D_{P_Y}) = \mathbb{E}_{y \sim P_Y} [\log(D_{P_Y}(y))] + \mathbb{E}_{x \sim P_X} [\log(1 - D_{P_Y}(G_{21}(x)))]$ $\mathcal{L}_{cycle}(G_{12}, G_{21}) = \mathbb{E}_{y \sim P_Y} [\ G_{21}(G_{12}(y)) - y\ _1] + \mathbb{E}_{x \sim P_X} [\ G_{12}(G_{21}(x)) - x\ _1]$

Tabla 3.1.: Tipos de GANs caracterizados por su función de pérdida

Tipo	Funcionamiento
ProGAN	<ul style="list-style-type: none"> -Crecimiento progresivo y suavizado en capas de mayor resolución -Desviación estándar de minibatch -Learning rate igualados -Normalización vectorial por píxeles
SGAN	<ul style="list-style-type: none"> -El discriminador es un clasificador multiclas que puede recibir 3 tipos de entrada: $(x, y), x, x^*$ y aprende a distinguir ejemplos falsos de reales y asignar la clase correcta a ejemplos reales -El generador es el mismo que una GANs original -El discriminador se pretende convertir en un clasificador semisupervisado
CGAN	<ul style="list-style-type: none"> -Necesidad de controlar la producción de nuevas imágenes -Generador y discriminador se condicionan durante el entrenamiento con información adicional -El discriminador acepta pares (x, y) coincidentes y rechaza pares que no coinciden o pares donde la imagen x es falsa incluso si coinciden
StyleGAN	<ul style="list-style-type: none"> -Versión mejorada de ProGAN -Capas AdaIN controlan el estilo de las imágenes generadas -Disponemos de más de un espacio latente y es común trabajar con extensiones de estos

Tabla 3.2.: Otros tipos de GANs

4. Creación y manipulación de Deepfakes

En este capítulo vamos a estudiar distintas técnicas de manipulación de imágenes faciales como son los métodos de Deepfakes o los métodos para detectar tales manipulaciones. Actualmente, el término de Deepfakes también se le conoce como ultrafalsificación. Más concretamente, abordaremos cuatro tipos de manipulaciones donde en cada una se estudiarán su técnica, las bases de datos usadas y su estado del arte en técnicas de detección a día de hoy. [TVRF⁺²⁰]

En la actualidad cada vez es más fácil sintetizar caras inexistentes o manipular el rostro de una persona en una imagen/vídeo. Esto se debe gracias a la alta accesibilidad a datos públicos a gran escala y a la evolución de las técnicas de Deep Learning que eliminan muchos pasos de edición manual como son los Autoencoders o las GANs. Existen cuatro tipos de manipulaciones faciales que procederemos a explicar con detalle y son la síntesis de cara completa, el intercambio de identidad, la manipulación de atributos y el intercambio de expresiones faciales. Además de esto, revisaremos otras manipulaciones entre las que cabe destacar la transformación facial. La síntesis de toda la cara permite crear imágenes de caras falsas a partir de imágenes de caras verdaderas haciendo uso de modelos GANs, más concretamente de StyleGANs y ProGANs. En el intercambio de identidad vamos a reemplazar el rostro de una persona en un vídeo por otro rostro generando vídeos falsos a través de FaceSwap y del enfoque DeepFakes. La tercera manipulación, la manipulación de atributos, consiste en manipular los atributos faciales de las imágenes de rostros para así generar rostros falsos, por ejemplo manipular los ojos de una cara. Finalmente, el intercambio de expresiones faciales permite intercambiar la expresión facial de un vídeo y generar vídeos falsos modificando expresiones faciales de la persona.

Estos cuatro tipos de manipulaciones serán ordenados de mayor a menor nivel de manipulación. Podemos apreciar como el primer tipo de manipulación presenta el mayor nivel de manipulación facial ya que tenemos que crear imágenes de rostros completamente falsos, mientras que el segundo tipo son rostros falsos pero en vídeos, por tanto hay un nivel inferior de manipulación. El tercer tipo de manipulación se puede observar que admite un nivel de manipulación inferior a los dos primeros tipos y superior al cuarto tipo.

Tanto los vídeos como las imágenes falsas han democratizado la manipulación de los contenidos visuales para influir en la opinión pública y difundir desinformación. Por tanto, aunque pueda usarse para buenos fines también existe un gran riesgo de utilizar esta tecnología para fines maliciosos, como por ejemplo la creación de noticias falsas (fake news).

4.1. Síntesis de toda la cara

Esta manipulación se encarga de crear caras no existentes, imágenes de caras falsas, normalmente haciendo uso de una potente GAN, por ejemplo StyleGAN. Se suele usar en diferentes sectores como son los videojuegos, el modelado 3D, pero también puede tener un mal uso

4. Creación y manipulación de Deepfakes

en aplicaciones dañinas que generen por ejemplo perfiles falsos en redes sociales entre otros malos usos.

Hay cuatro base de datos de imágenes falsas y todas ellas basadas en las mismas arquitecturas GAN: ProGAN y StyleGAN. Una cuestión relevante a mencionar es que cada imagen falsa puede ser caracterizada por una huella dactilar GAN que más adelante veremos su utilidad. Por otra parte, las cuatro bases de datos donde disponemos de todas las imágenes falsas se basan en imágenes reales de bases de datos públicas como son CelebA,FFHQ,CASIA-WebFace, VGGFace2,... y a partir de los modelos GANs se construyen las imágenes falsas.

- **100k Generated Images [KLA19]**

Esta base de datos contiene 100000 imágenes falsas y fue generada usando la arquitectura StyleGAN que fue entrenada con la base de datos FFHQ. StyleGAN es una versión mejorada de ProGAN, que como se vio anteriormente es un tipo de GAN donde se va mejorando el entrenamiento del generador y discriminador de forma progresiva, y además presenta una arquitectura de generador distinta a la GAN básica.

- **100k Face [AI18]**

Esta formada por 100000 imágenes falsas que se generaron usando StyleGAN y al contrario que antes, esta red fue entrenada usando 29000 fotos de 69 modelos diferentes, considerando imágenes de cara de escenarios mas controlados, por ejemplo un fondo plano.

- **DFFD-Diverse Fake Face Dataset [DLS⁺19]**

Esta formada por 100000 caras falsas creadas a través del modelo preentrenado ProGAN y 200000 con StyleGAN.

- **iFakeFaceBD[NTVR⁺20]**

Esta base de datos contiene 250000 caras falsas creadas con StyleGAN y 80000 creadas con ProGAN. A diferencia de las otras bases de datos, aquí las huellas dactilares GAN se eliminaron a través de GANprintR.

A continuación, vamos a ver el estado del arte de los diferentes enfoques de detección de caras falsas generadas mediante esta manipulación.

1. McCloskey and Albright (2018) se dieron cuenta que el color era diferente entre la imagen real y la falsa. Por tanto, propusieron un sistema basado las GANs y centrado en las características del color. Como clasificador final se uso SVM. Finalmente, se obtuvo un AUC=70 % con la base de datos NIST MFC2018.
2. Wang et al. (2019) observaron que monitorear el comportamiento de la neurona en un modelo GANs podría servir para detectar rostros falsos. Su enfoque propuesto lo llamaron FakeSpotter y lo probaron usando rostros reales y falsos creados por InterfaceGAN y StyleGAN. Finalmente, se tuvo un ACC=84,7% y también se uso como clasificador final SVM.
3. Guarnera et al. (2020) propusieron un sistema de detección basado en análisis de trazas convolucionales. Las características se extraían usando el algoritmo de maximización de expectativas(EM) y como clasificador final se podía usar k-NN,SVM o LDA. Este enfoque fue probado con caras falsas generadas por modelos AttGAN, GAN, StarGAN, StyleGAN y StyleGAN2, y se obtuvo un ACC=99.81 %.

4.2. Intercambio de identidad

4. Nataraj et al.(2019) estudió los sistemas de detección inspirados en estegoanálisis, propuso un sistema de detección basado en una combinación de matrices de concurrencia de píxeles y CNN. Las bases de datos que utilizó para generar caras falsas con StyleGAN fue 100k-Fake y obtuvo en la detección un valor de ERR=12.3 %.
5. Yu et al.(2019) propuso un sistema de detección cuya arquitectura de red mapeaba una imagen de entrada en su correspondiente huella digital, se van aprendiendo las huellas digitales del modelo. Este enfoque fue probado usando caras reales de CelebA y caras falsas creadas a través de diferentes tipos de GANs, se obtuvo un ACC=99.5 %.
6. Marra et al. (2019) propuso un método de detección de aprendizaje incremental multitarea para detectar y clasificar nuevos tipos de imágenes generadas por GAN. En base al algoritmo iCarRL para el aprendizaje incremental se propuso dos soluciones diferentes: MT_MC(MultiTask_MultiClassifier) y MT_SC(MultiTask_SingleClassifier). Se usaron distintos enfoques GANs para generar caras falsas como CycleGAN, ProGAN, Glow, StarGAN, StyleGAN. Finalmente, este enfoque de detección, basado en el modelo de XceptionNet, alcanzó un ACC=99.3 %.
7. Dang et al. (2020) propuso mecanismos de atención y usar modelos populares de CNN como son XceptionNet y VGG16. Se consideró caras reales de las bases de datos CelebA, FFHQ y FaceForensic, y imágenes falsas generadas a través de ProGAN y StyleGAN. Finalmente, se obtuvo AUC=100 % y ERR=0.1 %.
8. Neves et al. (2020) generó sistemas de detección de última generación y consideró condiciones experimentales, es decir, probó con imágenes tanto en escenarios controlados (por ejemplo un fondo plano) como en escenarios más salvajes. Partieron de 150000 rostros falsos creados por StyleGAN, 80000 creados por ProGAN, la base de datos 100k-face y iFakeFaceBD donde se eliminaron las huellas dactilares GAN. Los resultados en escenarios controlados fueron similares a los mejores enfoques previos, alcanzando ERR=0.02 %. Mientras que en escenarios más desafiantes hubo una disminución del rendimiento y se obtuvo ERR=4.5 % con iFakeFaceDB.
9. Hulzebosch et al. (2020) consideró diferentes escenarios como modelo cruzado, datos cruzados y postprocesamiento. Estos detectores se basaron en la red Xception y el enfoque de autoencoder conocido como ForensicTransfer. Finalmente, se alcanzó un valor de ACC=99.8 %

Podemos apreciar como los enfoques de detección [7](#) y [8](#) de rostros falsos alcanzaron muy buenos resultados y consiguieron detectar perfectamente imágenes donde se presenta la manipulación de síntesis de toda la cara.

4.2. Intercambio de identidad

Consiste en reemplazar la cara de una persona en un vídeo con la cara de otra persona. Existen dos diferentes enfoques para este tipo de manipulación: FaceSwap y las técnicas de deep learning conocida como DeepFakes. Suele usarse mucho en la industria del cine pero también puede usarse con malos fines: vídeos en pornografía, bulos, fraudes financieros,..

Anteriormente, las manipulaciones se hacían a nivel de imagen mientras que ahora el objetivo es reemplazar el rostro de una persona en un vídeo por el rostro de otra persona. Las

4. Creación y manipulación de Deepfakes

bases de datos se van a dividir en dos generaciones ya que presentan ciertas diferencias que comentaremos más adelante. Asimismo, cabe resaltar que a diferencia de las bases de datos anteriores, estas bases de datos van a almacenar tanto vídeos reales como falsos.

1. Primera generación

- **UADF** [[LCL18](#)]

Está formada por 49 vídeos reales de Youtube y 49 vídeos falsos. Los vídeos falsos se crearon con la aplicación FaceApp, intercambiando en los 49 vídeos reales los rostros por la cara de Nicolas Cage.

- **DeepfakeTIMIT** [[KM18](#)]

Contiene 620 vídeos falsos de 32 personas de la base de datos VidTIMIT. Estos vídeos falsos fueron generados a partir un algoritmo de intercambio de rostros basado en GAN. La red se toma del modelo CycleGAN utilizando los pesos de FaceNet. Aquí de acuerdo a los escenarios podemos agrupar los vídeos en dos tipos de calidades: de baja calidad(LQ) con imágenes de 64x64 píxeles y de alta calidad(HQ) con imágenes de 128x128 píxeles.

- **FaceForensics++** [[RCV⁺19](#)]

Formado por 1000 vídeos reales de Youtube, 1000 vídeos falsos generados con el algoritmo de FaceSwap y 1000 vídeos falsos generados usando enfoque de DeepFakes. El algoritmo de FaceSwap consiste en un alineamiento de caras, una optimización de Gauss-Newton y blending en imágenes para hacer el intercambio. Mientras que el enfoque de DeepFakes usa dos autoencoders con un encoder compartido.

2. Segunda generación

- **DeepFakeDetection** [[AI19](#)]

Tenemos 363 vídeos reales de 28 actores en 16 diferentes escenas y se crean 2068 vídeos falsos a través del enfoque Deepfakes. Además, se consideran distintas calidades de vídeo: RAW, HQ y LQ.

- **Celeb-DF** [[LYS⁺20](#)]

Contiene 890 vídeos reales extraídos de YouTube y 5639 vídeos falsos creados a través de una versión mejorada de un algoritmo público de generación de DeepFakes. Así, se consigue crear vídeos falsos de mejores cualidades visuales.

- **DFDC Preview** [[DHP⁺19](#)]

Esta formada de 1131 vídeos reales de 66 actores y 4119 vídeos falsos generados por un enfoque desconocido. Esta base de datos almacena un contenido de 470GB.

Los vídeos falsos almacenados en las bases de datos de la primera generación se caracterizan porque presentan distintas debilidades que limitan la naturalidad de la imagen y facilitan la detección de caras falsas. Entre estas debilidades destacan las caras sintetizadas de baja calidad, los diferentes contraste de color en las máscaras falsas, los límites visibles en máscaras falsas, elementos visibles del vídeo original y artefactos extraños entre fotogramas. No obstante, los vídeos falsos almacenados en las bases de datos de la segunda generación, incluyen

mejoras que aumentan su naturalidad y dificultan la detección de caras falsas, como son la presencia de distintos escenarios, distintas condiciones de luz, distancia de la persona a la cámara, variaciones entre poses,. De hecho, esta generación presenta una mayor cantidad de vídeos falsos.

A continuación, vamos a ver el estado del arte de los diferentes enfoques de detección de vídeos falsas generadas mediante esta manipulación.

1. Korshunov and Marcel (2018) basaron sus estudios en aprender características de audio y visuales. Se estudió un primer caso que centraba su atención en MFCC como características de audio y la distancia entre los puntos de referencia de una boca como características visuales, estas características se aprendían con un PCA seguido de una RNN basada en LSTM para detectar vídeos reales o falsos. Luego, se estudió un segundo caso donde el enfoque de detección se basaba en tomar como características caras sin procesar y en IQM. Aquí se usó como clasificador final PCA con LDA o SVM. Finalmente, la mejor propuesta fue IQM+SVM y obtuvo $ERR=3.3\%$ con DeepfakeLIMIT(LQ) y $ERR=8.9\%$ con DeepfakeLIMIT(HQ).
2. Matern et al. (2019) propuso un sistema de detección basado en aspectos visuales como son el color de ojos, los reflejos perdidos y detalles perdidos en el área de los ojos y los dientes. Como clasificadores finales se usó la regresión logística y MLP, obteniendo un valor de $AUC=85.1\%$ aplicando MLP a bases de datos privadas.
3. Yang et al. (2019) realizaron un estudio basado en las diferencias entre las poses de la cabeza utilizando un conjunto de puntos de referencia faciales. Cuando se tengan estas características extraídas, se normalizan y se considera un SVM como clasificador. Finalmente, se evaluó este modelo de detección para distintas bases de datos y el mejor resultado se alcanzó con UADFV consiguiendo un $AUC=89\%$.
4. Agarwal and Farid (2019) se basó tanto en las expresiones faciales como en los movimientos de cabeza. Para la extracción de características se usó OpenFace2 para los movimientos de músculos faciales y cuatro características relacionadas con los movimientos de la cabeza. Como clasificador final se usó SVM y la mejor detección alcanzó un valor de $AUC=96.3\%$ donde se usó vídeos reales de Youtube y vídeos falsos generados por FaceSwapGAN.
5. Jung et al. (2020) se centró en el estudio del parpadeo de los ojos, más concretamente fue el número de parpadeos y su frecuencia los que determinaban si un vídeo era real o falso. Este enfoque evaluado sobre una base de datos propia obtuvo un $ACC=87.5\%$.
6. Li et al. (2019) propuso un sistema de detección basado en CNNs para detectar la presencia de deformaciones faciales ya que se dieron cuenta que la creación de deepfake incluía pequeñas deformaciones que se podían detectar. Se entrenaron los modelos VGG16, ResNet50, ResNet101 y ResNet152. Este enfoque se probó para las bases de datos UADFV y DeepFakeLIMIT, con esta segunda base de datos se alcanzó los mejores resultados hasta el momento.
7. Afchar et al. (2018) centró sus estudio en modelos basados en características mesoscópicas y de esteganoanálisis. Propuso dos modelos: 'Meso-4' que consistía en una red CNN compuesta por 4 capas convolucionales seguida de una capa totalmente conectada y

4. Creación y manipulación de Deepfakes

'MesoInception-4' una variante de Inception introducida en el modelo anterior. Finalmente, se probó para bases de datos privadas alcanzando un ACC=98.4 % y también alcanzó resultados notables para FaceForensics++.

8. Zhou et al. (2018) fusionó la extracción de dos tipos de características, las características de estegeoanálisis extraídas con el clasificador final SVM y las características de rostros extraídas con una CNN para determinar si una imagen es falsa o no. Se alcanzó resultados muy buenos para la base de datos CelebA.
9. Rössler et al. (2019) mostró cinco sistemas de detección que fueron evaluados usando la base de datos FaceForensics++. Estos sistemas son: un sistema basado en CNN entrenada usando las características estegoanálisis, un sistema basado en CNN cuyas capas convolucionales se encargan de eliminar el contenido de alto nivel, un sistema basado en CNN con una capa de globalPooling, el sistema de detección conocido como MesoInception-4 y por último un sistema basado en una CNN XceptionNet que fue el que proporcionó mejores resultados tanto para DeepFakes como para FaceSwap.
10. Nguyen et al. (2019) propuso un sistema CNN que utilizaba el aprendizaje multitarea y se basaba en los autoencoders. Este enfoque alcanzó EER=15.07 % cuando se evaluó con el método de manipulación FaceSwap para la base de datos FaceForensic.
11. Nguyen et al. (2019) presentaron posteriormente un nuevo método de detección basado en Capsule Networks que alcanzó una precisión superior al 90 % para la base de datos FaceForensic++ pero en otras bases de datos no alcanzó muy buenos resultados.
12. Dang et al. (2019) planteó un sistema basado en CNN y mecanismos de atención con objeto de mejorar los mapas de características del modelo clasificador. Con la base de datos DFFD se obtuvo un AUC=99.43 % y un EER=3.1 %.
13. Dolhansky et al. (2019) presentó tres modelos de detección: un modelo CNN con 6 capas de convolución y 1 capa totalmente conectada, un modelo de XceptionNet entrenado usando solo imágenes de rostros y un modelo XceptionNet entrenado usando imágenes completas. Los mejores resultados se obtuvieron con el segundo modelo, se alcanzó valores de Precision = 93.0 % y Recall = 8.4 %.
14. Wang and Dantcheva (2020) se centraron en enfoques basados en 3DCNN para considerar tanto la información espacial como la de movimiento y lograron resultados muy buenos en los vídeos de baja calidad de FaceForensics.
15. Güera and Delp (2018) propuso sistemas de detección que consideraban tanto las características a nivel de la imagen como a nivel temporal para detectar videos falsos. Trataron con una combinación de CNN (InceptionV3) y RNN (un modelo LSTM), y finalmente, se obtuvo una precisión del 97.1 % con una base de datos propia.
16. Sabir et al. (2019) propuso la misma idea anterior pero en lugar de usar un modelo preentrenado, se va a usar un modelo entrenado de extremo a extremo. Al evaluarlo con la base de datos FaceForensics se obtuvo AUC=97.9 % para el método de DeepFake Y AUC=96.3 % para FaceSwap.
17. Tolosana et al. (2020) presentó modelos basados en las características de las regiones faciales que obtuvieron muy buenos resultados con valores AUC=91.0 % para DFDC Y AUC=83.6 % para Celeb-DF .

4.3. Manipulación de atributos

Se le conoce también como edición de rostros o retoque facial y su funcionamiento consiste en modificar algunos atributos de la cara, como puede ser el color del cabello, la piel, el género, la edad, añadirle gafas... Se suelen implementar con una potente GAN como puede ser StarGAN y un ejemplo de uso son las conocidas aplicaciones móviles FaceApp.

Hay muy pocas bases de datos disponibles para la investigación de este tipo de manipulación facial ya que con los distintos enfoques GAN se pueden generar todas las bases de datos que queramos. Por tanto, en esta manipulación en lugar de citar las bases de datos nos encargamos de citar los últimos enfoques GANs desde los más antiguos hasta los más nuevos.

- IcGAN, también conocida como la GAN condicional invertible.
- StarGAN
- attGAN
- STGAN

Si queremos comparar este tipo de manipulación con los otros tipos de manipulaciones, necesitamos una base de datos. Hasta ahora DFFD es la única base de datos pública que considera este tipo de manipulaciones faciales. Esta formada por 18416 imágenes falsas generadas a través de FaceApp y 79960 imágenes falsas generadas con StarGAN.

A continuación, vamos a ver el estado del arte de los diferentes enfoques de detección de caras falsas generadas mediante esta manipulación.

1. Wang et al. (2019) propuso el enfoque llamado FakeSpotter que usó la idea que se propuso en la síntesis de cara completa [2](#). FakeSpotter extraía como características el comportamiento de las neuronas de caras reales y falsas en los sistemas de reconocimiento facial profundo, como puede ser VGG-Face, OpenFace y FaceNet, y después usaba SVM como clasificador final. Se probó este enfoque usando caras reales de las bases de datos CelebA-HQ y FFHQ y caras falsas creadas a través de InterFaceGAN y StyleGAN. Finalmente, se consiguió un valor ACC=84,7 %.
2. Nataraj et al. (2019) propuso un sistema de detección inspirado en estegeoanálisis parecido al que se propuso para la síntesis de cara completa. [4](#). Logró un valor de ACC=99.4% para un conjunto de datos falsos creados a partir de la red StarGAN entrenada para la base de datos CelebA.
3. Bharati et al. (2016) presentó un enfoque de aprendizaje profundo basado en RBM cuyas entradas son parches faciales para así aprender si la imagen es original o retocada. Se generaron dos bases de datos falsas a partir de ND-IIITD y a partir de un conjunto de imágenes de famosos descargadas de Internet. Para la primera base de datos se obtuvo un valor ACC=96.2 % y para la segunda base de datos un valor ACC=87.1 %.
4. Jain et al. (2019) propuso un extracto de características de parches faciales formado por una CNN de 6 capas convolucionales y 2 capas totalmente conectadas, junto con el clasificador SVM. Los datos falsos se generaron a partir de ND-IIITD donde se obtuvo

4. Creación y manipulación de Deepfakes

un ACC=99.6 % y a partir de las imágenes falsas generadas con StarGAN donde se tuvo un ACC=99.7 %

5. Tariq et al. (2018) evaluó el uso de arquitecturas CNN como puede ser VGG16,VGG19, ResNet110, XceptionNet116,... Para probar su enfoque uso imágenes reales de CelebA y imágenes falsas generadas a partir de ProGAN donde se alcanzó ACC=99.99 % con el mejor modelo CNN y imágenes falsas generadas a partir de Adobe Photoshop donde se logró un valor inferior ACC=74.9 %.
6. Dang et al. (2019) se centró en utilizar mecanismos de atención para procesar y mejorar los mapas de características de los modelos CNN. Las imágenes falsas se crearon usando dos enfoques: imágenes falsas creadas a través de FaceApp y imágenes falsas creadas con StarGAN. Ambos enfoque obtuvieron muy buenos resultados con la base de datos DFFD, AUC=99.9 % y ERR=1.0 %.
7. Wang et al. (2019) propuso un modelo de clasificación global basado en redes residuales dilatadas, DRN, para predecir si la cara se ha deformado o no. Usando AdobePhotoshop para generar rostros falsos se obtuvo un rendimiento del 99.8 %.
8. Marra et al. (2019) propuso un sistema de detección que fue descrito en [6](#) y logró un ACC=99.3 % cuando se usó el modelo XceptionNet.
9. Zhang et al. (2019) mostró un sistema de detección basado en las características extraídas del dominio del espectro en vez de usar las características de los píxeles de la imagen. Se probó este modelo usando la red StarGAN para generar imágenes falsas y se alcanzó un valor de ACC=100 %.
10. Rathgeb et al. (2020) propuso un sistema de detección basado en Photo Response Non-Uniformity(PRNU). Se probó con una base de datos generada a partir de cinco aplicaciones móviles y logró ERR=13.7 %.

Podemos apreciar que la mayoría de sistemas de detección de este tipo de manipulación se basan en modelos de aprendizaje profundo y proporcionan muy buenos resultados, esto se debe a que podemos acceder a las huellas dactilares GAN de las imágenes falsas.

4.4. Intercambio de expresiones

Esta manipulación también se le conoce como reconstrucción facial ya que su objetivo es modificar la expresión facial de la persona. Las técnicas más populares son Face2Face y NeuralTexture que reemplazan la expresión facial de una persona en un vídeo por la expresión facial de otra persona.

- FaceForensic es una base de datos que se centró en el enfoque Face2Face para generar los vídeos falsos. Este enfoque transfiere la expresión de un vídeo inicial a un vídeo destino a partir de la selección manual de los fotogramas clave.
- FaceForensic++ es un nuevo enfoque basado en NeuralTexture que utiliza los datos del vídeo inicial para aprender una textura neuronal de la persona y así generar el vídeo falso.

En total tenemos en nuestra base de datos 1000 vídeos reales extraídos de Youtube y 2000 vídeos falsos, esto es, 1000 vídeos falsos por cada enfoque. Además, tenemos tres tipos de calidades de vídeo: RAW,HQ y LQ .

A continuación, vamos a resumir los resultados obtenidos para la detección de manipulaciones de intercambio de expresión.

1. Matern et al. (2019) centró sus estudios en las características visuales que podíamos observar en los vídeos , como vimos en [2](#). Evaluando esta detección para la base de datos FaceForensics++ usando la técnica de manipulación Face2Face se obtuvo un valor de AUC=86.6 %.
2. Afchar et al. (2018) presentó un método de detección basado en aprender las características mesoscópicas y estegeoanálisis (como se vió en [7](#)). Al evaluar este método usando los vídeos falsos generados con el enfoque Face2Face y la base de datos FaceForensics++ se obtuvieron muy buenos resultados, especialmente en los vídeos de alta calidad donde se alcanzó un ACC=96.8 %. También se probó usando los vídeos falsos generados con el enfoque NeuralTexture pero se alcanzó resultados peores.
3. Rössler et al. (2019) presentó métodos basados en el aprendizaje profundo (como se vio en [9](#)) y demostró que los métodos basados en XceptionNet evaluados tanto en vídeos falsos generados por Face2Face como con NeuralTexture obtuvieron valores de ACC cerca del 100 % en vídeos de calidad RAW.
4. Nguyen et al. (2019) propuso un enfoque basado en aprendizaje multitareas (como se vio en [10](#)) donde se uso como clasificador final un autoencoder y se evaluó para la base de datos FaceForensics++ con vídeos de calidad HQ. Finalmente, para el método Face2Face se consiguió un EER=7.1 % y para NeuralTexture un ERR=7.8 %.
5. Dang et al. (2020) presentó un enfoque de detección basado en los mecanismos de atención (como se vio en [7](#)) y logró valores de AUC=99.4 % y EER=3.4 % para la base de datos FaceForensics++ y el método Face2Face.
6. Wang and Dantcheva (2020) se centró en estudiar los modelos de aprendizaje profundo basados en 3DCNN como se vio [14](#). Se logró resultados muy buenos para los vídeos de baja calidad con la base de datos FaceForensics usando los dos métodos, Face2Face y NeuralTexture.
7. Sabir et al. (2019) presentó un método de detección basado en redes convolucionales recurrentes(RNN) para así analizar tanto la imagen como la información temporal. Cuando se probó este método para la base de datos FaceForensics++ y el método Face2Face, se obtuvo un valor de AUC=94.3 %.
8. Amerini et al. (2019) propuso trabajar con los campos de flujo óptico para analizar las posibles diferencias entre fotogramas y así podía extraer el movimiento de una escena. Además, se conoce que en un vídeo falso el flujo óptico no es natural y podemos aprovechar esta idea. Finalmente, se obtuvo valores de ACC=81.6 % usando redes VGG16 y ResNet50 con la base de datos FaceForensics++ y el método Face2Face.

Finalmente, podemos notar como la mayoría de los enfoques mostrados aquí los hemos usado en los sistemas de detección de intercambio de identidad.

4.5. Otras manipulaciones faciales

- **Transformación facial**

Es un tipo de manipulación facial cuyo objetivo es crear nuevos rostros faciales que se asemejen a la información facial de dos o más personas. Aquí nos centramos en crear muestras falsas a nivel de imagen y no de vídeo como se realizaba en la manipulación de intercambio de identidad. Claramente, este tipo de manipulación facial sería una gran amenaza para los sistemas de reconocimiento de rostros.

- **De-Identificación facial (de-ID)**

Su misión es eliminar la información de identidad que hay en la imagen facial o en el vídeo para así preservar la privacidad de la persona. Una forma muy simple sería ofuscar la cara desenfocándola o pixelandola y otra opción posible podría ser el intercambio de identidad facial.

- **Audio-to-Video y text-to-Video**

También se le conocen como falsificaciones profundas de sincronización de labios y consisten en la síntesis de vídeos falsos a partir de audios o textos. Si tenemos un audio y un vídeo, lo que realiza esta manipulación es sincronizar los labios de la persona del vídeo para que parezca que está emitiendo el audio proporcionado, mientras que si se toma como entrada un texto nuestro resultado será un nuevo vídeo en el que la persona del vídeo se sincroniza para que diga las palabras del texto.

Parte II.

Fundamentos matemáticos de las redes generativas adversarias (GANs)

5. Conceptos previos

En esta sección trataremos de explicar el concepto de la medida para así determinar cuando dos medidas son similares. Además, nos será de gran utilidad saber cuando una medida es continua y conocer determinados conceptos asociados a la medida de probabilidad. Primero, vamos a recordar los conceptos de la medida abstracta y la integral asociada a una medida, para luego proceder a detallar la medida de Jordan y de Lebesgue. Además, se mostrará el teorema de Radon Nikodyn que será una pieza clave en nuestro estudio. Finalmente, veremos la traducción de todos estos conceptos a conceptos probabilísticos ya que trataremos con medidas de probabilidad y se definirán una serie de conceptos necesarios sobre probabilidad. Los resultados expuestos en esta sección se han estudiado de las asignaturas cursadas a lo largo de mi formación en el grado y también se ha usado el libro [Tao11].

5.1. Teoría de la medida

5.1.1. Concepto abstracto de medida y espacio de medida

Definición 5.1. Sea X un conjunto, \mathcal{A} se dice que es una **σ -álgebra** sobre X si es una familia de subconjuntos de X cumpliendo las siguientes propiedades:

1. $\emptyset \in \mathcal{A}$
2. Si $A \in \mathcal{A}$, entonces el complementario $A^c := X \setminus A$ también está en \mathcal{A}
3. Si $A_1, A_2, \dots \in \mathcal{A}$, entonces $\bigcup_{n=1}^{\infty} A_n \in \mathcal{A}$

Definición 5.2. Sea X un conjunto y \mathcal{A} una σ -álgebra sobre X . Una función μ que le asigna a conjuntos de la σ -álgebra un valor de la recta real se dice que es una **medida** si cumple las siguientes propiedades:

- No negatividad: Para todo $A \in \mathcal{A}$, tenemos $\mu(A) \geq 0$
- Conjunto vacío nulo: $\mu(\emptyset) = 0$
- Aditividad numerable (o σ -aditividad): Para toda colección numerable $\{A_k\}_{k=1}^{\infty}$ de conjuntos disjuntos dos a dos, $\mu(\bigcup_{k=1}^{\infty} A_k) = \sum_{k=1}^{\infty} \mu(A_k)$

A la pareja (X, \mathcal{A}) se le conoce como **espacio medible** y a los conjuntos de \mathcal{A} se le conocen como **conjuntos medibles**.

Definición 5.3. Dada una familia \mathcal{Y} de subconjuntos de un conjunto X , llamaremos la σ -álgebra generada por \mathcal{Y} , a la menor σ -álgebra que contiene a \mathcal{Y} . Denotaremos a veces a esta σ -álgebra por $\sigma(\mathcal{Y})$.

Es claro que $\sigma(\mathcal{Y})$ está siempre bien definida. De hecho, $\sigma(\mathcal{Y})$ es la intersección de todas las σ -álgebras que contienen a \mathcal{Y} . (Nótese que al menos hay una σ -álgebra que contiene a \mathcal{Y} , la familia $\mathcal{P}(X)$ de todos los subconjuntos de X)

5. Conceptos previos

Definición 5.4. Se define la σ -álgebra de Borel en \mathbb{R} como la σ -álgebra generada por la clase de todos los intervalos; esto es, si notamos $\mathcal{Y} \subset \mathcal{P}(\mathbb{R})$ a la clase de todos los intervalos de \mathbb{R} , la σ -álgebra de Borel, \mathcal{B} , se define como $\mathcal{B} = \sigma(\mathcal{Y})$.

El espacio de Borel es el espacio medible $(\mathbb{R}, \mathcal{B})$ y un conjunto Borel es cualquier subconjunto de \mathbb{R} que pertenezca a \mathcal{B} .

Proposición 5.1. Sea $\mu : \mathcal{A} \rightarrow [0, +\infty]$ una medida en la σ -álgebra \mathcal{A} . Se cumple:

1. **Monotonicidad**

Si E, F son medibles y $E \subset F$, entonces $\mu(E) \leq \mu(F)$.

2. **Cierre booleano** $E \cup F, E \cap F, E \setminus F$, y $E \triangle F$ son conjuntos medibles. Esto es, existen las medidas $\mu(E \cup F), \mu(E \cap F), \mu(E \setminus F)$, y $\mu(E \triangle F)$.

3. **Aditividad finita**

Si k es un número natural, y E_1, \dots, E_k son medibles y disjuntos, entonces $\mu(E_1 \cup \dots \cup E_k) = \mu(E_1) + \dots + \mu(E_k)$.

4. **Subaditividad finita**

Si k es un número natural, y E_1, \dots, E_k son medibles, entonces $\mu(E_1 \cup \dots \cup E_k) \leq \mu(E_1) + \dots + \mu(E_k)$.

5. **Inclusión-exclusión de dos conjuntos**

Si E, F son medibles, entonces $\mu(E \cup F) + \mu(E \cap F) = \mu(E) + \mu(F)$.

Proposición 5.2. Sea $\mu : \mathcal{A} \rightarrow [0, +\infty]$ una medida en la σ -álgebra \mathcal{A} . Se cumple:

1. **Subaditividad numerable**

Si $E_1, E_2, \dots \in \mathcal{B}$ es una secuencia numerable de conjuntos medibles, entonces $\mu(\bigcup_{n=1}^{\infty} E_n) \leq \sum_{n=1}^{\infty} \mu(E_n)$

2. **Convergencia monótona hacia arriba**

Si $E_1 \subset E_2 \subset \dots$ son medibles, entonces $\mu(\bigcup_{n=1}^{\infty} E_n) = \lim_{n \rightarrow \infty} \mu(E_n) = \sup_n \mu(E_n)$

3. **Convergencia monótona hacia abajo**

Si $E_1 \supset E_2 \supset \dots$ son medibles, y $\mu(E_n) < \infty$ para al menos un n , entonces $\mu(\bigcap_{n=1}^{\infty} E_n) = \lim_{n \rightarrow \infty} \mu(E_n) = \inf_n \mu(E_n)$

5.1.2. Integral asociada a una medida

5.1.2.1. Aplicaciones medibles

Definición 5.5. Sean $(\Omega_1, \mathcal{A}_1)$ y $(\Omega_2, \mathcal{A}_2)$ dos espacios medibles. Una función $f : \Omega_1 \rightarrow \Omega_2$ se denomina **aplicación medible** si y solamente si $\forall B \in \mathcal{A}_2, f^{-1}(B) \in \mathcal{A}_1$.

Dada una aplicación $f : (\Omega_1, \mathcal{A}_1) \rightarrow (\Omega_2, \mathcal{A}_2)$, la σ -álgebra generada por f , que notaremos por $\sigma(f)$, es la menor σ -álgebra sobre Ω_1 que hace que f sea medible. Teniendo en cuenta que $f^{-1}(\mathcal{A}_2)$ es una σ -álgebra, es fácil ver que

$$\sigma(f) = f^{-1}(\mathcal{A}_2) = \{A \subset \Omega_1 / \exists B \in \mathcal{A}_2 \text{ y } A = f^{-1}(B)\}.$$

Definición 5.6. Una función medible finita es una aplicación medible de (Ω, \mathcal{A}) en $(\mathbb{R}^n, \mathcal{B}^n)$, para cualquier $n \in \mathbb{N}$; esto es, una aplicación medible en la que el espacio final es el espacio de Borel $(\mathbb{R}^n, \mathcal{B}^n)$. Si el espacio inicial (Ω, \mathcal{A}) es también un espacio de Borel, se habla de funciones medibles Borel ó simplemente de funciones Borel.

$$f : (\Omega, \mathcal{A}) \longrightarrow (\mathbb{R}^n, \mathcal{B}^n) \text{ aplicación medible} \iff \forall B \in \mathcal{B}^n, f^{-1}(B) \in \mathcal{A}$$

Si $n = 1$ se habla de **funciones medibles unidimensionales** y si $n > 1$ de **funciones medibles multidimensionales**.

5.1.2.2. Funciones simples. Caracterización de funciones medibles

Iniciamos el concepto de función simple para posteriormente caracterizar a una función medible como límite de una sucesión de funciones simples. Esta caracterización será fundamental para definir el concepto de integral asociada a una medida.

Definición 5.7. Una función $f : (\Omega, \mathcal{A}) \longrightarrow (\mathbb{R}, \mathcal{B})$ es una **función simple** si y solo si existe una lista de subconjuntos A_1, \dots, A_n que forman una partición de Ω y existen $x_1, \dots, x_n \in \mathbb{R}$ tal que

$$f(\omega) = \sum_{i=1}^n x_i \mathbb{1}_{A_i}(\omega) = \begin{cases} x_1 & \text{si } \omega \in A_1 \\ x_2 & \text{si } \omega \in A_2 \\ \vdots & \vdots \\ x_n & \text{si } \omega \in A_n \end{cases}$$

Proposición 5.3. Caracterización de funciones medibles mediante funciones simples

- Una función $f : (\Omega, \mathcal{A}) \longrightarrow (\mathbb{R}, \mathcal{B})$ no negativa es medible si y solamente si existe una sucesión no decreciente $\{f_n\}_{n \in \mathbb{N}}$ de funciones simples no negativas tal que

$$\lim_{n \rightarrow \infty} f_n(\omega) = f(\omega), \forall \omega \in \Omega$$

- Una función $f : (\Omega, \mathcal{A}) \longrightarrow (\mathbb{R}, \mathcal{B})$ es medible si y solamente si existe una sucesión convergente $\{f_n\}_{n \in \mathbb{N}}$ de funciones simples tal que

$$\lim_{n \rightarrow \infty} f_n(\omega) = f(\omega), \forall \omega \in \Omega$$

5.1.2.3. Integración de funciones medibles

Para introducir el concepto de integral de una función medible vamos a usar la caracterización de funciones medibles mediante funciones simples. Primero, vamos a definir la integral de una función simple no negativa, a continuación se definirá para funciones medibles no negativas y finalmente para funciones medibles cualesquiera.

Además, estudiaremos los teoremas de convergencia monótona, Fatou-Lebesgue y convergencia dominada.

5. Conceptos previos

- Sea $(\Omega, \mathcal{A}, \mu)$ un espacio de medida y $f : (\Omega, \mathcal{A}, \mu) \rightarrow (\mathbb{R}, \mathcal{B})$ una función simple no negativa, $f(\omega) = \sum_{i=1}^n x_i \mathbb{1}_{A_i}(\omega)$, siendo A_1, \dots, A_n una lista de subconjuntos que forman una partición de Ω y $x_1, \dots, x_n \geq 0$.

Definición 5.8. Se denomina integral de f respecto de μ y se representa mediante $\int_{\Omega} f d\mu$ a la expresión

$$\int_{\Omega} f d\mu = \sum_{i=0}^n x_i \mu(A_i)$$

- Si $x_i = 0$ y $\mu(A_i) = +\infty$, por convenio $x_i \mu(A_i) = 0$
- La integral existe ya que, por ser f no negativa, en la suma no aparecen a la vez sumandos iguales a $+\infty$ y $-\infty$.
- f es integrable si y sólo si su integral es finita.
- Puesto que la representación de una función simple no es única, se demuestra que la integral no depende de dicha representación (la integral está definida de forma única)

Definición 5.9. Sea $A \in \mathcal{A}$, se denomina integral de f respecto de μ en A y se representa mediante $\int_A f d\mu$ a la expresión

$$\int_A f d\mu = \int_{\Omega} f \mathbb{1}_A d\mu = \sum_{i=0}^n x_i \mu(A_i \cap A)$$

- Sea $(\Omega, \mathcal{A}, \mu)$ un espacio de medida y $f : (\Omega, \mathcal{A}, \mu) \rightarrow (\mathbb{R}, \mathcal{B})$ una función medible no negativa, $f = \lim_n f_n$, siendo $\{f_n\}_{n \in \mathbb{N}}$ una sucesión no decreciente de funciones simples no negativas.

Definición 5.10. Se denomina integral de f respecto de μ y se representa mediante $\int_{\Omega} f d\mu$ a la expresión

$$\int_{\Omega} f d\mu = \lim_n \int_{\Omega} f_n d\mu$$

- Para todo $n \in \mathbb{N}$, $\int_{\Omega} f_n d\mu$ es la integral de una función simple no negativa, definida en la sección anterior.
- La integral existe ya que, por las propiedades de la integral de una función simple no negativa, $\{\int_{\Omega} f_n d\mu\}_{n \in \mathbb{N}}$ es una sucesión no decreciente y por tanto, existe el límite.
- f es integrable si y solamente si su integral es finita
- Se demuestra que la integral no depende de la sucesión de funciones simples que converge a f (la integral está definida de forma única).

Definición 5.11. Sea $A \in \mathcal{A}$, se denomina integral de f respecto de μ en A y se representa mediante $\int_A f d\mu$ a la expresión

$$\int_A f d\mu = \int_{\Omega} f \mathbb{1}_A d\mu = \lim_n \int_{\Omega} f_n \mathbb{1}_A d\mu = \lim_n \int_A f_n d\mu$$

Teorema 5.1. Teorema de convergencia monótona

Sea $\{f_n\}_{n \in \mathbb{N}}$ una sucesión creciente de funciones medibles no negativas y $f = \lim_n f_n$, entonces

$$\lim_n \int_{\Omega} f_n d\mu = \int_{\Omega} f d\mu$$

Aplicando este teorema puede demostrarse fácilmente que si f es una función medible no negativa, la integral es σ -aditiva respecto al dominio de integración; esto es

$$\forall \{A_n\}_{n \in \mathbb{N}} \subset \mathcal{A} \text{ con } A_i \cap A_j = \emptyset, i \neq j \rightarrow \int_{\bigcup_n A_n} f d\mu = \sum_n \int_{A_n} f d\mu$$

- Sea $(\Omega, \mathcal{A}, \mu)$ un espacio de medida y $f : (\Omega, \mathcal{A}, \mu) \rightarrow (\mathbb{R}, \mathcal{B})$ una función medible,

$$f = f^+ - f^-$$

donde $f^+ = \max(f, 0)$ y $f^- = -\min(f, 0)$, sus partes positivas y negativas, son funciones medibles no negativas.

Definición 5.12. Se denomina integral de f respecto de μ y se representa mediante $\int_{\Omega} f d\mu$ a la expresión

$$\int_{\Omega} f d\mu = \int_{\Omega} f^+ d\mu - \int_{\Omega} f^- d\mu$$

siempre que exista la diferencia, esto es, siempre que f^+ o f^- (al menos una de ellas) sea integrable.

- $\int_{\Omega} f^+ d\mu$, $\int_{\Omega} f^- d\mu$ son integrales de funciones medibles no negativas, definidas en la sección anterior.
- La integral existe por la propia definición
- f es integrable si y solamente si su integral es finita; esto es, si y solamente si sus partes positivas y negativas son integrables.

Definición 5.13. Sea $A \in \mathcal{A}$, se denomina integral de f respecto de μ en A y se representa mediante $\int_A f d\mu$ a la expresión

$$\int_A f d\mu = \int_{\Omega} f \mathbb{1}_A d\mu = \int_{\Omega} f^+ \mathbb{1}_A d\mu - \int_{\Omega} f^- \mathbb{1}_A d\mu = \int_A f^+ d\mu - \int_A f^- d\mu$$

Teorema 5.2. Teorema de Fatou-Lebesgue

Sea $\{f_n\}_{n \in \mathbb{N}}$ una sucesión de funciones medibles y h una función medible integrable tal que $|f_n| \leq h$, $\forall n$, entonces

$$\int_{\Omega} \liminf f_n d\mu \leq \liminf \int_{\Omega} f_n d\mu \leq \limsup \int_{\Omega} f_n d\mu \leq \int_{\Omega} \limsup f_n d\mu$$

Teorema 5.3. Teorema de convergencia dominada

Sea $\{f_n\}_{n \in \mathbb{N}}$ una sucesión de funciones medibles y h una función medible integrable tal que $|f_n| \leq h$, $\forall n$. Si $f_n \xrightarrow{n \rightarrow \infty \text{ c.t.p.}} f$, entonces f es integrable y

$$\lim_n \int_{\Omega} f_n d\mu = \int_{\Omega} f d\mu$$

La siguiente desigualdad nos será de utilidad para probar propiedades sobre divergencias que estudiaremos en el siguiente capítulo.

Teorema 5.4. Desigualdad de Jensen

Sea (Ω, \mathcal{A}) un espacio medible y μ una medida cumpliendo que $\mu(\Omega) = 1$. Si f es una función real μ -integrable y ψ una función convexa en el eje real, entonces:

5. Conceptos previos

$$\psi(\int_{\Omega} f d\mu) \leq \int_{\Omega} \psi \circ f d\mu.$$

Demostración. Si ψ una función convexa en el eje real, entonces $\frac{\psi(t)-\psi(t_0)}{t-t_0}$ es no decreciente en $f(\Omega) \setminus t_0$ para cualquier $t_0 \in f(\Omega)$ (t_0 pertenece a la imagen de la función f), esto es, podemos encontrar un valor ϕ tal que

$$\sup_{t < t_0} \frac{\psi(t)-\psi(t_0)}{t-t_0} \leq \phi \leq \inf_{t > t_0} \frac{\psi(t)-\psi(t_0)}{t-t_0}$$

y por tanto, para todo t tenemos $(t - t_0)\phi \leq \psi(t) - \psi(t_0)$. Si establecemos $t = f(x)$ y $t_0 = \int_{\Omega} f d\mu$, sustituyendo tenemos

$$(f(x) - \int_{\Omega} f d\mu)\phi \leq \psi(f(x)) - \psi(\int_{\Omega} f d\mu)$$

Ahora usamos la hipótesis de que $\mu(\Omega) = 1$ y que f es una función real μ -integrable, podemos integrar en ambos lados y obtener

$$(\int_{\Omega} f d\mu - \int_{\Omega} f d\mu)\phi \leq \int_{\Omega} \psi \circ f d\mu - \psi(\int_{\Omega} f d\mu)$$

Finalmente, se tiene

$$\psi(\int_{\Omega} f d\mu) \leq \int_{\Omega} \psi \circ f d\mu$$

□

5.1.3. Medida elemental y medida de Jordan

Anteriormente, se ha introducido el concepto abstracto de medida y espacio de medida, ahora vamos a estudiar dos tipos de medida, la medida elemental y la medida de Jordan. Asimismo, estudiaremos las propiedades que cumplen estas medidas, considerando que heredan todas las propiedades citadas en la sección de medida abstracta.

Nos enfocaremos en definir la medida de conjuntos elementales y en ver cuando un conjunto es medible Jordan y medible Lebesgue.

Por otra parte, cabe resaltar que el concepto de medida de Jordan de un conjunto medible Jordan está estrechamente relacionado con la integral de Riemann. Cuando queremos medir conjuntos que surgen como límites de otros conjuntos necesitamos la medida de Lebesgue que extiende la medida de Jordan. Con la medida de Lebesgue se mantienen casi todas las propiedades de la medida de Jordan, pero con la propiedad adicional de que muchas características de la teoría de Lebesgue se conservan bajo límites.

Definición 5.14. Un **intervalo** es un subconjunto de \mathbb{R} de la forma $[a, b] := \{x \in \mathbb{R} : a \leq x \leq b\}$, $[a, b) := \{x \in \mathbb{R} : a \leq x < b\}$, $(a, b] := \{x \in \mathbb{R} : a < x \leq b\}$, o $(a, b) := \{x \in \mathbb{R} : a < x < b\}$, donde $a \leq b$ son números reales. Definimos la **longitud** $|I|$ de un intervalo $I = [a, b], [a, b), (a, b], (a, b)$ como $|I| := b - a$.

Una **caja** en \mathbb{R}^d es un producto cartesiano $B := I_1 \times \dots \times I_d$ de d intervalos I_1, \dots, I_d , así por ejemplo un intervalo es una caja unidimensional. El **volumen** $|B|$ de la caja B se define como $|B| = |I_1| \times \dots \times |I_d|$.

Un **conjunto elemental** es cualquier subconjunto de \mathbb{R}^d que sea unión de un número finito de cajas.

Lema 5.1. Medida de un conjunto elemental

Sea $E \subset \mathbb{R}^d$ un conjunto elemental.

1. E puede ser expresado como la unión finita de cajas disjuntas
2. Si E es particionado como unión finita de cajas disjuntas $B_1 \cup \dots \cup B_k$, entonces la medida $m(E) := |B_1| + \dots + |B_k|$ es independiente de la partición. En otras palabras, dada otra partición $B'_1 \cup \dots \cup B'_k$, de E , tenemos que $|B_1| + \dots + |B_k| = |B'_1| + \dots + |B'_k|$

Diremos que $m(E)$ es la medida elemental de E .

A continuación, podemos apreciar una serie de propiedades que verifica la medida elemental.

Lema 5.2. Invarianza de la traslación

La propiedad de invarianza en la traslación afirma $m(E + x) = m(E)$ para todo conjunto elemental E y $x \in \mathbb{R}^d$

5.1.3.1. Medida de Jordan

Definición 5.15. Sea $E \subset \mathbb{R}^d$ un conjunto acotado.

- La medida de Jordan interior $m_{*,(J)}(E)$ de E es definida como

$$m_{*,(J)}(E) := \sup_{A \subset E, A \text{ elemental}} m(A)$$

- La medida de Jordan exterior $m^{*,(J)}(E)$ de E es definida como

$$m^{*,(J)}(E) := \inf_{B \supset E, B \text{ elemental}} m(B)$$

- Si $m_{*,(J)}(E) = m^{*,(J)}(E)$ entonces decimos que E es medible Jordan, y llamamos $m(E) := m_{*,(J)}(E) = m^{*,(J)}(E)$ la medida de Jordan de E . (Si queremos enfatizar la dimensión solemos escribir $m^d(E)$ en lugar de $m(E)$)

Lema 5.3. Caracterización de la medibilidad Jordan

Sea $E \subset \mathbb{R}^d$ un conjunto acotado. Son equivalentes:

1. E es medible Jordan
2. Para cada $\epsilon > 0$, existen conjuntos elementales $A \subset E \subset B$ tal que $m(B \setminus A) \leq \epsilon$
3. Para cada $\epsilon > 0$, existe un conjuntos elemental A tal que $m^{*,(J)}(A \Delta E) \leq \epsilon$

Corolario 5.1. Invarianza de traslacion

Sean $E, F \subset \mathbb{R}^d$ conjuntos medibles Jordan. Para cualquier $x \in \mathbb{R}^d$, $E + x$ es medible Jordan y $m(E + x) = m(E)$.

5. Conceptos previos

5.1.4. Medida de Lebesgue

Siempre que tengamos que trabajar con conjuntos medibles Jordan usaríamos la teoría antes descrita. Sin embargo, no todos los conjuntos son medibles, incluso restringiendo a conjuntos acotados nos podemos encontrar con conjuntos que no sean medibles. De hecho, existen conjuntos abiertos acotados o conjuntos compactos que no son medibles Jordan. Otros conjuntos que no son medibles Jordan son las uniones o intersecciones de conjuntos numerables que son medibles Jordan.

Definición 5.16. Un conjunto $E \subset \mathbb{R}^d$ se dice que es medible Lebesgue si, por cada $\epsilon > 0$, existe un abierto $U \subset \mathbb{R}^d$ que contiene a E tal que $m^*(U \setminus E) \leq \epsilon$. Si E es medible Lebesgue, definimos $m(E) := m^*(E)$ como la medida de Lebesgue de E (nótese que esta cantidad puede ser $+\infty$). También escribimos $m(E)$ como $m^d(E)$ cuando deseamos enfatizar la dimensión d .

Hemos comenzado estudiando la medida exterior de Lebesgue m^* que se acaba de definir y toma valores en el eje real no negativo, $[0, +\infty]$. Ahora procederemos a comentar sus propiedades.

Lema 5.4. Aditividad finita para conjuntos separados

Sean $E, F \subset \mathbb{R}^d$ tal que $\text{dist}(E, F) > 0$, donde

$$\text{dist}(E, F) := \inf\{|x - y| : x \in E, y \in F\}$$

es la distancia entre E y F . Entonces $m^*(E \cup F) = m^*(E) + m^*(F)$

Lema 5.5. Medida exterior de los conjuntos elementales

Sea E un conjunto elemental. Entonces la medida exterior de Lebesgue $m^*(E)$ de E es igual a la medida elemental $m(E)$ de E : $m^*(E) = m(E)$

Lema 5.6. Medida exterior de uniones numerables de cajas casi disjuntas

Sea $E = \bigcup_{n=1}^{\infty} B_n$ una unión numerable de cajas casi disjuntas B_1, B_2, \dots . Notamos que dos cajas son casi disjuntas si sus interiores son disjuntos. Entonces

$$m^*(E) = \sum_{n=1}^{\infty} |B_n|$$

Así, por ejemplo, \mathbb{R}^d tiene una medida exterior infinita.

Lema 5.7. Sea $E \subset \mathbb{R}^d$ un conjunto abierto. Entonces E puede expresarse como la unión numerable de cajas casi disjuntas (y, de hecho, podría expresarse como unión numerable de cubos casi disjuntos)

Ahora tenemos una fórmula para la medida exterior de Lebesgue de cualquier conjunto abierto: es exactamente igual a la medida interior de Jordan de ese conjunto, o del volumen total de cualquier partición de ese conjunto de cajas casi disjuntas. Finalmente, aplicando esto tenemos una fórmula para la medida exterior de Lebesgue para un conjunto arbitrario.

Lema 5.8. Regularidad exterior

Sea $E \subset \mathbb{R}^d$ un conjunto arbitrario. Entonces tenemos

$$m^*(E) = \inf_{E \subset U, U \text{ abierto}} m^*(U)$$

Continuamos apreciando la existencia de conjuntos medibles Lebesgue junto con sus propiedades básicas.

Lema 5.9. Existencia de conjuntos medibles de Lebesgue

1. Cada conjunto abierto es medible Lebesgue.
2. Cada conjunto cerrado es medible Lebesgue.
3. Cada conjunto de medida de Lebesgue exterior cero es medible (Estos conjuntos se denominan conjuntos nulos)
4. El conjunto vacío \emptyset es medible Lebesgue.
5. Si $E \subset \mathbb{R}^d$ es medible Lebesgue, entonces también lo es su complementario $\mathbb{R}^d \setminus E$
6. Si $E_1, E_2, E_3, \dots \subset \mathbb{R}^d$ son una secuencia de conjuntos medibles Lebesgue, entonces la unión $\cup_{n=1}^{\infty} E_n$ es medible Lebesgue.
7. Si $E_1, E_2, E_3, \dots \subset \mathbb{R}^d$ son una secuencia de conjuntos medibles Lebesgue, entonces la intersección $\cap_{n=1}^{\infty} E_n$ es medible Lebesgue.

Corolario 5.2. Criterio de medibilidad

Sea $E \subset \mathbb{R}^d$. Son equivalentes:

1. E es medible Lebesgue
2. **Aproximación externa por abierto**
Por cada $\epsilon > 0$, uno puede contener E en un conjunto abierto U con $m^*(U \setminus E) \leq \epsilon$
3. **Casi abierto**
Por cada $\epsilon > 0$, uno puede encontrar un conjunto abierto U tal que $m^*(U \Delta E) \leq \epsilon$
4. **Aproximación interna por cerrados**
Por cada $\epsilon > 0$, uno puede encontrar un conjunto cerrado F contenido en E con $m^*(E \setminus F) \leq \epsilon$
5. **Casi cerrado**
Por cada $\epsilon > 0$, uno puede encontrar un conjunto cerrado F tal que $m^*(F \Delta E) \leq \epsilon$
6. **Casi medible**
Por cada $\epsilon > 0$, uno puede encontrar un conjunto medible E_ϵ tal que $m^*(E_\epsilon \Delta E) \leq \epsilon$

Lema 5.10. Cada conjunto medible Jordan es medible Lebesgue.

Acabamos de definir la medida de Lebesgue y por tanto, podemos construir la integral asociada a la medida de Lebesgue siguiendo las indicaciones de la sección 5.1.2.

Cuando exista la integral de f asociada a una medida μ se denotará por $\int_{\Omega} f d\mu$ ó $\int_{\Omega} f$. También puede notarse como $\int_{\Omega} f(x) d\mu(x)$, especificando la variable x sobre la que se evalúa la función.

5.1.5. Teorema Radon Nikodyn

Definición 5.17. Sea μ y ψ dos funciones de conjunto definidas sobre el espacio medible (Ω, \mathcal{A}) . Se dice que ψ es **absolutamente continua** respecto de μ , y se denota $\psi \ll \mu$, si y solo si, $\forall A \in \mathcal{A}$ con $\mu(A) = 0$ se tiene que $\psi(A) = 0$.

5. Conceptos previos

Definición 5.18. Sea (Ω, \mathcal{A}) un espacio medible y μ una medida en este espacio. La medida μ es conocida como una **medida σ -finita** si satisface una de estas cuatro condiciones equivalentes:

1. El conjunto Ω puede ser cubierto como máximo por una cantidad numerable de muchos conjuntos medibles con medida finita. Esto significa que existen conjuntos $A_1, A_2, \dots \in \mathcal{A}$ con $\mu(A_n) < \infty \forall n \in \mathbb{N}$ que satisfacen que $\cup_{n \in \mathbb{N}} A_n = \Omega$
2. El conjunto Ω puede ser cubierto como máximo por una cantidad numerable de muchos conjuntos disjuntos medibles con medida finita. Esto significa que existen conjuntos $B_1, B_2, \dots \in \mathcal{A}$ con $\mu(B_n) < \infty \forall n \in \mathbb{N}$ y $B_i \cap B_j = \emptyset$ para $i \neq j$ que satisfacen $\cup_{n \in \mathbb{N}} B_n = \Omega$
3. El conjunto Ω puede ser cubierto con una secuencia monótona de conjuntos medibles con medida finita. Esto significa que hay conjuntos $C_1, C_2, \dots \in \mathcal{A}$ con $C_1 \subset C_2 \subset \dots$ y $\mu(C_n) < \infty \forall n \in \mathbb{N}$ que satisfacen $\cup_{n \in \mathbb{N}} C_n = \Omega$
4. Existe una función medible estrictamente positiva f cuya integral es finita. Esto significa que $f(x) > 0 \forall x \in X$ y $\int f d\mu < \infty$

Si μ es una medida σ -finita, el espacio de medida $(\Omega, \mathcal{A}, \mu)$ es llamado **espacio medible σ -finito**.

La siguiente observación se obtiene aplicando 2 y considerando los intervalos $[k, k+1)$ para todos los enteros k , tenemos innumerables intervalos de medida 1 donde su unión es la recta real.

Observación 5.1. La medida de Lebesgue es σ -finita.

Definición 5.19. Sea $f : (\Omega, \mathcal{A}, \mu) \rightarrow (\mathbb{R}, \mathcal{B})$ una función medible cuya integral existe. La **integral indefinida** de f es una función de conjunto ϕ definida sobre la σ -álgebra \mathcal{A} por:

$$\phi(A) = \int_A f d\mu$$

Claramente, la integral indefinida ϕ verifica las siguientes propiedades:

1. ϕ es σ -aditiva
2. ϕ es absolutamente continua respecto de μ ($\phi \ll \mu$)
3. f es integrable si y sólo si ϕ es finita.

Teorema 5.5. Teorema de Radon-Nikodym Sea μ una medida σ -finita sobre (Ω, \mathcal{A}) y ϕ una función de conjunto definida sobre \mathcal{A} , σ -aditiva, σ -finita y verificando que $\phi \ll \mu$, entonces existe una función medible e integrable f , única c.t.p. y finita c.t.p., tal que

$$\forall A \in \mathcal{A}, \phi(A) = \int_A f d\mu$$

5.1.6. Medida de probabilidad

Finalmente, definimos la medida de probabilidad que es la medida finita que usaremos en todo nuestro estudio de los fundamentos de las GANs.

Definición 5.20. Un **espacio de probabilidad** es un espacio medible (Ω, \mathcal{A}, P) con medida total 1: $P(\Omega) = 1$. La medida P es conocida como medida de probabilidad.

5.1.6.1. Ejemplo: medida normalizada

Dado cualquier espacio medible (X, \mathcal{B}, μ) con $0 < \mu(x) < +\infty$, entonces el espacio $(X, \mathcal{B}, \frac{1}{\mu(X)}\mu)$ es un espacio de probabilidad.

5.2. Probabilidad

Inicialmente, presentamos los conceptos de variable aleatoria y de vector aleatorio. Enfocaremos todo nuestro estudio de probabilidad para el caso continuo, detallando propiedades del cambio de variable y de la esperanza matemática. Nótese que en muchas ocasiones aparecerán determinadas referencias a conceptos de teoría de la medida que se han estudiado previamente. Así conseguimos unificar todos los conceptos previos necesarios para el siguiente capítulo.

5.2.1. Variable aleatoria y vector aleatorio

En primer lugar vamos a enunciar la definición de variable aleatoria y de vector aleatorio, o también conocido como variable aleatoria multidimensional, así como las distribuciones de probabilidad de los mismos.

Definición 5.21. Una **variable aleatoria** sobre un espacio probabilístico base ([Def. 5.20](#)) (Ω, \mathcal{A}, P) es una función medible unidimensional ([Def. 5.6](#)), definida sobre un espacio probabilístico, esto es, $\mathcal{X} : (\Omega, \mathcal{A}, P) \rightarrow (\mathbb{R}, \mathcal{B})$ tal que

$$\mathcal{X}^{-1}(B) \in \mathcal{A}, \quad \forall B \in \mathcal{B},$$

donde $\mathcal{X}^{-1}(B) = \{\omega \in \Omega : \mathcal{X}(\omega) \in B\}$, equivalentemente,

$$\mathcal{X}^{-1}((-\infty, x]) \in \mathcal{A}, \quad \forall x \in \mathbb{R},$$

siendo $\mathcal{X}^{-1}((-\infty, x]) = \{\omega \in \Omega : \mathcal{X}(\omega) \leq x\}$. Se denota por \mathcal{B} a la σ -álgebra de Borel ([Def. 5.4](#)), que es la mínima σ -álgebra sobre \mathbb{R} que contiene a todos los intervalos.

Definición 5.22. Distribución de Probabilidad de una Variable Aleatoria unidimensional
Si \mathcal{X} es una variable aleatoria sobre (Ω, \mathcal{A}, P) , se define su distribución de probabilidad como una función $P_{\mathcal{X}} : \mathcal{B} \rightarrow [0, 1]$, definida por

$$P_{\mathcal{X}}(B) := P(\{\omega \in \Omega : \mathcal{X}(\omega) \in B\}), \quad \forall B \in \mathcal{B}$$

Nótese que podemos abreviar $\{\omega \in \Omega : \mathcal{X}(\omega) \in B\}$ como $\{\mathcal{X} \in B\}$ y definir $P_{\mathcal{X}}(B) = P(\mathcal{X} \in B)$

Definición 5.23. Función de distribución de una Variable Aleatoria unidimensional
Es una función $F_{\mathcal{X}} : \mathbb{R} \rightarrow [0, 1]$, definida por

$$F_{\mathcal{X}}(x) = P_{\mathcal{X}}((-\infty, x]) = P(\mathcal{X} \in (-\infty, x]), \quad \forall x \in \mathbb{R}$$

y satisface las siguientes propiedades:

- Monótona no decreciente
- Continua a la derecha

5. Conceptos previos

- $\exists \lim_{x \rightarrow -\infty} F_{\mathcal{X}}(x) = 0$ y $\lim_{x \rightarrow \infty} F_{\mathcal{X}}(x) = 1$

Definición 5.24. Un **vector aleatorio**, también conocido como variable aleatoria multidimensional, es una función medible multidimensional (Def. 5.6), definida sobre un espacio probabilístico (Def. 5.20); esto es, una función $\mathcal{X} = (\mathcal{X}_1, \dots, \mathcal{X}_n)^T : (\Omega, \mathcal{A}, P) \rightarrow (\mathbb{R}^n, \mathcal{B}^n)$ tal que

$$\mathcal{X}^{-1}(B) \in \mathcal{A}, \quad \forall B \in \mathcal{B}^n,$$

donde $\mathcal{X}^{-1}(B) = \{\omega \in \Omega : \mathcal{X}(\omega) \in B\}$, equivalentemente,

$$\mathcal{X}^{-1}((-\infty, x_1] \times \dots \times (-\infty, x_n]) = \{\omega \in \Omega : \mathcal{X}_1(\omega) \leq x_1, \dots, \mathcal{X}_n(\omega) \leq x_n\} \in \mathcal{A}, \quad \forall x = (x_1, \dots, x_n) \in \mathbb{R}^n$$

Teorema 5.6. Teorema de medibilidad

Sea (Ω, \mathcal{A}, P) un espacio probabilístico (Def. 5.20) y $\mathcal{X} = (\mathcal{X}_1, \dots, \mathcal{X}_n)^T : \Omega \rightarrow \mathbb{R}^n$ una función n -dimensional

$$\mathcal{X} \text{ es un vector aleatorio} \iff \mathcal{X}_1, \dots, \mathcal{X}_n \text{ son variables aleatorias}$$

Definición 5.25. Distribución de Probabilidad de una Variable Aleatoria multidimensional o vector aleatorio Si \mathcal{X} es un vector aleatorio sobre (Ω, \mathcal{A}, P) , se define su distribución de probabilidad como una función $P_{\mathcal{X}} : \mathcal{B}^n \rightarrow [0, 1]$, definida por

$$P_{\mathcal{X}}(B) = P(\{\omega \in \Omega : \mathcal{X}(\omega) \in B\}), \quad \forall B \in \mathcal{B}^n$$

Nótese que podemos abreviar $\{\omega \in \Omega : \mathcal{X}(\omega) \in B\}$ como $\{\mathcal{X} \in B\}$ y definir $P_{\mathcal{X}}(B) = P(\mathcal{X} \in B)$

Definición 5.26. Función de distribución de una Variable Aleatoria multidimensional o vector aleatorio

Es una función $F_{\mathcal{X}} : \mathbb{R}^n \rightarrow [0, 1]$, definida por

$$F_{\mathcal{X}}(x) = P_{\mathcal{X}}((-\infty, x]) = P(\mathcal{X} \leq x), \quad \forall x \in \mathbb{R}^n$$

Esto es, si $\mathcal{X} = (X_1, \dots, X_n)$

$$F_{\mathcal{X}}(x_1, \dots, x_n) = P_{\mathcal{X}}((-\infty, x_1] \times \dots \times (-\infty, x_n]) = P(\mathcal{X}_1 \leq x_1, \dots, \mathcal{X}_n \leq x_n), \quad \forall x_1, \dots, x_n \in \mathbb{R}$$

y satisface las siguientes propiedades:

- Monótona no decreciente en cada argumento
- Continua a la derecha en cada argumento
- $\forall i = 1, \dots, n \quad \forall x_1, \dots, x_n \in \mathbb{R}$

$$\begin{aligned} \exists \lim_{x_i \rightarrow -\infty} F_{\mathcal{X}}(x_1, \dots, x_{i-1}, x_i, x_{i+1}, \dots, x_n) &\equiv F_{\mathcal{X}}(x_1, \dots, x_{i-1}, -\infty, x_{i+1}, \dots, x_n) = 0 \text{ y} \\ \exists \lim_{x_i \rightarrow +\infty, i=1, \dots, n} F_{\mathcal{X}}(x_1, \dots, x_n) &\equiv F_{\mathcal{X}}(+\infty, \dots, +\infty) = 1 \end{aligned}$$

- $\forall x_1, \dots, x_n \in \mathbb{R}, \forall i = 1, \dots, n,$

$$\lim_{\epsilon \rightarrow 0, \epsilon > 0} F_{\mathcal{X}}(x_1, \dots, x_i - \epsilon, \dots, x_n) = P(\mathcal{X}_1 \leq x_1, \dots, \mathcal{X}_i < x_i, \dots, \mathcal{X}_n \leq x_n) = F_{\mathcal{X}}(x_1, \dots, x_i^-, \dots, x_n)$$

$$\begin{aligned} P(\mathcal{X}_1 \leq x_1, \dots, \mathcal{X}_{i-1} \leq x_{i-1}, \mathcal{X}_i = x_i, \mathcal{X}_{i+1} \leq x_{i+1}, \dots, \mathcal{X}_n \leq x_n) &= F_{\mathcal{X}}(x_1, \dots, x_i, \dots, x_n) - \\ &F_{\mathcal{X}}(x_1, \dots, x_i^-, \dots, x_n) \end{aligned}$$

- $\forall x_1, \dots, x_n \in \mathbb{R}$, y para cualquier $\epsilon_1, \dots, \epsilon_n \in \mathbb{R}_+$:

$$\begin{aligned} & F_{\mathcal{X}}(x_1 + \epsilon_1, \dots, x_n + \epsilon_n) - \sum_{i=1}^n F_{\mathcal{X}}(x_1 + \epsilon_1, \dots, x_{i-1} + \epsilon_{i-1}, x_i, x_{i+1} + \epsilon_{i+1}, \dots, x_n + \epsilon_n) \\ & + \sum_{i=1}^n \sum_{j=1}^n F_{\mathcal{X}}(x_1 + \epsilon_1, \dots, x_{i-1} + \epsilon_{i-1}, x_i, x_{i+1} + \epsilon_{i+1}, \dots, x_{j-1} + \epsilon_{j-1}, x_j, x_{j+1} + \epsilon_{j+1}, \dots, x_n + \epsilon_n) \\ & - \dots + (-1)^n F_{\mathcal{X}}(x_1, \dots, x_n) \geq 0 \end{aligned}$$

5.2.2. Variables aleatorias continuas

En todo nuestro estudio matemático nos centraremos en variables aleatorias continuas o en vectores aleatorios continuos, procedemos a explicar su definición junto con sus propiedades.

El teorema de Radon-Nikodym ([Teorema 5.5](#)) es un resultado que hemos visto en la sección de teoría de la medida donde se expresa la relación entre dos medidas definidas en un mismo espacio medible ([Def. 5.2](#)).

Sabemos que en el espacio medible $(\mathbb{R}, \mathcal{B})$ tenemos la medida de Lebesgue ([Subsección 5.1.4](#)) que notaremos como μ y queremos definir una nueva medida $P_{\mathcal{X}}$. Una forma de obtener una nueva medida a partir de una ya dada es asignar una densidad $f_{\mathcal{X}}(x)$ a cada punto x del espacio \mathbb{R} y luego calcular la integral asociada a la medida de Lebesgue de $f_{\mathcal{X}}$ sobre un subconjunto medible de $(\mathbb{R}, \mathcal{B})$, esto es, calcular la integral de Lebesgue ([5.1.4](#)) de $f_{\mathcal{X}}$ sobre intervalos $[a, b]$ con $a, b \in \mathbb{R}$. Entonces, tendríamos $P_{\mathcal{X}}([a, b]) = \int_a^b f_{\mathcal{X}} d\mu$ ó $P_{\mathcal{X}}([a, b]) = \int_a^b f_{\mathcal{X}}$ (Notar que aunque no se especifique la medida tenemos que resaltar que se está calculando la integral de Lebesgue). Básicamente, esta idea es aplicar el teorema de Radon-Nikodym ya que satisface que, bajo ciertas condiciones, cualquier medida puede expresarse de esta manera con respecto a otra medida μ en el mismo espacio. Por consiguiente, se determina la función de densidad de probabilidad de una variable aleatoria.

Análogamente aplicando esta idea al espacio medible $(\mathbb{R}^n, \mathcal{B}^n)$ tenemos la existencia de la función de densidad de un vector aleatorio y la medida $P_{\mathcal{X}}$.

Definición 5.27. Variable aleatorias continuas

Una variable aleatoria $\mathcal{X} : (\Omega, \mathcal{A}, P) \rightarrow (\mathbb{R}, \mathcal{B}, P_{\mathcal{X}})$, es una variable aleatoria continua si existe una función $f_{\mathcal{X}} : \mathbb{R} \rightarrow \mathbb{R}$ tal que la distribución de probabilidad se puede expresar como la siguiente integral de Lebesgue:

$$P_{\mathcal{X}}([a, b]) = \int_a^b f_{\mathcal{X}}, \quad \forall [a, b] \in \mathcal{B}.$$

$f_{\mathcal{X}}$ recibe el nombre de función de densidad de \mathcal{X} y satisface:

1. Es no negativa
2. Es integrable y $\int_{\mathbb{R}} f_{\mathcal{X}} = 1$

Cualquier función $f : \mathbb{R} \rightarrow \mathbb{R}$ satisfaciendo 1. y 2. es la función de densidad de una variable aleatoria continua. Así, se caracteriza a la función de densidad de una variable aleatoria.

Acabamos de explicar como enlazar el concepto de variable aleatoria continua y el teorema de Radon-Nikodym con el objetivo de tener la existencia de la función de densidad $f_{\mathcal{X}}$

5. Conceptos previos

cumpliendo que es medible e integrable, única c.t.p. y finita c.t.p. Pero nos falta ver que se cumplen las condiciones del teorema para poder aplicarlo. Obsérvese que vamos a estudiarlo para el caso general donde el espacio medible es $(\mathbb{R}^n, \mathcal{B}^n)$ con $n \in \mathbb{N}$.

Inicialmente, sabemos que la medida de Lebesgue es σ -finita ([Observación 5.1](#)) sobre $(\mathbb{R}^n, \mathcal{B}^n)$ para $n = 1$ ó $n > 1$ y $P_{\mathcal{X}}$ es una función de conjunto definida sobre \mathcal{B}^n .

En primer lugar tenemos que ver que $P_{\mathcal{X}}$ es una medida σ -finita ([Def. 5.18](#)) sobre $(\mathbb{R}^n, \mathcal{B}^n)$. Como sabemos que \mathbb{R}^n puede ser cubierto como máximo por una cantidad numerable de muchos conjuntos medibles, $A_1, A_2 \dots \in \mathcal{B}^n$, con medida finita, $P_{\mathcal{X}}(A_n) < \infty \forall n \in \mathbb{N}$, entonces $\cup_{n \in \mathbb{N}} A_n = \mathbb{R}^n$ y afirmamos que la medida de probabilidad $P_{\mathcal{X}}$ es σ -finita.

Luego, tenemos que probar que $P_{\mathcal{X}}$ es una medida σ -aditiva que se obtiene directamente de la propiedad de la aditividad numerable de la medida ([Def. 5.2](#)).

Claramente, se tiene que $P_{\mathcal{X}}$ es absolutamente continua ([Def. 5.17](#)) respecto de la medida de Lebesgue, $P_{\mathcal{X}} \ll \mu$, ya que para todo conjunto borel cuya medida de Lebesgue sea nula tenemos que $P_{\mathcal{X}}$ se anula en dicho conjunto.

Finalmente, aplicando estos razonamientos tenemos todas las hipótesis necesarias para aplicar el Teorema de Radon-Nikodym y obtener la existencia de la función de densidad $f_{\mathcal{X}}$ cumpliendo que es medible e integrable, única c.t.p. y finita c.t.p.

Proposición 5.4. *La función de densidad presenta las siguientes propiedades:*

- $\lim_{x \rightarrow -\infty} f_{\mathcal{X}}(x) = \lim_{x \rightarrow \infty} f_{\mathcal{X}}(x) = 0$
- $f_{\mathcal{X}}$ es continua salvo a lo sumo en un conjunto numerable de puntos(o conjunto de medida nula)
- En los puntos de continuidad de $f_{\mathcal{X}}$, se tiene que $F_{\mathcal{X}}$ es derivable y satisface

$$\frac{dF_{\mathcal{X}}(x)}{dx} = f_{\mathcal{X}}(x)$$

- $f_{\mathcal{X}}$ se puede modificar en un conjunto numerable de puntos sin afectar a $F_{\mathcal{X}}$

Definición 5.28. Vector aleatorio continuo

Un vector aleatorio $\mathcal{X} : (\Omega, \mathcal{A}, P) \rightarrow (\mathbb{R}^n, \mathcal{B}^n)$ es de tipo continuo si existe un función $f_{\mathcal{X}} : \mathbb{R}^n \rightarrow \mathbb{R}$, tal que la distribución de probabilidad de \mathcal{X} se puede expresar como la siguiente integral de Lebesgue:

$$P_{\mathcal{X}}([a_1, b_1] \times \dots \times [a_n, b_n]) = \int_{a_1}^{b_1} \dots \int_{a_n}^{b_n} f_{\mathcal{X}}, \quad \forall [a_1, b_1] \times \dots \times [a_n, b_n] \in \mathcal{B}^n$$

La función $f_{\mathcal{X}}$ recibe el nombre de función de densidad de probabilidad . Como consecuencia se tiene que la función de distribución $F_{\mathcal{X}}$ es continua sobre \mathbb{R}^n , derivable salvo un conjunto de medida nula de \mathbb{R}^n , con derivada continua sobre el dominio donde se define $F_{\mathcal{X}}$.

La función de densidad $f_{\mathcal{X}}$ satisface las siguientes propiedades , de hecho cualquier función $f : \mathbb{R}^n \rightarrow \mathbb{R}$ satisfaciendo 1. y 2. es la función de densidad de una variable aleatoria continua. Así, se caracteriza a la función de densidad de un vector aleatorio.

1. Es no negativa
2. Es integrable y $\int_{-\infty}^{\infty} \dots \int_{-\infty}^{\infty} f_{\mathcal{X}} = 1$

Proposición 5.5. La función de densidad presenta las siguientes propiedades:

- $\lim_{x_i \rightarrow -\infty, i=1,\dots,n} f_{\mathcal{X}}(x_1, \dots, x_n) = \lim_{x_i \rightarrow \infty, i=1,\dots,n} f_{\mathcal{X}}(x_1, \dots, x_n) = 0$
- El conjunto de discontinuidades de $f_{\mathcal{X}}$ en \mathbb{R}^n es numerable, es decir, tiene medida nula.
- Si $x = (x_1, \dots, x_n)$ es un punto de continuidad de $f_{\mathcal{X}}$, entonces

$$\exists \frac{d^n F_{\mathcal{X}}(x_1, \dots, x_n)}{dx_1 \dots dx_n} = f_{\mathcal{X}}(x_1, \dots, x_n)$$

- Los valores de $f_{\mathcal{X}}$ pueden modificarse en un conjunto de medida nula sin afectar a $F_{\mathcal{X}}$ (como primitiva de $f_{\mathcal{X}}$ en sus puntos de continuidad)
- Puesto que $f_{\mathcal{X}}$ determina a $P_{\mathcal{X}}$, también determina a la función de distribución $F_{\mathcal{X}}$. Es decir,

$$F_{\mathcal{X}}(x) = P_{\mathcal{X}}((-\infty, x]), \quad \forall x \in \mathbb{R}^n$$

5.2.3. Cambio de variable multidimensional

Vamos a ver ahora como se realizaría un cambio de variable multidimensional, para el caso unidimensional se realizaría de forma análoga con $m = 1$.

Sea $\mathcal{X} = (\mathcal{X}_1, \dots, \mathcal{X}_n)$ un vector aleatorio sobre (Ω, \mathcal{A}, P) , tal que

$$P(\mathcal{X} \in E_{\mathcal{X}}) = P_{\mathcal{X}}(E_{\mathcal{X}}) = 1,$$

para un cierto conjunto $E_{\mathcal{X}} \subseteq \mathbb{R}^n$. Se considera una función $g : E_{\mathcal{X}} \rightarrow \mathbb{R}^m$ medible ([Def. 5.6](#)). Entonces, $\mathcal{Y} = g(\mathcal{X})$ es una variable aleatoria m -dimensional sobre (Ω, \mathcal{A}, P) , cuya distribución de probabilidad y función de distribución de probabilidad vienen respectivamente dadas por:

$$\begin{aligned} P_{\mathcal{Y}}(B) &= P_{\mathcal{X}}(g^{-1}(B)) = P(\mathcal{X} \in g^{-1}(B)), \quad \forall B \in \mathcal{B}^m \\ F_{\mathcal{Y}}(y) &= P_{\mathcal{X}}(g^{-1}((-\infty, y_1] \times \dots \times (-\infty, y_m])) \quad \forall y = (y_1, \dots, y_m) \in \mathbb{R}^m \end{aligned}$$

5.2.4. Esperanza matemática

Definición 5.29. Si \mathcal{X} es una variable aleatoria definida en el espacio de probabilidad (Ω, \mathcal{A}, P) , entonces la esperanza de \mathcal{X} , denotada como $\mathbb{E}_P[\mathcal{X}]$, está definido como la integral de Lebesgue ([5.1.4](#))

$$\mathbb{E}_P[\mathcal{X}] = \int_{\Omega} \mathcal{X} dP$$

Observación 5.2. Hay veces que no se especifica la distribución de probabilidad P en la definición de esperanza, notemos que la esperanza de la variable aleatoria \mathcal{X} definida en el espacio probabilístico (Ω, \mathcal{A}, P) puede referenciarse simplemente como $\mathbb{E}[\mathcal{X}]$ en lugar de $\mathbb{E}_P[\mathcal{X}]$.

5. Conceptos previos

Definición 5.30. Sea $\mathcal{X} = (\mathcal{X}_1, \dots, \mathcal{X}_n) : (\Omega, \mathcal{A}, P) \longrightarrow (\mathbb{R}^n, \mathcal{B}^n, P_X)$ un vector aleatorio n-dimensional. La esperanza matemática $\mathbb{E}[\mathcal{X}]$ de \mathcal{X} , si existe, se define como un vector cuyas componentes son las medias o esperanzas matemáticas de sus componentes aleatorias, es decir

$$\mathbb{E}[\mathcal{X}] = (\mathbb{E}[\mathcal{X}_1], \dots, \mathbb{E}[\mathcal{X}_n]) = (\mu_1, \dots, \mu_n) \in \mathbb{R}^n$$

Proposición 5.6. La esperanza matemática de un vector aleatorio $\mathcal{X} = (\mathcal{X}_1, \dots, \mathcal{X}_n)$ existe \iff existen las esperanzas matemáticas de sus componentes aleatorias.

$$\exists \mathbb{E}[\mathcal{X}] \iff \exists \mathbb{E}[\mathcal{X}_i], i = 1, \dots, n.$$

Proposición 5.7. La esperanza matemática cumple las siguientes propiedades:

- **Linealidad**

Para cualesquiera $(a_1, \dots, a_n), (b_1, \dots, b_n) \in \mathbb{R}^n$,

$$\begin{aligned} \exists \mathbb{E}[\mathcal{X}_i] \rightarrow \exists \mathbb{E}[a_i \mathcal{X}_i + b_i], \quad i = 1, \dots, n \\ \downarrow \\ \exists \mathbb{E}[\sum_{i=1}^n a_i \mathcal{X}_i + b_i] = \sum_{i=1}^n a_i \mathbb{E}[\mathcal{X}_i] + b_i \end{aligned}$$

- **Monotonía**

Sean \mathcal{X}_1 y \mathcal{X}_2 variables aleatorias unidimensionales, tales que, $\exists \mathbb{E}[\mathcal{X}_1], \exists \mathbb{E}[\mathcal{X}_2]$, entonces

$$\mathcal{X}_1 \leq \mathcal{X}_2 \rightarrow \mathbb{E}[\mathcal{X}_1] \leq \mathbb{E}[\mathcal{X}_2].$$

A continuación, veamos como definir la esperanza matemática de una variable aleatoria continua. Dada \mathcal{X} una variable aleatoria continua definida sobre (Ω, \mathcal{A}, P) , podemos definir la esperanza como $\mathbb{E}[\mathcal{X}]$ sin especificar la distribución de probabilidad P . Más adelante en nuestro estudio de las GANs, se usará la notación $\mathbb{E}_{\Omega \sim P}[\mathcal{X}]$ para mostrar la dependencia de los datos.

Definición 5.31. Esperanza matemática de una variable aleatoria continua

Si $\exists \int_{\mathbb{R}} |x| f_{\mathcal{X}}(x) dx \leq \infty$, donde $f_{\mathcal{X}}(x)$ es la función de densidad de la variable aleatoria continua \mathcal{X} , entonces la esperanza se define como

$$\mathbb{E}[\mathcal{X}] = \int_{\mathbb{R}} x f_{\mathcal{X}}(x) dx$$

Centrándonos en variables aleatorias continuas, vamos a ver como afecta el cambio de variable al cálculo de la esperanza matemática.

Proposición 5.8. Sea \mathcal{X} una variable aleatoria continua con función de densidad $f_{\mathcal{X}}$.

Entonces existe la esperanza matemática de la variable aleatoria $g(\mathcal{X})$ si y solo si $\int_{\mathbb{R}^n} |g(x_1, \dots, x_n)| f_{\mathcal{X}}(x_1, \dots, x_n) dx_1 \dots dx_n < \infty$. En tal caso se define como

$$\mathbb{E}[g(\mathcal{X})] = \int_{\mathbb{R}^n} g(x_1, \dots, x_n) f_{\mathcal{X}}(x_1, \dots, x_n) dx_1 \dots dx_n$$

6. Introducción matemática para la formulación de las GANs

Para comprender mejor las GANs, debemos de comprender la base matemática que hay detrás y para ello matematizamos nuestros objetivos usando el enfoque básico de las GANs. Estudiamos los distintos teoremas que proporcionan información relevante sobre el problema min-max junto con el algoritmo de las GANs. La información de este capítulo se ha estudiado en los siguientes artículos [Wan20] y [GPAM⁺14].

6.1. Comprensión del problema matemático

Las GANs aparecieron como un nuevo modelo generativo cuyo objetivo era resolver la siguiente cuestión: tenemos un conjunto de datos de objetos con cierto grado de consistencia, por ejemplo, una colección de imágenes de perros o dígitos escritos a mano, o pinturas de Leonardo da Vinci, etc., ¿podemos generar artificialmente objetos similares?

En esta sección nos encargamos de formular esta pregunta y de responderla en base a conceptos matemáticos. Primero vamos a explicar que se entiende por "objetos con cierto grado de consistencia" u "objetos similares" antes de continuar el estudio. Debemos de suponer que nuestros objetos son puntos de \mathbb{R}^n , el conjunto de objetos lo denotamos como $X \subset \mathbb{R}^n$.

Definimos las distribuciones de probabilidad (Def. 5.25) $P_{\mathcal{X}} : \mathcal{B}^n \rightarrow [0, 1]$ y $P_{\mathcal{Y}} : \mathcal{B}^n \rightarrow [0, 1]$ cuyas funciones de densidad (Def. 5.28) son $f_{\mathcal{X}} : \mathbb{R}^n \rightarrow [0, 1]$ y $f_{\mathcal{Y}} : \mathbb{R}^n \rightarrow [0, 1]$ para las variables aleatorias multidimensionales \mathcal{X} e \mathcal{Y} .

- Si decimos que los objetos en el conjunto de datos X tienen **cierto grado de consistencia**, queremos decir que son muestras generadas a partir de una misma distribución de probabilidad $P_{\mathcal{X}}$ en \mathbb{R}^n , donde $f_{\mathcal{X}}$ es su función de densidad.
Al suponer que $P_{\mathcal{X}}$ tiene una función de densidad, entonces estamos afirmando que $P_{\mathcal{X}}$ es absolutamente continua.
- Explicar que se entiende por **objetos similares** es un poco más complicado ya que hay muchas formas de cuantificar la similitud. Por ejemplo, podemos definir una función distancia y así decir que dos objetos son similares si la distancia entre ellos es pequeña. Pero esta idea no es útil aquí ya que nuestro objetivo no es generar nuevos objetos que tengan distancias pequeñas a algunos objetos del conjunto X . Más bien, queremos generar nuevos objetos que aunque no estén tan cerca de los objetos del conjunto de entrenamiento pero si pertenecerán a la misma clase, por ejemplo que sean pinturas de Leonardo da Vinci.

Un mejor enfoque es definir la similitud entre distribuciones de probabilidad. Es decir, tenemos nuestro conjunto de datos de entrenamiento $X \subset \mathbb{R}^n$ que está formado a partir de muestras de una distribución de probabilidad $P_{\mathcal{X}}$, con función de densidad $f_{\mathcal{X}}$, y

6. Introducción matemática para la formulación de las GANs

nos gustaría encontrar una distribución de probabilidad P_Y , con función de densidad f_Y , tal que P_Y sea una buena aproximación de P_X . Por tanto, dos conjuntos de datos son similares si son ejemplos de dos distribuciones de probabilidad aproximadas(o de una misma distribución de probabilidad).

Después de haber matematizado nuestro objetivo, podemos preguntarnos por qué no definimos $P_X = P_Y$ y tomamos muestras de P_X para generar objetos similares.

El problema es que desconocemos P_X y solo conocemos un conjunto finito de muestras X . Por tanto, nuestro estudio se centrará en aprender la distribución P_X solo a partir de su conjunto finito de muestras y luego encontrar P_Y como proceso de aproximación de P_X .

6.2. Enfoque básico de las redes generativas adversarias

Inicialmente, se comienza con una distribución de probabilidad inicial P_Z definida en \mathbb{R}^d con objeto de conseguir aproximar P_X . Establecemos que P_Z sea la distribución normal $N(0, I_d)$, la variable aleatoria Z sigue una distribución normal, aunque podría haber sido cualquier otra distribución.

El enfoque básico de las GANs es encontrar una función $G : \mathbb{R}^d \rightarrow \mathbb{R}^n$ tal que si una variable aleatoria $z \in \mathbb{R}^d$ tiene una distribución de probabilidad P_Z entonces la variable $G(z)$ debe tener una distribución P_X . Claramente, la distribución de $G(z)$ es $P_Z \circ G^{-1}$. Por tanto, queremos encontrar una función $G(z)$ tal que $P_Z \circ G^{-1} = P_X$ ó $P_Z \circ G^{-1}$ ser una buena aproximación de P_X .

Partiendo de que solo conocemos muestras de P_X , si llegamos a determinar G tendremos muestras $G(z)$ donde z se extrae de la distribución P_Z . Pero aparece el problema de como determinar a partir de estas muestras que nuestra distribución $P_Z \circ G^{-1}$ es la misma o una buena aproximación de P_X .

La técnica de las GANs es introducir una función discriminadora $D(x)$ que intenta descartar las muestras generadas por G y decir que son muestras falsas. $D(x)$ es un clasificador que trata de distinguir las muestras del conjunto de entrenamiento X de las muestras generadas $G(z)$. Al principio la función discriminadora D sabe perfectamente distinguir las muestras pero más adelante cuando G aprenda a generar muestras más similares a X , G estará aprendiendo y le tocará al discriminador mejorar para ir distinguiendo correctamente. A través de este proceso de entrenamiento se alcanza un punto de equilibrio donde las muestras generadas deberán tener una distribución muy similar a las muestras de entrenamiento X .

Para relacionar todo el aspecto matemático con el informático nos preguntamos donde aparecen las redes neuronales y el aprendizaje profundo en todo esto. La respuesta nos la dan los teoremas de Aproximación Universal ya que afirman que las redes neuronales profundas se pueden usar para aproximar casi cualquier función , mediante el aprendizaje de los parámetros de la red utilizando los conjuntos de entrenamiento.

En definitiva modelaremos tanto la función discriminadora D como la función generadora G como redes neuronales con parámetros ω y θ . Por tanto, escribiremos $D(x)$ como $D_\omega(x)$ y

$G(z)$ como $G_\theta(z)$ y denotaremos $P_{\mathcal{Y}_\theta} := P_{\mathcal{Z}} \circ G_\theta^{-1}$. Nuestro objetivo es encontrar la función $G_\theta(z)$ ajustando el valor de θ mediante el entrenamiento de la red.

6.3. Problema min-max

El problema min-max surge de la teoría de juegos donde intervienen dos jugadores que compiten entre si. Minmax es un método de decisión para minimizar la pérdida máxima esperada en juegos con adversarios y con información perfecta. Su funcionamiento consiste en elegir el mejor movimiento para ti suponiendo que tu rival escogerá el peor movimiento para ti. Esto es, se intenta minimizar la ganancia del rival ya que queremos que obtenga el peor resultado. En términos matemáticos, dada una función evaluadora, el jugador intenta maximizar su valor mientras que el adversario intenta minimizarlo.

Vamos a estudiar una serie de definiciones y teoremas en \mathbb{R}^2 pero pueden extraerse a \mathbb{R}^n .

Definición 6.1. Un punto de silla (\bar{x}, \bar{y}) de una función g es un punto crítico que no es extremo local de la función. Esto quiere decir que para cualquier entorno del punto (\bar{x}, \bar{y}) existen puntos (x_1, y_1) y (x_2, y_2) del entorno verificando

$$g(x_1, y_1) \leq g(\bar{x}, \bar{y}) \leq g(x_2, y_2)$$

Definición 6.2. Un punto minmax de una función $g : \mathbb{R}^2 \rightarrow \mathbb{R}$ es un punto $(\bar{x}, \bar{y}) \in \mathbb{R}^2$ que verifica la siguiente igualdad:

$$g(\bar{x}, \bar{y}) = \min_x \max_y g(x, y) = \max_y \min_x g(x, y)$$

Teorema 6.1. Si una función $g : \mathbb{R}^2 \rightarrow \mathbb{R}$ tiene un punto minmax $(\bar{x}, \bar{y}) \in \mathbb{R}^2$, esto es

$$g(\bar{x}, \bar{y}) = \min_x \max_y g(x, y) = \max_y \min_x g(x, y)$$

entonces la función g tiene un punto de silla con respecto a \mathbb{R}^2 y (\bar{x}, \bar{y}) es el punto de silla.

Demostración. De la hipótesis del teorema tenemos

$$\min_x g(x, \bar{y}) = g(\bar{x}, \bar{y}) = \max_y g(\bar{x}, y)$$

Consecuentemente, para cualquier $(x, y) \in \mathbb{R}^2$ que se encuentre en un entorno local de (\bar{x}, \bar{y}) tenemos

$$g(x, \bar{y}) \geq g(\bar{x}, \bar{y}) \geq g(\bar{x}, y)$$

Veamos que (\bar{x}, \bar{y}) es un punto de crítico de g . Por un lado tenemos que el punto (\bar{x}, \bar{y}) alcanza el mínimo local en g con respecto a la variable x , esto es, $\min_x g(x, \bar{y}) = g(\bar{x}, \bar{y})$. Entonces, tenemos que $\frac{\partial g}{\partial x}(\bar{x}, \bar{y}) = 0$. Por otro lado, como el punto (\bar{x}, \bar{y}) es un máximo local de la función g con respecto a la variable y , entonces $\frac{\partial g}{\partial y}(\bar{x}, \bar{y}) = 0$. Finalmente, se obtiene que es un punto crítico ya que

$$\nabla g(\bar{x}, \bar{y}) = \left(\frac{\partial g}{\partial x}, \frac{\partial g}{\partial y} \right)(\bar{x}, \bar{y}) = (0, 0)$$

lo que implica que (\bar{x}, \bar{y}) es un punto de silla. □

6. Introducción matemática para la formulación de las GANs

6.3.1. Ejemplo de problema min-max

Sea $f : \mathbb{R}^2 \rightarrow \mathbb{R}$ definida como $f(x, y) = xy$ y vamos a hallar $\min_y \max_x f(x, y)$. En este ejemplo un jugador tiene que minimizar la función de costo xy , mientras que el otro jugador trata de minimizar la función de costo $-xy$ (ya que hacer esto equivale a maximizar la función xy).

Si modelamos a cada jugador haciendo pasos muy pequeños del gradiente con el algoritmo del descenso del gradiente ([Subsección 1.2.3](#)) con objeto de reducir su propio coste sin tener en cuenta el otro jugador, llegamos a una órbita circular estable en lugar de al punto de equilibrio $(0, 0)$. Esto se debe a que estaríamos modelando de forma independiente a cada jugador.

Por esta razón, tenemos que tener muy en cuenta que el equilibrio en un juego minmax es el punto que es mínimo para las funciones de costes de ambos jugadores. Esto es, es el punto de silla de f , ya se corresponde con el mínimo local con respecto a los parámetros del primer jugador y con el máximo local con respecto a los parámetros del segundo jugador. Para alcanzar el punto de equilibrio ambos jugadores se irán turnando para ir aumentando e ir reduciendo f hasta aterrizar en el origen. A continuación, vemos como se representa este ejemplo extraído del libro de [[Goo16](#)] haciendo uso de la herramienta de Geogebra.

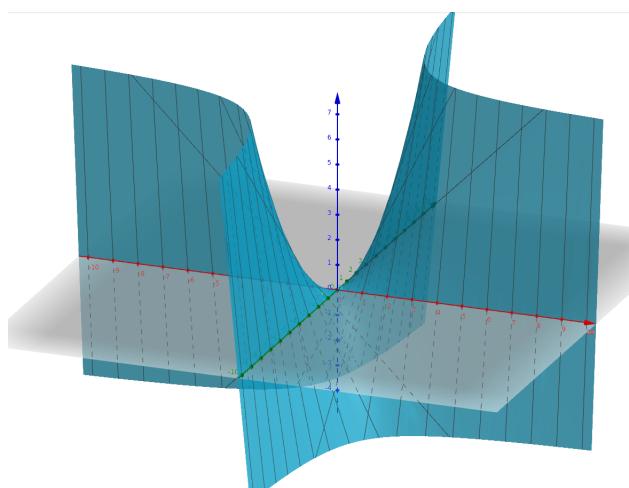


Figura 6.1.: Ejemplo MinMax f

Para determinar que el punto $(0, 0)$ es un punto de silla podemos usar [Teorema 6.1](#). Veamos que el punto $(0, 0)$ es un punto minmax, esto es, que cumple

$$f(0, 0) = \min_x \max_y f(x, y) = \max_y \min_x f(x, y)$$

Para obtener $\min_x f(x, y)$ calculamos $\frac{\partial f(x, y)}{\partial x} = y$, tenemos que $\frac{\partial f(x, y)}{\partial x} = 0$, si y solo si, $y = 0$. Asimismo, para $\max_y f(x, y)$ calculamos $\frac{\partial f(x, y)}{\partial y} = x$, tenemos que $\frac{\partial f(x, y)}{\partial y} = 0$, si y solo si, $x = 0$. Finalmente, el punto minmax es $(0, 0)$ y aplicando el teorema tenemos que es el punto de silla de f .

6.4. Formulación min-max del enfoque básico de las redes generativas adversarias

El enfoque básico de las GANs descrito en la [Sección 6.2](#), conocido en inglés como Vanilla GAN, se puede formular mediante un problema min-max entre dos funciones: el generador $G : \mathbb{R}^d \rightarrow \mathbb{R}^n$ y el discriminador $D : \mathbb{R}^n \rightarrow [0, 1]$.

Vamos a usar la siguiente función de pérdida como función objetivo del problema minmax:

$$V(D, G) = \mathbb{E}_{x \sim P_{\mathcal{X}}} [\log D(x)] + \mathbb{E}_{z \sim P_{\mathcal{Z}}} [\log(1 - D(G(z)))]$$

donde \mathbb{E} denota la esperanza con respecto a la distribución de probabilidad especificada en el subíndice (La notación $X \sim P_{\mathcal{X}}$ significa que los datos X son generados a partir de la distribución de probabilidad $P_{\mathcal{X}}$). Vamos a resolver el siguiente problema min-max:

$$\min_G \max_D V(D, G) = \min_G \max_D (\mathbb{E}_{x \sim P_{\mathcal{X}}} [\log D(x)] + \mathbb{E}_{z \sim P_{\mathcal{Z}}} [\log(1 - D(G(z)))])$$

- Dado un generador G , $\max_D V(D, G)$ obtiene el discriminador D que rechaza las muestras generadoras $G(z)$ ya que intenta asignar valores altos a las muestras de la distribución $P_{\mathcal{X}}$ y valores bajos a las muestras generadas $G(z)$.

Básicamente al maximizar estamos asignándole un valor alto a $D(x)$ cuando x es una muestra de la distribución $P_{\mathcal{X}}$ y asignándole un valor bajo a $D(G(z))$ cuando z es una muestra de $P_{\mathcal{Z}}$. Esto equivale a decir que el discriminador le asigne una probabilidad alta a las muestras de entrenamiento y una probabilidad baja a las muestras falsas construidas a partir del generador.

- Dado un discriminador D , $\min_G V(D, G)$ obtiene el generador G que genera muestras $G(z)$ que intentan "engaños" al discriminador D para que le asigne valores altos.

Analizándolo con detalle, nos damos cuenta que para alcanzar el mínimo le tenemos que asignar un valor alto a $D(G(z))$ cuando z es una muestra de $P_{\mathcal{Z}}$, así $(1 - D(G(z)))$ alcanza el mínimo, y por otra parte el valor $D(x)$ es constante ya que viene dado por el discriminador. Es decir, el discriminador le asigna una probabilidad alta a las muestras falsas construidas a partir del generador, de esta forma se consigue engañar al discriminador.

Sea $y = G(z)$ que tiene distribución de probabilidad $P_{\mathcal{Y}} := P_{\mathcal{Z}} \circ G^{-1}$, podemos reescribir $V(D, G)$ en términos de D y $P_{\mathcal{Y}}$ como

$$\begin{aligned} \bar{V}(D, P_{\mathcal{Y}}) &:= V(D, G) = \mathbb{E}_{x \sim P_{\mathcal{X}}} [\log D(x)] + \mathbb{E}_{z \sim P_{\mathcal{Z}}} [\log(1 - D(G(z)))] \\ &= \mathbb{E}_{x \sim P_{\mathcal{X}}} [\log D(x)] + \mathbb{E}_{y \sim P_{\mathcal{Y}}} [\log(1 - D(y))] \\ &= \int_{\mathbb{R}^n} \log D(x) dP_{\mathcal{X}}(x) + \int_{\mathbb{R}^n} \log(1 - D(y)) dP_{\mathcal{Y}}(y) \end{aligned}$$

El problema minmax estudia

$$\min_G \max_D \bar{V}(D, P_{\mathcal{Y}}) = \min_G \max_D (\int_{\mathbb{R}^n} \log D(x) dP_{\mathcal{X}}(x) + \int_{\mathbb{R}^n} \log(1 - D(y)) dP_{\mathcal{Y}}(y))$$

Supongamos que la distribución de probabilidad $P_{\mathcal{X}}$ tiene a $f_{\mathcal{X}}$ como función de densidad y la distribución $P_{\mathcal{Y}}$ a $f_{\mathcal{Y}}$ como función de densidad. Entonces

6. Introducción matemática para la formulación de las GANs

$$\bar{V}(D, P_Y) = \int_{\mathbb{R}^n} (\log D(x) f_X(x) + \log(1 - D(x)) f_Y(x)) dx$$

Haciendo uso de las funciones de densidad el problema minmax se puede reescribir como

$$\min_G \max_D V(D, G) = \min_G \max_D \int_{\mathbb{R}^n} (\log D(x) f_X(x) + \log(1 - D(x)) f_Y(x)) dx$$

Si usamos $\bar{V}(D, P_Y)$, entonces nuestro problema minmax sería $\min_{P_Y} \max_D \bar{V}(D, P_Y)$ bajo la restricción $P_Y = P_Z \circ G^{-1}$. Nos vamos a centrar en el caso de que conocemos las funciones de densidad y en este caso se han establecido los siguientes resultados.

Proposición 6.1. *Dadas dos distribuciones de probabilidad continuas P_X y P_Y en \mathbb{R}^n con funciones de densidad f_X y f_Y respectivamente,*

$$\max_D \bar{V}(D, P_Y) = \max_D \int_{\mathbb{R}^n} (\log D(x) f_X(x) + \log(1 - D(x)) f_Y(x)) dx$$

es alcanzado por $D_{f_X, f_Y}(x) = \frac{f_X(x)}{f_X(x) + f_Y(x)}$ para x en el interior de $\text{supp}(P_X) \cup \text{supp}(P_Y)$, donde $\text{supp}(P_X)$ es la clausura del conjunto $\{x \in \mathbb{R}^n | P_X(x) \neq 0\}$.

Demostración. Para cualquier generador G cuya distribución de probabilidad es P_Y , tratamos de determinar el discriminador D que maximiza la siguiente función.

$$\bar{V}(D, P_Y) = \int_{\mathbb{R}^n} (\log D(x) f_X(x) + \log(1 - D(x)) f_Y(x)) dx$$

Sabemos que para cualquiera dos números reales positivos, no simultáneamente nulos a y b , la función $f : [0, 1] \rightarrow \mathbb{R}$ definida como $f(y) = a \log(y) + b \log(1 - y)$ alcanza su máximo en $\frac{a}{a+b}$.

Claramente, $f'(y) = \frac{a}{y} - \frac{b}{1-y}$ y $f'(y) = 0$ si y solo si $y = \frac{a}{a+b}$. Por tanto, tenemos que $\frac{a}{a+b}$ es un punto crítico y ahora vamos a ver que es un máximo:

- La función es cóncava ya que $f''(y) = -\frac{a}{y^2} - \frac{b}{(1-y)^2} < 0$
- Como $f''(\frac{a}{a+b}) = -\frac{(a+b)^2}{a^2} - \frac{(a+b)^2}{b^2} = -(a+b)^2(\frac{1}{a} + \frac{1}{b}) < 0$, tenemos un único máximo absoluto que se alcanza en el punto crítico $\frac{a}{a+b}$.

Procedemos a aplicar este resultado para concluir la demostración.

Como $(f_X(x), f_Y(x)) \in \mathbb{R}^2 \setminus \{(0,0)\}$ siempre que x pertenezca al interior de $\text{supp}(P_X) \cup \text{supp}(P_Y)$, es decir, son dos números reales positivos que no pueden ser nulos de forma simultánea, y la función discriminadora D asigna una probabilidad entre 0 y 1, entonces tenemos que el máximo de $\bar{V}(D, P_Y)$ se alcanza en $D_{f_X, f_Y}(x) = \frac{f_X(x)}{f_X(x) + f_Y(x)}$ para x perteneciendo al interior de $\text{supp}(P_X) \cup \text{supp}(P_Y)$. \square

6.4.1. Medidas de similitud entre dos distribuciones de probabilidad

Vamos a estudiar formas de medir la distancia entre dos distribuciones de probabilidad continuas P_X y P_Y . Primero, vamos a presentar el concepto de f -divergencia junto con sus propiedades y luego detallaremos casos particulares como son las divergencias de Kullback-Leibler(KL) y de Jensen-Shannon(JSD). Además, probaremos un resultado de f -divergencia que necesitaremos para demostrar el teorema que nos proporciona la solución del enfoque

6.4. Formulación min-max del enfoque básico de las redes generativas adversarias

básico de las GANs formulado como un problema minmax. Para todo este estudio se ha revisado la documentación [PCo8] y [Nie19] junto con los artículos mencionados anteriormente.

El método que utiliza las GANs para medir la diferencia o la distancia entre la distribución generada y la distribución real es la divergencia de Jensen-Shannon que se construye a partir de la divergencia de Kullback-Leibler.

La divergencia KL vamos a ver que mide la diferencia entre dos distribuciones pero no es realmente una distancia porque no es simétrica, para medir la distancia entre las distribuciones de probabilidad usaremos la divergencia JSD que si es simétrica.

6.4.1.1. f -divergencia

Definición 6.3. Sea f una función estrictamente convexa con dominio $I \subseteq \mathbb{R}$ tal que $f(1) = 0$. Suponemos que $f(x) = +\infty$ para todo $x \notin I$. Sean $f_{\mathcal{X}}(x)$ y $f_{\mathcal{Y}}(x)$ dos funciones de densidad de probabilidad en \mathbb{R}^n . Entonces la f -divergencia de $f_{\mathcal{X}}$ y $f_{\mathcal{Y}}$ es definida como

$$D_f(f_{\mathcal{X}}||f_{\mathcal{Y}}) := \mathbb{E}_{x \sim P_{\mathcal{Y}}} [f(\frac{f_{\mathcal{X}}(x)}{f_{\mathcal{Y}}(x)})] = \int_{\mathbb{R}^n} f(\frac{f_{\mathcal{X}}(x)}{f_{\mathcal{Y}}(x)}) f_{\mathcal{Y}}(x) dx$$

donde nosotros adoptamos por convención que $f(\frac{f_{\mathcal{X}}(x)}{f_{\mathcal{Y}}(x)}) f_{\mathcal{Y}}(x) = 0$ si $f_{\mathcal{Y}}(x) = 0$

Proposición 6.2. Sea $f(x)$ una función estrictamente convexa en el dominio $I \subseteq \mathbb{R}$ tal que $f(1) = 0$. Supongamos que $\text{supp}(f_{\mathcal{X}}) \subseteq \text{supp}(f_{\mathcal{Y}})$ (es equivalente a decir $f_{\mathcal{X}} \ll f_{\mathcal{Y}}$ ó que $f(t) > 0$ para $t \in [0, 1]$). Entonces, $D_f(f_{\mathcal{X}}||f_{\mathcal{Y}}) \geq 0$, y $D_f(f_{\mathcal{X}}||f_{\mathcal{Y}}) = 0$ si y solo si $f_{\mathcal{X}} = f_{\mathcal{Y}}$.

Demostración. Por la convexidad de f y la desigualdad de Jensen's ([Teorema 5.4](#))

$$D_f(f_{\mathcal{X}}||f_{\mathcal{Y}}) = \mathbb{E}_{x \sim P_{\mathcal{Y}}} [f(\frac{f_{\mathcal{X}}(x)}{f_{\mathcal{Y}}(x)})] \geq f(\mathbb{E}_{x \sim P_{\mathcal{Y}}} [\frac{f_{\mathcal{X}}(x)}{f_{\mathcal{Y}}(x)}]) = f(\int_{\text{supp}(f_{\mathcal{Y}})} f_{\mathcal{X}}(x) dx) =: f(r)$$

donde la igualdad se cumple si y solo si $\frac{f_{\mathcal{Y}}(x)}{f_{\mathcal{X}}(x)}$ es una constante o f es lineal en el rango de $\frac{f_{\mathcal{X}}(x)}{f_{\mathcal{Y}}(x)}$. Como f es estrictamente convexa solo puede cumplirse que $\frac{f_{\mathcal{Y}}(x)}{f_{\mathcal{X}}(x)}$ sea una constante, $\frac{f_{\mathcal{X}}(x)}{f_{\mathcal{Y}}(x)} = r$ en $\text{supp}(f_{\mathcal{Y}})$. Por lo tanto, para que se cumpla la igualdad debemos tener $f_{\mathcal{X}}(x) = r f_{\mathcal{Y}}(x)$ en $\text{supp}(f_{\mathcal{Y}})$.

Claramente vemos que $r \leq 1$.

- Si suponemos que $\text{supp}(f_{\mathcal{X}}) \subseteq \text{supp}(f_{\mathcal{Y}})$ entonces $\int_{\text{supp}(f_{\mathcal{Y}})} f_{\mathcal{X}}(x) dx = 1$ y tenemos $r = 1$. Por tanto, sustituyendo el valor de r en la desigualdad anterior y aplicando la hipótesis de que $f(1) = 0$ tenemos $D_f(f_{\mathcal{X}}||f_{\mathcal{Y}}) \geq 0$.
- Si $f(t) > 0$ para todo $t \in [0, 1]$ entonces también tenemos $D_f(f_{\mathcal{X}}||f_{\mathcal{Y}}) \geq f(r) \geq 0$ para todo $r \in [0, 1]$. Para $r < 1$ tenemos $D_f(f_{\mathcal{X}}||f_{\mathcal{Y}}) \geq f(r) > 0$. Así, si $D_f(f_{\mathcal{X}}||f_{\mathcal{Y}}) = 0$ tenemos que $r = 1$ y como $f_{\mathcal{X}}(x) = r f_{\mathcal{Y}}(x)$ entonces $f_{\mathcal{X}} = f_{\mathcal{Y}}$.

□

Observación 6.1. La proposición anterior nos afirma que la f -divergencia se comporta como una distancia aunque puede no ser simétrica.

6. Introducción matemática para la formulación de las GANs

6.4.1.2. Casos particulares

- Consideramos la f -divergencia con $f(x) = -\log(x)$ y vamos a ver que cumple las condiciones de Def. 6.3. Sabemos que la función logaritmo es estrictamente cóncava, entonces f es estrictamente convexa. Además, tenemos que $f(1) = -\log(1) = 0$. Por tanto, tenemos la siguiente f -divergencia

$$\begin{aligned} D_f(f_Y||f_X) &:= \mathbb{E}_{x \sim P_X} [f(\frac{f_Y(x)}{f_X(x)})] = \int_{\mathbb{R}^n} f(\frac{f_Y(x)}{f_X(x)}) f_X(x) dx \\ &= \int_{\mathbb{R}^n} -\log(\frac{f_Y(x)}{f_X(x)}) f_X(x) dx = \int_{\mathbb{R}^n} \log(\frac{f_X(x)}{f_Y(x)}) f_X(x) dx \end{aligned}$$

Definición 6.4. La divergencia de Kullback-Leibler mide la distancia entre dos funciones de densidad, f_X y f_Y , también se conoce como divergencia de información y entropía relativa, viene dada por:

$$KL(f_X||f_Y) = \int_{\mathbb{R}^n} f_X(x) \log(\frac{f_X(x)}{f_Y(x)}) dx$$

Observación 6.2. La divergencia de Kullback-Leibler $KL(f_X||f_Y)$ es finita si y solo si $f_Y(x) \neq 0$ en $supp(f_X)$ (ó equivalentemente $supp(f_X) \subset supp(f_Y)$)
Notemos que si $f_X(x) \neq 0$ entonces tenemos $f_Y(x) \neq 0$.

Lema 6.1. La divergencia de Kullback-Leibler entre dos funciones de densidad, f_X y f_Y , cumple

1. $KL(f_X||f_Y) \geq 0$
2. No es simétrica.

Demostración. 1. Se obtiene de Proposición 6.2 por que como acabamos de ver la divergencia KL es un caso particular de la f -divergencia donde $f(x) = -\log(x)$.

2. Sean dos funciones de densidad distintas f_X y f_Y , procedemos a demostrar la asimetría de la divergencia de Kullback-Leibler.

Suponemos $KL(f_X||f_Y) = KL(f_Y||f_X)$ y estudiamos un contraejemplo donde no se cumple la simetría. Sean f_X y f_Y las siguientes funciones de densidad

$$f_X(x) = \begin{cases} 1 & \text{si } x \in [0, 1] \\ 0 & \text{si } x \notin [0, 1] \end{cases}$$

$$f_Y(x) = e^{-\pi x^2} \quad \forall x \in \mathbb{R}$$

Estudiamos las divergencias de KL entre estas funciones de densidad, $KL(f_X||f_Y)$ y $KL(f_Y||f_X)$. Sabemos que $KL(f_Y||f_X) = \int_{\mathbb{R}} f_Y(x) \log(\frac{f_Y(x)}{f_X(x)}) dx$ no es finita ya que el soporte de f_Y no está contenido en el soporte de f_X mientras que

$$\begin{aligned} KL(f_X||f_Y) &= \int_{\mathbb{R}} f_X(x) \log(\frac{f_X(x)}{f_Y(x)}) dx = \int_0^1 \log(\frac{1}{e^{-\pi x^2}}) dx \\ &= \int_0^1 -\log(e^{-\pi x^2}) dx = \int_0^1 \pi x^2 dx = \frac{\pi}{3} \end{aligned}$$

6.4. Formulación min-max del enfoque básico de las redes generativas adversarias

Por tanto, podemos afirmar que la divergencia de Kullback-Leibler no es simétrica.

□

- Consideramos la f -divergencia $f : (0, +\infty) \rightarrow \mathbb{R}$ con $f(x) = \frac{1}{2}x \log(x) - \frac{1}{2}(x+1) \log(\frac{x+1}{2})$ y vamos a ver que cumple las condiciones de Def. 6.3. Primero tenemos que probar que f es estrictamente convexa para ello estudiamos el signo de la segunda derivada. Sabemos que si la segunda derivada de f es positiva entonces la función f es estrictamente convexa,

$$f''(x) = \frac{1}{2x^2+2x} > 0 \quad \forall x \in (0, +\infty)$$

Además, tenemos que $f(1) = \frac{1}{2} \log(1) - \frac{1}{2}(1+1) \log(\frac{1+1}{2}) = 0 + 0 = 0$. Por tanto, tenemos la siguiente f -divergencia que es justo una simetrización de la divergencia anterior.

$$\begin{aligned} D_f(f_{\mathcal{X}} || f_{\mathcal{Y}}) &= \int_{\mathbb{R}^n} f\left(\frac{f_{\mathcal{X}}}{f_{\mathcal{Y}}}\right) f_{\mathcal{Y}} \\ &= \int_{\mathbb{R}^n} \frac{1}{2} \frac{f_{\mathcal{X}}}{f_{\mathcal{Y}}} \log\left(\frac{f_{\mathcal{X}}}{f_{\mathcal{Y}}}\right) - \frac{1}{2} \left(\frac{f_{\mathcal{X}}}{f_{\mathcal{Y}}} + 1\right) \log\left(\frac{\frac{f_{\mathcal{X}}}{f_{\mathcal{Y}}} + 1}{2}\right) \\ &= \frac{1}{2} \int_{\mathbb{R}^n} \frac{f_{\mathcal{X}}}{f_{\mathcal{Y}}} \log\left(\frac{f_{\mathcal{X}}}{f_{\mathcal{Y}}}\right) - \left(\frac{f_{\mathcal{X}}}{f_{\mathcal{Y}}} + 1\right) \log\left(\frac{\frac{f_{\mathcal{X}}}{f_{\mathcal{Y}}} + 1}{2}\right) \\ &= \frac{1}{2} \int_{\mathbb{R}^n} \frac{f_{\mathcal{X}}}{f_{\mathcal{Y}}} \left(\log\left(\frac{f_{\mathcal{X}}}{f_{\mathcal{Y}}}\right) - \log\left(\frac{\frac{f_{\mathcal{X}}}{f_{\mathcal{Y}}} + 1}{2}\right)\right) - \log\left(\frac{\frac{f_{\mathcal{X}}}{f_{\mathcal{Y}}} + 1}{2}\right) \\ &= \frac{1}{2} \int_{\mathbb{R}^n} \frac{f_{\mathcal{X}}}{f_{\mathcal{Y}}} \left(\log\left(\frac{f_{\mathcal{X}}}{f_{\mathcal{Y}}}\right) - \log\left(\frac{2}{f_{\mathcal{X}} + f_{\mathcal{Y}}}\right)\right) + \log\left(\frac{2}{f_{\mathcal{X}} + f_{\mathcal{Y}}}\right) \\ &= \frac{1}{2} \int_{\mathbb{R}^n} \frac{f_{\mathcal{X}}}{f_{\mathcal{Y}}} \log\left(\frac{2f_{\mathcal{X}}}{f_{\mathcal{X}} + f_{\mathcal{Y}}}\right) + \log\left(\frac{2f_{\mathcal{Y}}}{f_{\mathcal{X}} + f_{\mathcal{Y}}}\right) \\ &= \frac{1}{2} \int_{\mathbb{R}^n} f_{\mathcal{X}} \log\left(\frac{2f_{\mathcal{X}}}{f_{\mathcal{X}} + f_{\mathcal{Y}}}\right) + f_{\mathcal{Y}} \log\left(\frac{2f_{\mathcal{Y}}}{f_{\mathcal{X}} + f_{\mathcal{Y}}}\right) \\ &= \frac{1}{2} (KL(f_{\mathcal{X}} || \frac{f_{\mathcal{X}} + f_{\mathcal{Y}}}{2}) + KL(f_{\mathcal{Y}} || \frac{f_{\mathcal{X}} + f_{\mathcal{Y}}}{2})) \end{aligned}$$

Definición 6.5. La divergencia de Jensen-Shannon es otro método para medir la distancia entre dos distribuciones de probabilidad, es una versión simetrizada y suavizada de la divergencia de Kullback-Leibler $KL(f_{\mathcal{X}} || f_{\mathcal{Y}})$ definida como

$$JSD(f_{\mathcal{X}} || f_{\mathcal{Y}}) = \frac{1}{2} (KL(f_{\mathcal{X}} || \frac{f_{\mathcal{X}} + f_{\mathcal{Y}}}{2}) + KL(f_{\mathcal{Y}} || \frac{f_{\mathcal{X}} + f_{\mathcal{Y}}}{2}))$$

Lema 6.2. La divergencia de Jensen-Shannon entre dos funciones de densidad, $f_{\mathcal{X}}$ y $f_{\mathcal{Y}}$, cumple

- $JSD(f_{\mathcal{X}} || f_{\mathcal{Y}}) \geq 0$

6. Introducción matemática para la formulación de las GANs

2. Es simétrica.

Demostración. 1. Se obtiene de [Proposición 6.2](#) por que como acabamos de ver la divergencia JSD es un caso particular de la f-divergencia donde $f(x) = \frac{1}{2}x \log(x) - \frac{1}{2}(x+1) \log(\frac{x+1}{2})$.

2. Claramente, se obtiene $JSD(f_X||f_Y) = JSD(f_Y||f_X)$.

$$\begin{aligned} JSD(f_X||f_Y) &= \frac{1}{2}(KL(f_X||\frac{f_X + f_Y}{2}) + KL(f_Y||\frac{f_X + f_Y}{2})) = \\ &= \frac{1}{2}(KL(f_Y||\frac{f_X + f_Y}{2}) + KL(f_X||\frac{f_X + f_Y}{2})) = JSD(f_Y||f_X) \end{aligned}$$

□

6.4.2. Solución al problema minmax

Bajo la suposición de que las distribuciones de probabilidad son continuas tenemos el siguiente resultado que nos proporciona la solución al problema minmax planteado en nuestro enfoque básico de las GANs. Obsérvese que si las distribuciones de probabilidad no fuesen continuas no tendríamos la solución de nuestro problema. Más adelante, generalizaremos este teorema para tener un resultado que nos de la solución para cualesquiera distribuciones de probabilidad P_X y P_Y .

Teorema 6.2. Sea $f_X(x)$ una función de densidad de probabilidad en \mathbb{R}^n . Para la distribución de probabilidad continua P_Y con función de densidad $f_Y(x)$ y $D : \mathbb{R}^n \rightarrow [0, 1]$ consideramos el problema minmax

$$\min_{P_Y} \max_D \bar{V}(D, P_Y) = \min_{P_Y} \max_D \int_{\mathbb{R}^n} (\log D(x)f_X(x) + \log(1 - D(x))f_Y(x))dx$$

Entonces la solución se obtiene con $f_Y = f_X$ y $D(x) = 1/2$ para todo $x \in \text{supp}(P_X)$.

Demostración. Sabemos que $\max_D \bar{V}(D, P_Y) = \max_D \int_{\mathbb{R}^n} (\log D(x)f_X(x) + \log(1 - D(x))f_Y(x))dx$ se alcanza en $D_{f_X, f_Y}(x) = \frac{f_X(x)}{f_X(x) + f_Y(x)}$ para $x \in \text{supp}(P_X) \cup \text{supp}(P_Y)$.

Definimos la función $c(P_Y) := \bar{V}(D_{f_X, f_Y}, P_Y) = \max_D \bar{V}(D, P_Y)$ donde

$$\begin{aligned} \bar{V}(D_{f_X, f_Y}, P_Y) &= \int_{\mathbb{R}^n} \left(\log \frac{f_X(x)}{f_X(x) + f_Y(x)} f_X(x) + \log \left(1 - \frac{f_X(x)}{f_X(x) + f_Y(x)}\right) f_Y(x) \right) dx \\ &= \int_{\mathbb{R}^n} \left(\log \left(\frac{f_X(x)}{f_X(x) + f_Y(x)} \right) f_X(x) + \log \left(\frac{f_Y(x)}{f_X(x) + f_Y(x)} \right) f_Y(x) \right) dx \\ &= \mathbb{E}_{x \sim P_X} \left[\log \frac{f_X(x)}{f_X(x) + f_Y(x)} \right] + \mathbb{E}_{x \sim P_Y} \left[\log \frac{f_Y(x)}{f_X(x) + f_Y(x)} \right] \end{aligned}$$

Queremos determinar la distribución de probabilidad P_Y donde alcanza el mínimo la función $c(P_Y)$.

6.5. Algoritmo de entrenamiento de las redes generativas adversarias

Si $f_X = f_Y$, entonces $D_{f_X, f_Y} = \frac{1}{2}$ y

$$\begin{aligned} c(P_Y) &= \mathbb{E}_{x \sim P_X}[-\log 2] + \mathbb{E}_{y \sim P_Y}[-\log 2] \\ &= \mathbb{E}_{x \sim P_X}[-\log 2] + \mathbb{E}_{y \sim P_X}[-\log 2] \\ &= \mathbb{E}_{x \sim P_X}[-2 \log 2] \\ &= \mathbb{E}_{x \sim P_X}[-\log 4] = \int_{\mathbb{R}^n} -\log(4)f_X = -\log 4 \end{aligned}$$

En la primera igualdad se usa que $f_X = f_Y$ y entonces aplicando las propiedades de los logaritmos tenemos $\log(\frac{1}{2}) = -\log(2)$, en la segunda igualdad se ha usado la definición de la esperanza matemática junto con que $\int_{\mathbb{R}^n} f_Y = \int_{\mathbb{R}^n} f_X = 1$ y para la tercera igualdad aplicamos la definición de la esperanza matemática. Finalmente, aplicando las propiedades de los logaritmos y la definición de esperanza matemática se concluye a que $c(P_Y) = -\log 4$.

Entonces probando que el mínimo se alcanza en $c(P_Y) = -\log 4$ tenemos lo que se quiere probar. Veamos como alterar la definición de $c(P_Y)$ con objeto de probar lo que se desea y sin modificar su definición. Nos damos cuenta que podemos reescribir $c(P_Y)$ como

$$\begin{aligned} c(P_Y) &= -\log 4 + \int_{\mathbb{R}^n} \log(2 \cdot \frac{f_X(x)}{f_X(x)+f_Y(x)})f_X(x)dx + \int_{\mathbb{R}^n} \log(2 \cdot \frac{f_Y(x)}{f_X(x)+f_Y(x)})f_Y(x)dx = \\ &= -\log 4 + \log 2 + \log 2 + \int_{\mathbb{R}^n} \log(\frac{f_X(x)}{f_X(x)+f_Y(x)})f_X(x)dx + \int_{\mathbb{R}^n} \log(\frac{f_Y(x)}{f_X(x)+f_Y(x)})f_Y(x)dx = \\ &\quad \mathbb{E}_{x \sim P_X}[\log \frac{f_X(x)}{f_X(x)+f_Y(x)}] + \mathbb{E}_{x \sim P_Y}[\log \frac{f_Y(x)}{f_X(x)+f_Y(x)}] \end{aligned}$$

Usando las definiciones [Def. 6.4](#) y [Def. 6.5](#), podemos reescribir:

$$c(P_Y) = -\log 4 + KL(f_X || \frac{f_X+f_Y}{2}) + KL(f_Y || \frac{f_X+f_Y}{2}) = -\log 4 + 2JSD(f_X || f_Y)$$

La función alcanza un mínimo cuando se anula la divergencia JSD , aplicando [Proposición 6.2](#) y que la divergencia JSD es un caso particular de la f -divergencia, entonces tenemos que esto ocurre si $f_X = f_Y$.

Si $f_X = f_Y$ entonces aplicando [Proposición 6.1](#) tenemos que $D(x) = \frac{f_X(x)}{f_X(x)+f_Y(x)} = \frac{f_X(x)}{2f_X(x)} = \frac{1}{2}$ para todo $x \in supp(P_X)$ \square

6.5. Algoritmo de entrenamiento de las redes generativas adversarias

En esta sección se va a presentar el algoritmo de entrenamiento del enfoque básico de las GANs que trata de optimizar la ecuación matemática presentada anteriormente, $V(D, G)$, usando el descenso de gradiente estocástico(SGD) ([1.2.3](#)). Nuestro objetivo es encontrar los pesos de los modelos generador y discriminador que optimizan nuestra función. Primero, se presentará el algoritmo y a continuación, enunciaremos la convergencia de dicho algoritmo.

Hay que tener en cuenta que el problema minmax sin restricciones ([Teorema 6.2](#) y [Teorema 6.5](#)) no es el mismo problema que el problema minmax original donde P_Y se restringe a $P_Y = P_Z \circ G^{-1}$ ([Sección 6.4](#)). En la práctica ambos presentarán las mismas propiedades, de hecho restringiremos las funciones del generador y del discriminador para que actúen

6. Introducción matemática para la formulación de las GANs

como redes neuronales ([Capítulo 1](#)) presentando dicha formulación las mismas propiedades. En consecuencia, tenemos las funciones: $D = D_\omega$ y $G = G_\theta$ donde ω son los pesos que aprende la red discriminadora y θ los pesos que aprende la red generadora, y ajustando $P_{\mathcal{Y}_\theta} = P_{\mathcal{Z}} \circ G_\theta^{-1}$ nuestro problema minmax viene dado como

$$\begin{aligned} \min_{\theta} \max_{\omega} V(D_\omega, G_\theta) &= \min_{\theta} \max_{\omega} \mathbb{E}_{x \sim P_{\mathcal{X}}} [\log D_\omega(x)] + \mathbb{E}_{z \sim P_{\mathcal{Z}}} [\log(1 - D_\omega(G_\theta(z)))] \\ &= \min_{\theta} \max_{\omega} \int_{\mathbb{R}^n} (\log D_\omega(x) dP_{\mathcal{X}} + \log(1 - D_\omega(G_\theta(z))) dP_{\mathcal{Y}_\theta}) \end{aligned}$$

Obsérvese que cada vez que aparezca D_ω y G_θ estamos nombrando a la red neuronal que modela el discriminador y el generador. Además, la equivalencia que existe entre funciones y redes neuronales se motivó en [Teorema 1.1](#).

Algoritmo entrenamiento GAN

- Para un número de iteraciones de entrenamiento **hacer**
 - Para k pasos **hacer**
 - Tomar una muestra de m ejemplos $\{z_1, \dots, z_m\}$ en \mathbb{R}^n de la distribución $P_{\mathcal{Z}}$.
 - Tomar una muestra de m ejemplos $\{x_1, \dots, x_m\} \subset X$ del conjunto de entrenamiento X .
 - Actualizar el discriminador D_ω por ascenso de su gradiente estocástico con respecto a ω :
 - $\nabla_{\omega} \frac{1}{m} \sum_{i=1}^m [\log D_\omega(x_i) + \log(1 - D_\omega(G_\theta(z_i)))]$
 - **fin**
 - Tomar una muestra de m ejemplos $\{z_1, \dots, z_m\}$ en \mathbb{R}^n de la distribución $P_{\mathcal{Z}}$.
 - Actualizar el generador G_θ por descenso de su gradiente estocástico con respecto a θ :
- $\nabla_{\theta} \frac{1}{m} \sum_{i=1}^m [\log(1 - D_\omega(G_\theta(z_i)))]$

■ fin

El siguiente resultado nos sirve para afirmar la convergencia en el algoritmo de optimización iterativo y la existencia de un punto de equilibrio. Una iteración del bucle de este algoritmo consiste en actualizar primero D para un $f_{\mathcal{Y}}$ dado y luego actualizar $f_{\mathcal{Y}}$ con el nuevo D . Repetir este proceso nos conducirá a la solución deseada.

Proposición 6.3. *Si en cada ciclo, el discriminador D se le permite alcanzar su óptimo dado $f_{\mathcal{Y}}(x)$, seguido de una actualización de $f_{\mathcal{Y}}$ para mejorar el criterio de minimización*

$$\min_{f_{\mathcal{Y}}} \int_{\mathbb{R}^n} (\log D(x) f_{\mathcal{X}}(x) + \log(1 - D(x)) f_{\mathcal{Y}}(x)) dx$$

entonces $f_{\mathcal{Y}}$ converge a $f_{\mathcal{X}}$.

Una vez probada la convergencia del algoritmo, se dieron cuenta que al actualizar la red neuronal G_θ a partir de minimizar $\mathbb{E}[\log(1 - D_\omega(G_\theta(z)))]$ puede saturarse antes de que finalice el algoritmo. Por tanto, se decidió minimizar $-\mathbb{E}[\log D_\omega(G_\theta(z))]$ en lugar de $\mathbb{E}[\log(1 - D_\omega(G_\theta(z)))]$.

6.6. Formulación minmax para el marco general de las redes generativas adversarias

En la sección 6.4. hemos asumido que nuestros datos siguen una distribución de probabilidad continua y bajo esa hipótesis el problema de minmax alcanza una solución. En esta sección veremos que la técnica minmax sigue dándonos la solución incluso cuando la distribución de probabilidad no sea continua. Por otra parte, se estudiará como se puede generalizar nuestro enfoque básico a un marco general conocido como f -GAN.

Definición 6.6. La conjugada convexa de una función convexa $f(x)$ es también conocida como la transformada de Fenchel ó la transformada de Fenchel-Legendre de f , ya que es una generalización de la conocida transformada de Legendre. Sea $f(x)$ una función convexa definida en un intervalo $I \subseteq \mathbb{R}$. Entonces su conjugada convexa $f^* : \mathbb{R} \rightarrow \mathbb{R} \cup \{+\infty\} \cup \{-\infty\}$ se define como

$$f^*(y) := \sup_{t \in I} \{ty - f(t)\}$$

Vamos a formular una generalización del enfoque básico de las GANs usando el concepto de f -divergencia (Def. 6.3) conocida como f -GAN. Para una distribución de probabilidad dada P_X , el objetivo de una f -GAN es minimizar la f -divergencia $D_f(P_X || P_Y)$ con respecto a la distribución de probabilidad P_Y .

Primero, vamos a estudiar como estimar la f -divergencia usando la conjugada convexa f^* . Suponemos que $P_X \ll P_Y$ y aplicando que $f(x) = (f^*)^*(x)$ tenemos

$$\begin{aligned} D_f(P_X || P_Y) &:= \int_{\mathbb{R}^n} f\left(\frac{f_X}{f_Y}\right) f_Y \\ &= \int_{\mathbb{R}^n} \sup_t \left\{ t \frac{f_X}{f_Y} - f^*(t) \right\} f_Y \\ &= \int_{\mathbb{R}^n} \sup_t \left\{ t f_X - f^*(t) f_Y \right\} \\ &\geq \int_{\mathbb{R}^n} (T f_X - f^*(T) f_Y) \\ &= \mathbb{E}_{x \sim P_X} [T(x)] - \mathbb{E}_{x \sim P_Y} [f^*(T(x))] \end{aligned}$$

donde $T(x)$ es cualquier función Borel. Por tanto, tomando el supremo sobre todas las funciones Borel T tenemos

$$D_f(P_X || P_Y) \geq \sup_T (\mathbb{E}_{x \sim P_X} [T(x)] - \mathbb{E}_{x \sim P_Y} [f^*(T(x))])$$

Observación 6.3. Obsérvese que para cada x , $\sup_t \{t f_X - f^*(t) f_Y\}$ es alcanzado para algún $t = T^*(x)$, entonces se da la igualdad

$$D_f(P_X || P_Y) = (\mathbb{E}_{x \sim P_X} [T^*(x)] - \mathbb{E}_{x \sim P_Y} [f^*(T^*(x))])$$

y podemos cambiar el supremo por un máximo.

6. Introducción matemática para la formulación de las GANs

El siguiente resultado nos asegura que bajo ciertas condiciones se da la igualdad, se alcanza el supremo y nos define la función T^*

Teorema 6.3. *Sea $f(t)$ una función estrictamente convexa y continuamente diferenciable en $I \subseteq \mathbb{R}$. Sean P_X y P_Y medidas de probabilidad de Borel en \mathbb{R}^n cumpliendo $P_X \ll P_Y$. Entonces*

$$D_f(P_X || P_Y) = \sup_T (\mathbb{E}_{x \sim P_X}[T(x)] - \mathbb{E}_{x \sim P_Y}[f^*(T(x))])$$

donde \sup_T es tomado entre todas las funciones Borel $T : \mathbb{R}^n \rightarrow \text{Dom}(f^*)$. Además, se asume que $f_X(x) \in I$ para todo x . Entonces el optimizador se alcanza en $T^*(x) := f'(f_X(x))$.

Observación 6.4. Cuando P_X no sea absolutamente continua respecto a P_Y , el supremo no se alcanza y toma el valor infinito. Esto puede estudiarse con más detalle en el teorema 3.5. del artículo [Wan20].

Como se ha probado que la f -divergencia puede estimarse usando la conjugada convexa y describirse como la función objetivo de un problema minmax, entonces se puede definir la f -GAN de la siguiente forma. El problema minmax se enunciará en términos de ínfimos y supremos ya que no hay garantía de que se alcance la función T que maximiza y la distribución P_Y que minimiza la función objetivo.

Definición 6.7. Una f -GAN es una generalización del enfoque básico de las GANs que resuelve el siguiente problema minmax

$$\inf_{P_Y} \sup_T (\mathbb{E}_{x \sim P_Y}[T(x)] - \mathbb{E}_{x \sim P_X}[f^*(T(x))])$$

Este problema de optimización se conoce como *Minimización de la Divergencia Variacional (VDM)*.

Cuando las distribuciones de probabilidad P_X y P_Y son continuas el [Teorema 6.2](#) nos da la solución al problema minmax pero si no son continuas tendremos que estudiar cual es su solución.

A continuación, veamos un ejemplo extraído de la página 17 de [Wan20] de una f -GAN donde apreciaremos como la f -GAN es una generalización del enfoque básico de las GANs.

Ejemplo de f -GAN

Sea $f(x) = x \log(x) - (x + 1) \log(\frac{x+1}{2})$ conocida como *la divergencia de Jensen-Shannon* y definida en $I = (0, \infty)$, entonces vamos a estudiar la conjugada convexa de la función f definida como $f^*(y) = \sup_{x \in I} \{xy - f(x)\}$.

Para un valor $y \in I^*$ arbitrario definimos la función $g(x) := xy - f(x) = xy + x \log(x) - (x + 1) \log(\frac{x+1}{2})$. A continuación, vamos a estudiar los puntos críticos de la función g con objeto de hallar su máximo.

$$\begin{aligned} g'(x) &= y - \log(x) - 1 + \log\left(\frac{x+1}{2}\right) + 1 \\ &= y - \log(2) + \log\left(\frac{x+1}{x}\right) \end{aligned}$$

6.6. Formulación minmax para el marco general de las redes generativas adversarias

Tenemos que $g'(x) = 0$ si y solo si $x = \frac{1}{2e^{-y}-1}$, entonces el supremo de la función g se alcanza en $\frac{1}{2e^{-y}-1}$. Por tanto, la conjugada convexa se define como

$$\begin{aligned} f^*(y) &= g\left(\frac{1}{2e^{-y}-1}\right) \\ &= \frac{1}{2e^{-y}-1}y - \frac{1}{2e^{-y}-1}\log\left(\frac{1}{2e^{-y}-1}\right) + \left(\frac{1}{2e^{-y}-1}+1\right)\log\left(\frac{\frac{1}{2e^{-y}-1}+1}{2}\right) \\ &= \frac{y}{2e^{-y}-1} - \frac{1}{2e^{-y}-1}\log\left(\frac{1}{2e^{-y}-1}\right) + \frac{1}{2e^{-y}-1}\log\left(\frac{\frac{1}{2e^{-y}-1}+1}{2}\right) + \log\left(\frac{\frac{1}{2e^{-y}-1}+1}{2}\right) \\ &= \frac{y}{2e^{-y}-1} - \frac{1}{2e^{-y}-1}\log\left(\frac{1}{2e^{-y}-1}\right) + \frac{1}{2e^{-y}-1}\log\left(\frac{e^{-y}}{2e^{-y}-1}\right) + \log\left(\frac{e^{-y}}{2e^{-y}-1}\right) \\ &= \frac{y}{2e^{-y}-1} - \frac{1}{2e^{-y}-1}\log\left(\frac{1}{2e^{-y}-1}\right) - \frac{y}{2e^{-y}-1} + \frac{1}{2e^{-y}-1}\log\left(\frac{1}{2e^{-y}-1}\right) \\ &\quad + \log\left(\frac{e^{-y}}{2e^{-y}-1}\right) = \log\left(\frac{e^{-y}}{2e^{-y}-1}\right) = \log(e^{-y}) - \log(2e^{-y}-1) \\ &= -\log(e^y) - \log(2e^{-y}-1) = -\log(e^y(2e^{-y}-1)) = -\log(2-e^y) \end{aligned}$$

Finalmente, tenemos $f^*(y) = -\log(2-e^y)$ con dominio $I^* = (-\infty, \log 2)$ y el correspondiente problema minmax de la f -GAN es

$$\begin{aligned} \inf_{P_y} \sup_T (\mathbb{E}_{x \sim P_y}[T(x)] - \mathbb{E}_{x \sim P_x}[f^*(T(x))]) &= \\ &= \inf_{P_y} \sup_T (\mathbb{E}_{x \sim P_y}[T(x)] + \mathbb{E}_{x \sim P_x}[\log(2-e^{T(x)})]) \end{aligned}$$

Veamos la equivalencia con nuestro problema minmax formulado para el enfoque básico de las GANs, tomando $T(x) = \log(1-D(x))+\log 2$, entonces tenemos $D(x) = 1 - \frac{1}{2}e^{T(x)}$ donde $T(x) < \log 2$.

$$\begin{aligned} \inf_{P_y} \sup_T (\mathbb{E}_{x \sim P_y}[T(x)] - \mathbb{E}_{x \sim P_x}[f^*(T(x))]) &= \\ &= \inf_{P_y} \sup_T (\mathbb{E}_{x \sim P_y}[T(x)] + \mathbb{E}_{x \sim P_x}[\log(2-e^{T(x)})]) \\ &= \inf_{P_y} \sup_{D>0} (\mathbb{E}_{x \sim P_y}[\log(1-D(x))+\log 2] + \mathbb{E}_{x \sim P_x}[\log(2-e^{\log(1-D(x))+\log 2})]) \\ &= \inf_{P_y} \sup_{D>0} (\mathbb{E}_{x \sim P_y}[\log(1-D(x))] + \log 2 + \mathbb{E}_{x \sim P_x}[\log(2-2(1-D(x)))])) \\ &= \inf_{P_y} \sup_{D>0} (\mathbb{E}_{x \sim P_y}[\log(1-D(x))] + \log 2 + \mathbb{E}_{x \sim P_x}[\log(2D(x))]) \\ &= \inf_{P_y} \sup_{D>0} (\mathbb{E}_{x \sim P_y}[\log(1-D(x))] + \log 2 + \mathbb{E}_{x \sim P_x}[\log(D(x))] + \log 2) \\ &= \inf_{P_y} \sup_{D>0} (\mathbb{E}_{x \sim P_x}[\log(D(x))] + \mathbb{E}_{x \sim P_y}[\log(1-D(x))]) + \log 4 \end{aligned}$$

Obsérvese que en la segunda igualdad se utiliza la definición de $T(x)$ y nos queda la formulación en términos de $D(x)$, entonces tratamos de encontrar la función D donde alcanza el

6. Introducción matemática para la formulación de las GANs

supremo. En la tercera igualdad se aplican propiedades de logaritmos y exponentiales, y en la cuarta igualdad se simplifica la expresión resultante. Nótese como en la quinta igualdad se aplica la propiedad del logaritmo que enuncia que el logaritmo de un producto es igual a la suma de los logaritmos de los factores y la propiedad de la linealidad de la esperanza ([Proposición 5.7](#)). Finalmente, en la sexta igualdad se reordenan los términos y se aplican las propiedades de los logaritmos.

Por tanto, ignorando la constante $\log 4$ tenemos que el enfoque básico de las GANs es un caso especial de f -GAN siendo f la divergencia de Jensen-Shannon ([Def. 6.5](#)). El ínfimo puede sustituirse por un mínimo y el supremo por un máximo ya que tenemos garantizada la existencia de una función Borel donde se alcanza el supremo y una medida de probabilidad Borel donde se alcanza el ínfimo.

El siguiente teorema nos proporciona la solución de una f -GAN y nos servirá de soporte para demostrar la solución del problema minmax formulado para un marco general de una GAN donde no se asume que las distribuciones de probabilidades deban ser continuas.

Teorema 6.4. *Sea $f(t)$ una función semicontinua inferior estrictamente convexa tal que el dominio I^* de f^* tiene $\sup I^* = b^* \in [0, \infty)$. Suponga además que f es continuamente diferenciable en su dominio y que $f'(t) > 0$ para $t \in (0, 1)$. Sea $P_{\mathcal{X}}$ la medida de probabilidad de Borel en \mathbb{R}^n . Entonces $P_{\mathcal{X}}$ es el único optimizador de*

$$\inf_{P_{\mathcal{Y}}} \sup_T (\mathbb{E}_{x \sim P_{\mathcal{Y}}} [T(x)] - \mathbb{E}_{x \sim P_{\mathcal{X}}} [f^*(T(x))])$$

donde \sup_T se alcanza entre todas las funciones Borel $T : \mathbb{R}^n \rightarrow \text{Dom}(f^*)$ y $\inf_{P_{\mathcal{Y}}}$ se toma entre todas las medidas de probabilidad de Borel.

Observación 6.5. La hipótesis de que $\sup I^* = b^* \in [0, \infty)$ es necesaria para que que se alcance el supremo y podamos intercambiar supremo por máximo. La importancia de esta hipótesis se refleja en el teorema 3.6. del artículo [[Wan20](#)].

Para el marco general de cualquier distribución de probabilidad donde no se conocen las funciones de densidad tenemos el siguiente teorema cuya demostración utiliza los resultados expuestos anteriormente.

Teorema 6.5. *Sea $P_{\mathcal{X}}$ una distribución de probabilidad dada en \mathbb{R}^n . Para la distribución de probabilidad $P_{\mathcal{Y}}$ y la función $D : \mathbb{R}^n \rightarrow [0, 1]$ consideramos el problema minmax*

$$\min_{P_{\mathcal{Y}}} \max_D \bar{V}(D, P_{\mathcal{Y}}) = \min_{P_{\mathcal{Y}}} \max_D \int_{\mathbb{R}^n} (\log D(x) dP_{\mathcal{X}}(x) + \log(1 - D(x)) dP_{\mathcal{Y}}(x))$$

Entonces la solución se obtiene con $P_{\mathcal{X}} = P_{\mathcal{Y}}$ y $D(x) = \frac{1}{2}$ $P_{\mathcal{X}}$ -casi por doquier (Esto es, se obtiene esta solución en todo punto salvo a lo sumo en un conjunto de medida nula).

Demostración. Sea $\bar{V}(D, P_{\mathcal{Y}}) = \int_{\mathbb{R}^n} (\log D(x) dP_{\mathcal{X}}(x) + \log(1 - D(x)) dP_{\mathcal{Y}}(x)) = \mathbb{E}_{x \sim P_{\mathcal{X}}} [\log D(x)] + \mathbb{E}_{x \sim P_{\mathcal{Y}}} [\log(1 - D(x))]$.

Notemos que resolver este problema minmax para el enfoque básico de las GANs es equivalente a resolver el problema minmax para la f -GAN mostrada en el ejemplo ([6.6](#)), $f(x) = x \log(x) - (x + 1) \log(\frac{x+1}{2})$, y demostrar nuestro teorema. Por tanto, podemos usar [Teorema 6.4](#) con la función del ejemplo y probar todas las hipótesis de [Teorema 6.4](#) para poder aplicarlo.

6.6. Formulación minmax para el marco general de las redes generativas adversarias

- Primero, tenemos que probar que f es estrictamente convexa pero esto ya se estudió en [6.4.1.2](#) donde se vió que la función f es una f -divergencia, concretamente la *divergencia de Jensen-Shannon* ([Def. 6.5](#)).
- Claramente f es continuamente diferenciable, $f \in \mathcal{C}^\infty$, al ser combinación lineal de funciones polinómicas y logaritmos.

$$f'(x) = \log(x) + 1 - \log(\frac{x+1}{2}) - 1 = \log(x) - \log(\frac{x+1}{2})$$

Por tanto, $\exists f'$ y es continua en $(0, +\infty)$.

- A continuación, vamos a probar que $f(x) > 0$ para todo $x \in (0, 1)$. Estudiamos los puntos críticos de la función para así poder analizar el signo de f .

$$\begin{aligned} f'(x) &= \log(x) - \log(\frac{x+1}{2}) = \log(2) + \log(\frac{x}{x+1}) \\ f'(x) &= 0 \iff x = 1 \\ f''(x) &= \frac{1}{x(x+1)} \quad y \quad f''(1) = \frac{1}{2} > 0 \end{aligned}$$

Por tanto, f alcanza un mínimo en el punto $x = 1$ y como $f(1) = 0$ podemos afirmar que en todos los puntos a la izquierda y a la derecha de 1 se tiene que la función es positiva. Luego, $f(x) > 0$ para todo $x \in (0, 1)$.

- La conjugada convexa ([Def. 6.6](#)) de la función f se estudió en el ejemplo anterior [6.6](#). Tenemos $f^*(u) = -\log(2 - e^u)$ con dominio $I^* = (-\infty, \log 2)$, cumpliendo que $\sup I^* = b^* = \log 2 \in [0, \infty)$.

Entonces si aplicamos [Teorema 6.4](#) tenemos que el único optimizador del enfoque f -GAN es $P_{\mathcal{X}} = P_{\mathcal{Y}}$. La equivalencia con nuestro problema minmax se vio en el ejemplo [6.6](#), entonces el único optimizador del enfoque básico de las GANs es $P_{\mathcal{X}} = P_{\mathcal{Y}}$.

Claramente, si $P_{\mathcal{X}} = P_{\mathcal{Y}}$ tenemos que $D(x) = \frac{1}{2}$ $P_{\mathcal{X}}$ -casi por doquier □

Parte III.

Experimentación y resultados

7. Experimentación y resultados

En este capítulo se van a realizar experimentos para verificar la utilidad de crear contenido sintético haciendo uso de redes generativas adversarias. Concretamente, se van a generar dígitos falsos y caras falsas a través de modelos GANs como son DCGAN y WGAN-GP. La experimentación puede encontrarse en el siguiente enlace de Github: <https://github.com/silviabm98/TFG>

7.1. Metodología

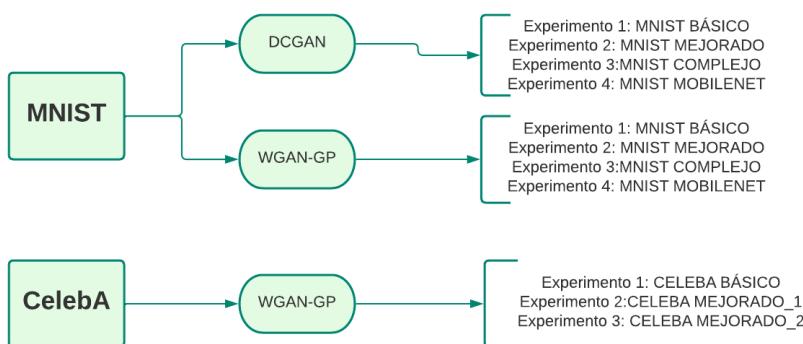
7.1.1. Descripción

Inicialmente se realizó un primer experimento con el modelo DCGAN debido a su simplicidad y a su alto rendimiento en el tratamiento con imágenes. Su simplicidad se debe a que introdujeron cambios sin modificar la arquitectura de la GAN, los cambios básicos fueron implementar tanto el generador como el discriminador como CNN.

Durante el entrenamiento de este modelo se apreció la lenta convergencia del discriminador y se optó por otro modelo, WGAN-GP, donde se alcanzó un rendimiento mucho mejor ya que tiene una base teórica muy buena y que se puede estudiar con más detalle [GAA⁺17].

A continuación, vamos a ver la implementación de las clases que definen los modelos. Con cada modelo realizaremos una serie de experimentos, definiendo distintos modelos de generador y discriminador, usando distintos optimizadores, etc. Los experimentos detallados los veremos en Subsección 7.1.3 y los parámetros que se han utilizado en cada experimento se han escogido tras realizar diversas pruebas pero no se ha realizado una búsqueda exhaustiva con objeto de encontrar los mejores parámetros que generen las mejores imágenes sintéticas. Este objetivo sería interesante plantearlo para un trabajo futuro.

Estos modelos se han probado para las bases de datos MNIST y CelebA, podemos ver un resumen que engloba todas las pruebas realizadas.



7. Experimentación y resultados

7.1.1.1. Modelo DCGAN

La siguiente clase GAN nos permite definir nuestro modelo DCGAN que usaremos en la base de datos MNIST.

```
1 class GAN(keras.Model):
2     def __init__(self, discriminator, generator, latent_dim):
3         super().__init__()
4         self.discriminator = discriminator
5         self.generator = generator
6         self.latent_dim = latent_dim
7         self.d_loss_metric = keras.metrics.Mean(name="d_loss")
8         self.g_loss_metric = keras.metrics.Mean(name="g_loss")
9
10    def compile(self, d_optimizer, g_optimizer, loss_fn):
11        super(GAN, self).compile()
12        self.d_optimizer = d_optimizer
13        self.g_optimizer = g_optimizer
14        self.loss_fn = loss_fn
15
16    def metrics(self):
17        return [self.d_loss_metric, self.g_loss_metric]
18
19
20    def evaluate_D(self):
21        batch_size = tf.shape(x_train)[0]
22        random_latent_vectors = tf.random.normal( shape=(batch_size, self.
latent_dim))
23        generated_images = self.generator(random_latent_vectors)
24
25        self.discriminator.compile(optimizer=self.d_optimizer, loss=self.
loss_fn,metrics=['accuracy'])
26
27        loss_real, acc_real = self.discriminator.evaluate(x_test,tf.ones((
batch_size, 1),dtype=tf.dtypes.float32), verbose=1)
28
29        loss_fake, acc_fake = self.discriminator.evaluate(generated_images,
tf.zeros((batch_size, 1),dtype=tf.dtypes.float32) , verbose=1)
30
31        print('>Accuracy real: %.0f%%, fake: %.0f%%' % (acc_real*100,
acc_fake*100))
32        print('>Loss real: ')
33        print(loss_real)
34        print('>Loss fake: ')
35        print(loss_fake)
36
37    def evaluate_G(self):
38        batch_size = tf.shape(x_train)[0]
39        random_latent_vectors = tf.random.normal( shape=(batch_size, self.
latent_dim))
40        generated_images = self.generator(random_latent_vectors)
41
42        for i in range(9):
```

```

43     random_latent_vectors = tf.random.normal(
44         shape=(9, 128))
45     pred = generator(random_latent_vectors)
46     plt.subplot(331 + i)
47     plt.axis('off')
48     plt.imshow(np.squeeze(pred[0]), cmap='gray')
49     plt.show()
50
51 def train_step(self, real_images):
52     batch_size = tf.shape(real_images)[0]
53     random_latent_vectors = tf.random.normal(shape=(batch_size, self.
54     latent_dim))
55     generated_images = self.generator(random_latent_vectors)
56     combined_images = tf.concat([generated_images, real_images], axis
57     =0)
58     labels = tf.concat(
59         [tf.ones((batch_size, 1)), tf.zeros((batch_size, 1))],
60         axis=0
61     )
62     labels += 0.05 * tf.random.uniform(tf.shape(labels))
63
64     with tf.GradientTape() as tape:
65         predictions = self.discriminator(combined_images)
66         d_loss = self.loss_fn(labels, predictions)
67
68         grads = tape.gradient(d_loss, self.discriminator.trainable_weights)
69         self.d_optimizer.apply_gradients(
70             zip(grads, self.discriminator.trainable_weights)
71         )
72
73     random_latent_vectors = tf.random.normal(
74         shape=(batch_size, self.latent_dim))
75     misleading_labels = tf.zeros((batch_size, 1))
76
77     with tf.GradientTape() as tape:
78         predictions = self.discriminator(
79             self.generator(random_latent_vectors))
80         g_loss = self.loss_fn(misleading_labels, predictions)
81
82         grads = tape.gradient(g_loss, self.generator.trainable_weights)
83         self.g_optimizer.apply_gradients(
84             zip(grads, self.generator.trainable_weights))
85
86         self.d_loss_metric.update_state(d_loss)
87         self.g_loss_metric.update_state(g_loss)
88     return {"d_loss": self.d_loss_metric.result(),
89             "g_loss": self.g_loss_metric.result()}
90
91
92

```

7. Experimentación y resultados

Analizamos con detalle la clase definida para crear un modelo DCGAN.

- El constructor **init** se encarga de inicializar el discriminador, el generador, el tamaño del espacio latente, las funciones de pérdida del discriminador y generador. Estas métricas se inicializan con Mean que calcula la media de la función de pérdida con la que compilamos el modelo.
- El método **compile** trata de compilar el modelo GAN inicializando los optimizadores y la función de pérdida. En este modelo vamos a usar como optimizador Adam y usaremos la entropía cruzada binaria como función de pérdida del modelo GANs.
- Los métodos **evaluate_D** y **evaluate_G** se encargan de evaluar al discriminador y al generador. Para evaluar al generador simplemente se generan imágenes de forma aleatoria y se visualizan, mientras que para evaluar al discriminador se usa la función **evaluate** del modelo. De hecho, se evalúa al discriminador con las imágenes reales y con las falsas, y finalmente se muestra por pantalla la pérdida que se obtiene.
- El método **metrics** devuelve las métricas obtenidas con el generador y discriminador.

- El método **train_step** se encarga del entrenamiento GAN y su funcionamiento consiste:

1. Crear vectores aleatorios en el espacio latente a partir de una distribución normal. Se pretende crear tantos vectores como tamaño tenga el batch, `batch_size`, y estos vectores deben tener las dimensiones `latent_dim`.

```
1 random_latent_vectors = tf.random.normal( shape=(batch_size, self.  
latent_dim))  
2
```

2. Generar imágenes con el generador a partir de los vectores aleatorios.

```
1 generated_images = self.generator(random_latent_vectors)  
2
```

3. Mezclar las imágenes generadas con las imágenes reales.

```
1 combined_images = tf.concat([generated_images, real_images], axis  
=0)  
2
```

Las etiquetas de las imágenes combinadas las creamos nosotros introduciendo algo de ruido.

```
1 labels = tf.concat( [tf.ones((batch_size, 1)), tf.zeros((  
batch_size, 1))],axis=0)  
2 labels += 0.05 * tf.random.uniform(tf.shape(labels))  
3
```

4. Entrenar el discriminador usando las imágenes mezcladas con las etiquetas correspondientes: *real* para las imágenes reales y *falso* para las imágenes generadas. Las etiquetas las creamos nosotros introduciendo algo de ruido.

- La función `tf.GradientTape()` la denotamos como `tape` y nos servirá para calcular el gradiente. Esta función es útil para implementar el algoritmo de retropropagación en el entrenamiento de redes neuronales.

```

1 with tf.GradientTape() as tape:
2

```

- Calculamos el valor de la función de pérdida de la GANs entre las etiquetas reales, labels, y las predicciones de nuestro modelo, predictions = self.discriminator(combined_images)

```

1 d_loss = self.loss_fn(labels, predictions)
2

```

- Calculamos el gradiente de la función de pérdida con los pesos entrenables del discriminador. Para calcular el gradiente de una función se usa la función gradient() de tf.GradientTape() cuyo primer argumento es la función, en este caso la función de pérdida d_loss, y de segundo argumento la variable sobre la que estamos calculando el gradiente, esto es los pesos del discriminador.

```

1 grads = tape.gradient(d_loss, self.discriminator.
2                         trainable_weights)
3

```

- Aplicamos el optimizador del modelo discriminador (en este caso Adam) a el gradiente para actualizar los pesos del discriminador. La función apply_gradients necesita conocer una lista de parejas gradiente y variable del gradiente, entonces con la función zip juntamos el gradiente calculado antes con los pesos del discriminador.

```

1 self.d_optimizer.apply_gradients(
2     zip(grads, self.discriminator.trainable_weights)
3 )
4

```

5. Crear nuevos vectores aleatorios en el espacio latente

```

1 random_latent_vectors = tf.random.normal(
2     shape=(batch_size, self.latent_dim))
3 misleading_labels = tf.zeros((batch_size, 1))
4

```

6. Entrenar el generador usando estos vectores aleatorios, con etiquetas de *real*. De esta forma actualizamos los pesos del generador con el objetivo de que el discriminador prediga que las imágenes generadas son reales y así engañamos al discriminador.

Se realiza un comportamiento análogo al entrenamiento del discriminador (al paso 4) pero actualizando la pérdida del generador, g_loss, calculando su gradiente y actualizando los pesos entrenables del generador.

7. Finalmente, actualizamos los valores de las funciones de pérdida de los modelos y los mostramos por pantalla.

7.1.1.2. Modelo WGAN-GP

La siguiente clase WGAN será usada para definir nuestro modelo WGAN-GP que usaremos con las bases de datos MNIST y CelebA. A continuación, analizaremos con detalle cada método de la clase.

7. Experimentación y resultados

```
1  class WGAN(keras.Model):
2      def __init__(self, discriminator, generator, latent_dim,
3                   discriminator_extra_steps=3, gp_weight=10.0):
4          super(WGAN, self).__init__()
5
6          self.discriminator = discriminator
7          self.generator = generator
8          self.latent_dim = latent_dim
9          self.d_steps = discriminator_extra_steps
10         self.gp_weight = gp_weight
11
12
13     def compile(self, d_optimizer, g_optimizer, d_loss_fn, g_loss_fn):
14         super(WGAN, self).compile()
15         self.d_optimizer = d_optimizer
16         self.g_optimizer = g_optimizer
17         self.d_loss_fn = d_loss_fn
18         self.g_loss_fn = g_loss_fn
19
20
21     def evaluate_D(self):
22         batch_size = tf.shape(x_train)[0]
23         random_latent_vectors = tf.random.normal( shape=(batch_size, self.
24 latent_dim))
25         generated_images = self.generator(random_latent_vectors)
26
27         self.discriminator.compile(optimizer=self.d_optimizer, loss=self.
28 d_loss_fn,metrics=['accuracy'])
29
30         loss_real, acc_real = self.discriminator.evaluate(x_test,tf.ones((
31 batch_size, 1),dtype=tf.dtypes.float32), verbose=1)
32
33         loss_fake, acc_fake = self.discriminator.evaluate(generated_images,tf
34 .zeros((batch_size, 1),dtype=tf.dtypes.float32) , verbose=1)
35
36         print('>Accuracy real: %.0f%%, fake: %.0f%%' % (acc_real*100,
37 acc_fake*100))
38         print('>Loss real: ')
39         print(loss_real)
40         print('>Loss fake: ')
41         print(loss_fake)
42
43     def evaluate_G(self):
44         batch_size = tf.shape(x_train)[0]
45         random_latent_vectors = tf.random.normal( shape=(batch_size, self.
46 latent_dim))
47         generated_images = self.generator(random_latent_vectors)
48
49         for i in range(9):
50             random_latent_vectors = tf.random.normal(
51             shape=(9, 128))
52             pred = generator(random_latent_vectors )
```

```

47     plt.subplot(331 + i)
48     plt.axis('off')
49     plt.imshow(np.squeeze(pred[0]), cmap='gray')
50     plt.show()
51
52
53     def gradient_penalty(self, batch_size, real_images, fake_images):
54         alpha = tf.random.normal([batch_size, 1, 1, 1], 0.0, 1.0)
55         diff = fake_images - real_images
56         interpolated = real_images + alpha * diff
57
58         with tf.GradientTape() as gp_tape:
59             gp_tape.watch(interpolated)
60             pred = self.discriminator(interpolated, training=True)
61
62             grads = gp_tape.gradient(pred, [interpolated])[0]
63             norm = tf.sqrt(tf.reduce_sum(tf.square(grads), axis=[1, 2, 3]))
64             gp = tf.reduce_mean((norm - 1.0) ** 2)
65
66         return gp
67
68     def train_step(self, real_images):
69         if isinstance(real_images, tuple):
70             real_images = real_images[0]
71
72         batch_size = tf.shape(real_images)[0]
73
74         for i in range(self.d_steps):
75             random_latent_vectors = tf.random.normal(
76                 shape=(batch_size, self.latent_dim)
77             )
78             with tf.GradientTape() as tape:
79                 fake_images = self.generator(random_latent_vectors, training=
80 True)
81
82                 fake_logits = self.discriminator(fake_images, training=True)
83                 real_logits = self.discriminator(real_images, training=True)
84
85                 d_cost = self.d_loss_fn(real_img=real_logits, fake_img=
86 fake_logits)
87                 gp = self.gradient_penalty(batch_size, real_images,
88 fake_images)
89                 d_loss = d_cost + gp * self.gp_weight
90
91                 d_gradient = tape.gradient(d_loss, self.discriminator.
92 trainable_variables)
93
94                 self.d_optimizer.apply_gradients(zip(d_gradient, self.
95 discriminator.trainable_variables))
96
97                 random_latent_vectors = tf.random.normal(shape=(batch_size, self.
98 latent_dim))
99                 with tf.GradientTape() as tape:

```

7. Experimentación y resultados

```
92         generated_images = self.generator(random_latent_vectors, training  
93 =True)  
94         gen_img_logits = self.discriminator(generated_images, training=  
95 True)  
96         g_loss = self.g_loss_fn(gen_img_logits)  
97  
98         gen_gradient = tape.gradient(g_loss, self.generator.  
trainable_variables)  
99         self.g_optimizer.apply_gradients(zip(gen_gradient, self.generator.  
trainable_variables))  
100     return {"d_loss": d_loss, "g_loss": g_loss}
```

- El constructor **init** se encarga de inicializar el discriminador, el generador y el tamaño del espacio latente como hacíamos con la clase GAN. Una de las diferencias que presenta este modelo es que se necesita entrenar al discriminador varias veces entre cada actualización del generador y por eso necesitamos inicializar la variable `d_steps`. Además, la función de pérdida del discriminador añade un término de penalización que se modela con la variable `gp_weight`.
- El método **compile** trata de compilar el modelo WGAN-GP y este modelo difiere en que ahora se presentan dos funciones de pérdida diferentes, una para el discriminador y otra para el generador. Como optimizador se usará al igual que antes Adam y la función de pérdida es la pérdida de Weierstrass. A continuación, procedemos a observar la declaración inicial de optimizadores y funciones de pérdida en nuestro código.

```
1 generator_optimizer = tf.keras.optimizers.Adam(  
2     learning_rate=0.0001, beta_1=0.5, beta_2=0.9)  
3  
4 discriminator_optimizer = tf.keras.optimizers.Adam(  
5     learning_rate=0.0001, beta_1=0.5, beta_2=0.9)  
6  
7 def discriminator_loss(real_img, fake_img):  
8     real_loss = tf.reduce_mean(real_img)  
9     fake_loss = tf.reduce_mean(fake_img)  
10    return fake_loss - real_loss  
11  
12 def generator_loss(fake_img):  
13    return -tf.reduce_mean(fake_img)
```

- Los métodos **evaluate_D** y **evaluate_G** se encargan de evaluar el modelo tal y como se realizó para el modelo DCGAN.
- El método **gradient_penalty** calcula la penalización del gradiente a partir del tamaño del batch, una imagen generada y una real. Primero, construimos un vector aleatorio del tamaño del batch a partir de una distribución normal de media 0 y desviación típica 1. Ese vector será multiplicado por la diferencia que existe entre la imagen falsa y la real, y esto se le sumará a la imagen real. De esta forma conseguimos una imagen interpolada que le introduciremos al discriminador con objeto de obtener la probabilidad de que sea real. Así podremos calcular su gradiente y la penalización de dicho

gradiente. Para calcular la penalización del gradiente debemos de calcular su norma y luego hallar la media de restarle 1 a la norma y elevarla al cuadrado. Matemáticamente estaríamos calculando $\mathbb{E}[(\|\nabla_x D(x)\|_2 - 1)^2]$ como se puede observar con más detalle en [GAA⁺17].

- El método `train_step` se encargará del entrenamiento del modelo WGAN-GP y su funcionamiento consiste:

1. Como se va a entrenar al discriminador `d_steps` veces, en cada paso se realizará lo siguiente.
 - Crear vectores aleatorios en el espacio latente del mismo modo que se realizó en el entrenamiento de la clase GAN anterior.
 - La función `tf.GradientTape()` la denotamos como `tape` y nos servirá para calcular el gradiente. Esta función es útil para implementar el algoritmo de retropropagación en el entrenamiento de redes neuronales.

```

1 with tf.GradientTape() as tape:
2

```

- Vamos a generar la imagen falsa y obtener tanto los logits de la imagen real como de la falsa que acabamos de generar. Esto se consigue evaluando al discriminador con las dos imágenes

```

1
2 fake_images = self.generator(random_latent_vectors, training=
3     True)
4 fake_logits = self.discriminator(fake_images, training=True)
5 real_logits = self.discriminator(real_images, training=True)
6

```

- A partir de los logits calculamos la pérdida del discriminador.

```

1 d_cost = self.d_loss_fn(real_img=real_logits, fake_img=
2     fake_logits)
3

```

- Calculamos la penalización del gradiente haciendo uso del método implementado con objeto de añadirse la penalización a la pérdida del discriminador.

```

1 gp = self.gradient_penalty(batch_size, real_images,
2     fake_images)
3 d_loss = d_cost + gp * self.gp_weight

```

- Finalmente, se calcula el gradiente de la función de pérdida y actualizamos los pesos del discriminador usando su optimizador. El cálculo del gradiente y la aplicación del optimizador es similar al modelo DCGAN(4).

```

1 d_gradient = tape.gradient(d_loss, self.discriminator.
2     trainable_variables)
3 self.d_optimizer.apply_gradients(zip(d_gradient, self.
4     discriminator.trainable_variables))

```

7. Experimentación y resultados

2. Una vez se halla entrenado al discriminador d_steps veces, entonces toca entrenar al generador. Primero, se crean nuevos vectores aleatorios en el espacio latente. Después, generaremos imágenes a partir de estos vectores y determinaremos sus logits. De esta forma, calculamos la función de pérdida del generador evaluada en los logits de la imagen generada. Luego, se realiza un comportamiento análogo al entrenamiento del discriminador pero actualizando los pesos del generador, calculando su gradiente y aplicando su optimizador para actualizar sus pesos.
3. Finalmente, actualizamos los valores de las funciones de pérdida de los modelos y los mostramos por pantalla

Su comportamiento difiere del modelo GAN en dos aspectos fundamentales: por un lado se necesita añadir un término de penalización al gradiente de la pérdida del discriminador y por otro lado se entrena al discriminador un número de pasos antes de entrenar al generador. Otra diferencia a resaltar es que ahora usaremos etiquetas 1 y -1 que hemos denotado como logits mientras que antes nuestras etiquetas eran 0 y 1, esto se consigue eliminando la activación sigmoidea de la arquitectura del discriminador. Además, podemos apreciar cuando hemos declarado las funciones de pérdida que estamos usando la pérdida de Weierstrass especificada en la [Tabla 3.1](#).

7.1.2. Datos

Nuestros modelos se probaran para las siguientes bases de datos.

7.1.2.1. MNIST

La base de datos MNIST(Base de datos del Instituto Nacional de Estándares y Tecnología) está formada por dígitos manuscritos que se utiliza comúnmente para el entrenamiento de varios sistemas de procesamiento de imágenes en el campo del aprendizaje automático. Fue desarrollada por Yann LeCun, Corinna Cortes y Christopher Burges para la evaluación de modelos de aprendizaje sobre el problema de la clasificación de los dígitos escritos a mano. Cada imagen tiene 28x28 píxeles, el conjunto de entrenamiento está formado por 60000 imágenes y el conjunto de prueba por 10000 imágenes.

Para cargar este conjunto de datos se ha usado la biblioteca de aprendizaje profundo de Keras. A continuación, se le cambia el tamaño a las imágenes para que formen un tensor de tamaño 28x28x1 y cambiamos los valores a coma flotante ya que la matriz de píxeles almacenaba enteros con objeto de poder normalizarlos. Además, se le va a añadir algo de ruido al conjunto de prueba. Por otra parte, como los valores de los píxeles se encuentran entre 0 y 255, tenemos que dividir entre 255 para normalizarlos al rango 0 y 1. Finalmente, se baraja el conjunto de datos para no tener clases contiguas y nos quedamos con una parte del conjunto para entrenar nuestros modelos.



Figura 7.1.: Dígitos de MNIST

7.1.2.2. CelebA

La base de datos CelebA es un conjunto de datos de atributos faciales a gran escala con más de 200000 imágenes de celebridades, cada una con 40 anotaciones de atributos. Las imágenes en este conjunto de datos cubren grandes variaciones de poses y distintos fondos. En resumen, tenemos 10177 identidades de celebridades, 202599 imágenes de rostros del tamaño $178 \times 218 \times 3$ y 40 etiquetas binarias por imagen que indican atributos faciales como el color del cabello, el género y la edad. Este conjunto de datos se ha usado en las siguientes tareas de visión por computador: reconocimiento de atributos faciales, reconocimiento facial, detección de rostros, localización de puntos de referencia(o partes faciales) y edición y síntesis de rostros.

El conjunto de datos lo vamos a preprocesar con la función `image_dataset_from_directory` de Keras.preprocessing con objeto de construir los conjuntos de train y test a partir de las imágenes alojadas en un directorio. Vamos a redimensionar las imágenes para que tengan un tamaño de $64 \times 64 \times 3$ para ello especificamos el argumento `image_size` y usaremos un tamaño de batch de 32 con objeto de quedarnos con 6332 imágenes. Para que en cada ejecución se tomen 6332 aleatorias se especifica el argumento `shuffle` a True. Una vez realizado esto se normalizarán las imágenes al rango $[0, 1]$ como se realizó con la base de datos MNIST. A continuación, veamos el resultado de preprocesar una imagen.



Figura 7.2.: Imagen de CelebA

7.1.3. Configuración de las redes neuronales

7.1.3.1. MNIST

Tanto para el modelo DCGAN como para el modelo WGAN-GP se realizan los siguientes experimentos.

- **Experimento 1: MNIST BÁSICO**

Tenemos que definir el modelo generador y discriminador que utilizaremos en nuestro modelo GAN, para ello se ha usado parte de la implementación descrita en [Cho18]. El generador toma un vector z de tamaño 128 del espacio latente y lo conecta con una capa totalmente conectada de tamaño 6272 para a continuación transformarlo a un tensor de tamaño $7 \times 7 \times 128$. Una vez tengamos el tensor, se van a introducir tres capas convolucionales con función de activación LeakyReLU. La arquitectura diseñada para el generador podemos apreciarla en la siguiente imagen y podemos observar como finalmente la salida es una imagen de tamaño $28 \times 28 \times 1$.

7. Experimentación y resultados

Model: "generator"		
Layer (type)	Output Shape	Param #
dense_1 (Dense)	(None, 6272)	889088
reshape (Reshape)	(None, 7, 7, 128)	0
conv2d_transpose (Conv2DTranspose)	(None, 14, 14, 128)	147584
leaky_re_lu_2 (LeakyReLU)	(None, 14, 14, 128)	0
conv2d_transpose_1 (Conv2DTranspose)	(None, 28, 28, 64)	73792
leaky_re_lu_3 (LeakyReLU)	(None, 28, 28, 64)	0
conv2d_2 (Conv2D)	(None, 28, 28, 1)	577
<hr/>		
Total params: 1,031,041		
Trainable params: 1,031,041		
Non-trainable params: 0		

Figura 7.3.: Generador MNIST BÁSICO

El discriminador se define a partir de una entrada de tamaño $28 \times 28 \times 1$, introduciendo dos capas convolucionales con función de activación LeakyReLU y finalmente introduciendo una capa totalmente conectada junto con Dropout. A continuación, veamos su arquitectura.

Model: "discriminator"		
Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 14, 14, 64)	640
leaky_re_lu (LeakyReLU)	(None, 14, 14, 64)	0
conv2d_1 (Conv2D)	(None, 7, 7, 128)	73856
leaky_re_lu_1 (LeakyReLU)	(None, 7, 7, 128)	0
flatten (Flatten)	(None, 6272)	0
dropout (Dropout)	(None, 6272)	0
dense (Dense)	(None, 1)	6273
<hr/>		
Total params: 80,769		
Trainable params: 80,769		
Non-trainable params: 0		

Figura 7.4.: Discriminador MNIST BÁSICO

Ya podemos configurar la GAN que encadena el generador y discriminador y así nuestro modelo DCGAN y WGAN-GP se define como:

```

1 gan = GAN(discriminator=discriminator, generator=generator, latent_dim
   =128)
2 wgan = WGAN(discriminator=discriminator, generator=generator, latent_dim
   =128, discriminator_extra_steps=5,)
```

■ Experimento 2: MNIST MEJORADO

Nos hemos dado cuenta que la pérdida del generador empieza a incrementar cuando la pérdida del discriminador tiende a cero. Para solucionar esto se ha propuesto realizar algunos cambios en el modelo del discriminador como son reducir la tasa de aprendizaje del optimizador del discriminador e incrementar la tasa de dropout. Así reducimos el aprendizaje del discriminador con objeto de que los dos modelos aprendan a la vez y alcancen valores de pérdida igualados, ya que anteriormente nuestro

discriminador aprendía mucho más que nuestro generador.

La arquitectura del nuevo discriminador se puede ver en la siguiente imagen que es análoga a la anterior, en lo único que difiere es que la tasa de dropout ahora es de 0.4 y antes era de 0.2 y por otra parte nuestro modelo discriminador se compilará con un optimizador Adam con tasa $1e - 9$ (antes la tasa era de $1e - 3$).

Model: "discriminator2"		
Layer (type)	Output Shape	Param #
conv2d_3 (Conv2D)	(None, 14, 14, 64)	640
leaky_re_lu_4 (LeakyReLU)	(None, 14, 14, 64)	0
conv2d_4 (Conv2D)	(None, 7, 7, 128)	73856
leaky_re_lu_5 (LeakyReLU)	(None, 7, 7, 128)	0
flatten_1 (Flatten)	(None, 6272)	0
dropout_1 (Dropout)	(None, 6272)	0
dense_2 (Dense)	(None, 1)	6273

Total params:	80,769
Trainable params:	80,769
Non-trainable params:	0

Figura 7.5.: Discriminador MNIST MEJORADO

■ Experimento 3: MNIST COMPLEJO

Aquí se propone una red más compleja con muchas más capas de convolución e introduce capas de BatchNormalization. Las arquitecturas del generador y discriminador se pueden apreciar a continuación.

Layer (type)	Output Shape	Param #
dense_1 (Dense)	(None, 100352)	12945408
reshape (Reshape)	(None, 28, 28, 128)	0
conv2d_3 (Conv2D)	(None, 28, 28, 32)	36896
batch_normalization (BatchN ormalization)	(None, 28, 28, 32)	128
leaky_re_lu_3 (LeakyReLU)	(None, 28, 28, 32)	0
average_pooling2d (AverageP ooling2D)	(None, 14, 14, 32)	0
conv2d_4 (Conv2D)	(None, 14, 14, 64)	18496
batch_normalization_1 (Bac hNormalization)	(None, 14, 14, 64)	256
leaky_re_lu_4 (LeakyReLU)	(None, 14, 14, 64)	0
average_pooling2d_1 (Averag ePooling2D)	(None, 7, 7, 64)	0
conv2d_5 (Conv2D)	(None, 7, 7, 128)	73856
batch_normalization_2 (Bac hNormalization)	(None, 7, 7, 128)	512
leaky_re_lu_5 (LeakyReLU)	(None, 7, 7, 128)	0
conv2d_6 (Conv2D)	(None, 7, 7, 128)	147584
leaky_re_lu_6 (LeakyReLU)	(None, 7, 7, 128)	0
up_sampling2d (UpSampling2D)	(None, 14, 14, 128)	0
conv2d_7 (Conv2D)	(None, 14, 14, 64)	73792
leaky_re_lu_7 (LeakyReLU)	(None, 14, 14, 64)	0
up_sampling2d_1 (UpSampling 2D)	(None, 28, 28, 64)	0
conv2d_8 (Conv2D)	(None, 28, 28, 1)	577

Total params:	13,297,595
Trainable params:	13,297,057
Non-trainable params:	448

Figura 7.6.: Generador MNIST COMPLEJO

7. Experimentación y resultados

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 14, 14, 64)	640
leaky_re_lu (LeakyReLU)	(None, 14, 14, 64)	0
dropout (Dropout)	(None, 14, 14, 64)	0
conv2d_1 (Conv2D)	(None, 7, 7, 128)	73856
leaky_re_lu_1 (LeakyReLU)	(None, 7, 7, 128)	0
dropout_1 (Dropout)	(None, 7, 7, 128)	0
conv2d_2 (Conv2D)	(None, 7, 7, 256)	295168
leaky_re_lu_2 (LeakyReLU)	(None, 7, 7, 256)	0
dropout_2 (Dropout)	(None, 7, 7, 256)	0
flatten (Flatten)	(None, 12544)	0
dense (Dense)	(None, 1)	12545

=====

Total params: 382,209

Trainable params: 382,209

Non-trainable params: 0

Figura 7.7.: Discriminador MNIST COMPLEJO

■ Experimento 4: MNIST MOBILENET

Vamos a definir nuestro modelo discriminador con una arquitectura que ya viene pre-configurada en Keras, llamada MobileNet.

Esta arquitectura tiene como limitación que acepta imágenes de tamaño mínimo 32×32 , esto es, el parámetro `input_shape` de MobileNet no debe ser inferior a 32. Por tanto, tenemos que volver a leer nuestros datos de MNIST y redimensionarlos de 28×28 a 32×32 . A continuación, podemos apreciar el código usado para generar esta arquitectura.

```

1 discriminator4=tf.keras.applications.MobileNet(
2     input_shape=(32,32,1),
3     alpha=1.0,
4     depth_multiplier=1,
5     dropout=0.5,
6     include_top=False,
7     weights=None,
8     input_tensor=None,
9     pooling=None,
10    classes=2,
11    classifier_activation="sigmoid"
12 )

```

El generador es una versión adaptada del generador definido en Experimento 3, de forma que produce imágenes de tamaño $32 \times 32 \times 1$.

Layer (type)	Output Shape	Param #
<hr/>		
dense_10 (Dense)	(None, 131072)	16908288
reshape_2 (Reshape)	(None, 32, 32, 128)	0
conv2d_24 (Conv2D)	(None, 32, 32, 32)	36896
batch_normalization_3 (Batch Normalization)	(None, 32, 32, 32)	128
leaky_re_lu_24 (LeakyReLU)	(None, 32, 32, 32)	0
average_pooling2d_2 (AveragePooling2D)	(None, 16, 16, 32)	0
conv2d_25 (Conv2D)	(None, 16, 16, 64)	18496
batch_normalization_4 (Batch Normalization)	(None, 16, 16, 64)	256
leaky_re_lu_25 (LeakyReLU)	(None, 16, 16, 64)	0
average_pooling2d_3 (AveragePooling2D)	(None, 8, 8, 64)	0
conv2d_26 (Conv2D)	(None, 8, 8, 128)	73856
batch_normalization_5 (Batch Normalization)	(None, 8, 8, 128)	512
leaky_re_lu_26 (LeakyReLU)	(None, 8, 8, 128)	0
conv2d_27 (Conv2D)	(None, 8, 8, 128)	147584
leaky_re_lu_27 (LeakyReLU)	(None, 8, 8, 128)	0
up_sampling2d_2 (UpSampling2D)	(None, 16, 16, 128)	0
conv2d_28 (Conv2D)	(None, 16, 16, 64)	73792
leaky_re_lu_28 (LeakyReLU)	(None, 16, 16, 64)	0
up_sampling2d_3 (UpSampling2D)	(None, 32, 32, 64)	0
conv2d_29 (Conv2D)	(None, 32, 32, 1)	577
<hr/>		
Total params:	17,260,385	
Trainable params:	17,259,937	
Non-trainable params:	448	

Figura 7.8.: Generador MNIST MOBILENET

7. Experimentación y resultados

7.1.3.2. CelebA

Se realizarán los siguientes experimentos con el modelo WGAN-GP.

■ Experimento 1: CELEBA BÁSICO

Para definir el modelo generador y discriminador que utilizaremos en nuestro modelo se ha optado por arquitecturas sencillas para ver como funcionan sin tener que ajustar demasiados parámetros.

El generador toma un vector z de tamaño 128 del espacio latente y lo conecta con una capa totalmente conectada de tamaño 32768 para a continuación transformarlo a un tensor de tamaño $16 \times 16 \times 128$. Una vez tengamos el tensor, se van a introducir tres capas convolucionales con función de activación LeakyReLU. La arquitectura diseñada para el generador podemos apreciarla en la siguiente imagen y podemos observar como finalmente la salida es una imagen de tamaño $64 \times 64 \times 3$.

Layer (type)	Output Shape	Param #
dense_1 (Dense)	(None, 32768)	4227072
reshape (Reshape)	(None, 16, 16, 128)	0
conv2d_transpose (Conv2DTra nspose)	(None, 32, 32, 128)	147584
leaky_re_lu_2 (LeakyReLU)	(None, 32, 32, 128)	0
conv2d_transpose_1 (Conv2DT ranspose)	(None, 64, 64, 64)	73792
leaky_re_lu_3 (LeakyReLU)	(None, 64, 64, 64)	0
conv2d_2 (Conv2D)	(None, 64, 64, 3)	1731
<hr/>		
Total params: 4,456,179		
Trainable params: 4,456,179		
Non-trainable params: 0		

Figura 7.9.: Generador CELEBA BÁSICO

El discriminador se define a partir de una imagen de entrada de tamaño $64 \times 64 \times 3$, introduciendo dos capas convolucionales con función de activación LeakyReLU y finalmente introduciendo una capa totalmente conectada junto con Dropout. A continuación, veamos su arquitectura.

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 32, 32, 64)	1792
leaky_re_lu (LeakyReLU)	(None, 32, 32, 64)	0
conv2d_1 (Conv2D)	(None, 16, 16, 128)	73856
leaky_re_lu_1 (LeakyReLU)	(None, 16, 16, 128)	0
flatten (Flatten)	(None, 32768)	0
dropout (Dropout)	(None, 32768)	0
dense (Dense)	(None, 1)	32769
<hr/>		
Total params: 108,417		
Trainable params: 108,417		
Non-trainable params: 0		

Figura 7.10.: Discriminador CELEBA BÁSICO

■ Experimento 2: CELEBA MEJORADO_1

Con objeto de alcanzar mejores resultados se han realizado los mismos cambios que en el experimento 2 de la base de datos MNIST ([7.1.3.1](#)) pero sobre la arquitectura que

acabamos de definir. Esto es, el discriminador aumentará su tasa de dropout a 0.4 y se entrenará con el optimizador Adam con tasa $1e - 6$, antes se utilizó una tasa de 0.0001.

- **Experimento 3: CELEBA MEJORADO_2**

Misma arquitectura y optimizadores que el experimento anterior, únicamente difiere en que será entrenado para un número mayor de épocas.

7.1.4. Herramientas

Se ha usado la herramienta de Google Colab y el lenguaje de programación Python para llevar a cabo toda la experimentación. Google Colab, también conocido como Colaboratory, es un producto de Google Research que permite a cualquier usuario escribir y ejecutar código arbitrario de Python en el navegador. Es especialmente adecuado para tareas de aprendizaje automático, análisis de datos y educación. La decisión de usar Google Colab ha sido por las ventajas que presenta ya que nos proporciona una GPU gratuita, permite almacenar cuadernos Python en google Drive, no necesita hacer ninguna configuración y permite compartir el código de forma fácil.

En cuanto a bibliotecas de Python se ha usado Keras para el diseño de las redes neuronales usadas en los distintos experimentos. Keras es una biblioteca de redes neuronales de código abierto en Python que vamos a ejecutar sobre Tensorflow y esta diseñada para hacer posible la experimentación con redes de Aprendizaje Profundo. Tensorflow es una biblioteca de código libre para la computación numérica, desarrollada por Google Brain para aplicaciones de aprendizaje automático y de redes neuronales profundas, y que permite principalmente procesar fácilmente matrices o tensores. Esta biblioteca provee herramientas de alto nivel donde nos encontramos con la biblioteca de Python mencionada anteriormente, Keras. Podemos observar que si se quiere programar una nueva red neuronal o realizar cálculos muy costosos con matrices se deberá usar Tensorflow pero para usar alguna red neuronal de las que ya existen es mejor usar Keras.

También se ha usado la herramienta Wandb.ai que permite el seguimiento de los distintos experimentos y la evaluación de los modelos. Es una plataforma de aprendizaje automático que permite a los desarrolladores construir modelos más rápidos ya que permite rastrear rápidamente experimentos, iterar sobre conjuntos de datos, evaluar el rendimiento del modelo, reproducir modelos, visualizar gráficos... En nuestra experimentación se ha utilizado para mostrar gráficos que determinan como evoluciona las funciones de pérdida del generador y discriminador durante el entrenamiento del modelo.

7.2. Resultados

En esta sección se pretende mostrar los resultados obtenidos después de realizar distintas pruebas cambiando determinados parámetros, arquitecturas, funciones, número de imágenes de entrenamiento hasta conseguir resultados destacables. Básicamente, se pretende mostrar una experimentación que refleja todo el marco teórico estudiado en este trabajo.

Sabemos que el modelo WGAN-GP produce resultados muy buenos, con la experimentación realizada para MNIST podemos afirmar que el modelo WGAN-GP obtiene resultados

7. Experimentación y resultados

mejores que DCGAN.

Por otra parte, esta experimentación pretende mostrar como un modelo GAN es capaz de generar caras falsas haciendo uso de la base de datos CelebA. Se ha optado por usar el modelo WGAN-GP ya que con MNIST se ha evidenciado que obtiene mejores resultados que DCGAN. Por tanto, se llevarán a cabo distintos experimentos con el modelo WGAN-GP que nos generará caras falsas a partir de la base de datos CelebA.

7.2.1. MNIST

En nuestra implementación se ha ejecutado cada modelo GANs con 250 imágenes seleccionadas de forma aleatoria, se ha entrenado cada modelo con 75 épocas y se ha usado el optimizador Adam.

7.2.1.1. Modelo DCGAN

Vamos a visualizar en las siguientes tablas los valores de las funciones de pérdida del discriminador (D) y del generador (G) tanto en el entrenamiento como en la evaluación de los distintos experimentos.

En el experimento 1 se utiliza tanto para el generador como para el discriminador un learning rate de $1e - 3$ para el optimizador Adam. Anteriormente, se comentó la necesidad de reducir la tasa de learning rate del discriminador que a partir del experimento 2 tomará el valor de $1e - 9$ y así podemos ver a continuación como se igualan las pérdidas. En el experimento 3 podemos apreciar que se consiguen casi igualar las pérdidas porque este experimento trata de mejorar al experimento anterior usando una arquitectura más refinada. El experimento 4 tiene como objetivo ver los resultados que proporciona el usar una arquitectura preconfigurada en Keras aunque los resultados empeoran.

Experimento	Loss D	Loss G
1: MNIST BÁSICO	0.02145	5.772
2: MNIST MEJORADO	0.6926	0.7251
3: MNIST COMPLEJO	0.6845	0.6968
4: MNIST MOBILENET	8.1085	0.0000

Tabla 7.1.: Entrenamiento

En la evaluación del discriminador se obtienen los siguientes valores donde *loss real* hace referencia al valor medio de la función de pérdida obtenido al evaluar el modelo discriminador sobre imágenes reales y *loss fake* al valor medio obtenido al evaluar nuestro discriminador con imágenes falsas.

Experimento	Loss real	Loss fake
1: MNIST BÁSICO	4.05	4.89
2: MNIST MEJORADO	0.66	0.72
3: MNIST COMPLEJO	0.70	0.69
4: MNIST MOBILENET	15.42	0.0

Tabla 7.2.: Evaluación del discriminador

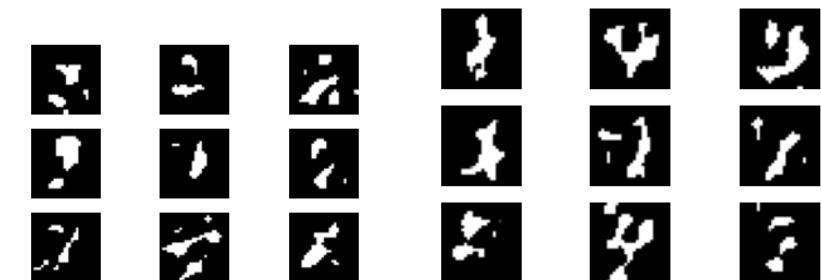
Al evaluar cada generador en los cuatro experimentos vemos como se generan las siguientes imágenes.



(a) Experimento 1: MNIST BÁSICO

(b) Experimento 2: MNIST MEJORADO

Figura 7.11.: Experimentos DCGAN



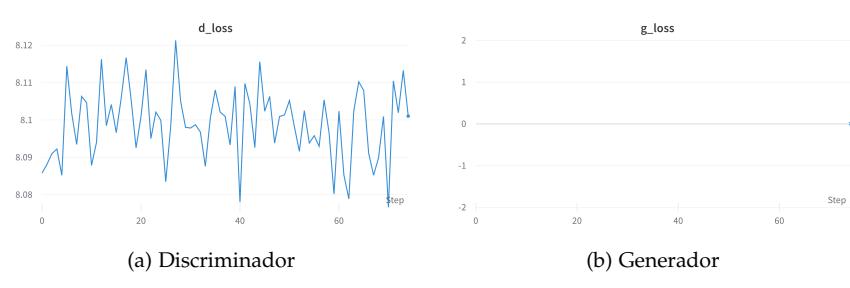
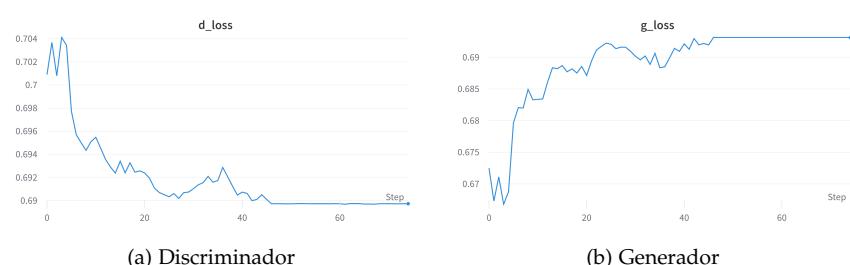
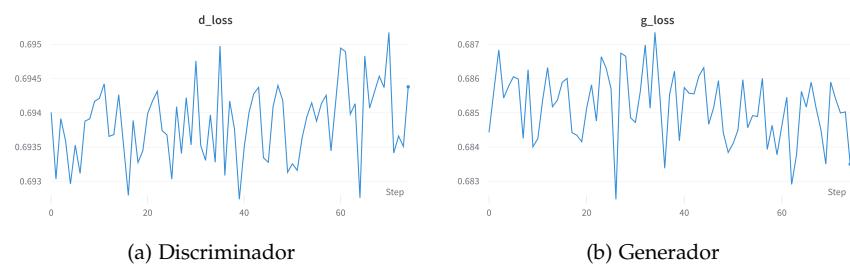
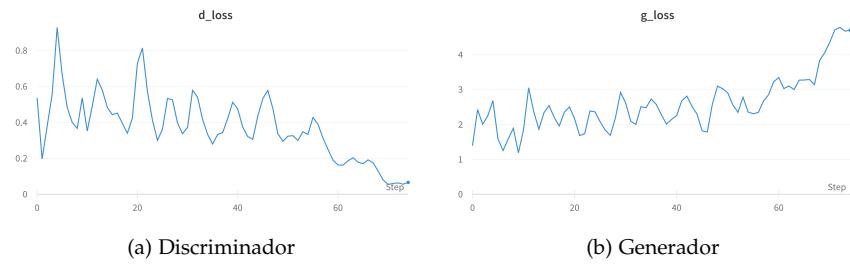
(a) Experimento 3: MNIST COMPLEJO

(b) Experimento 4: MNIST MOBILENET

Figura 7.12.: Experimentos DCGAN

Haciendo uso de la herramienta wandb podemos observar como se comportan las funciones de pérdida del generador y discriminador durante el entrenamiento. Se puede apreciar como en el experimento 1 la pérdida del generador empieza a incrementar cuando la pérdida del discriminador tiende a cero y que a partir del experimento 2 deja de suceder debido a los cambios introducidos en la tasa de dropout y en el learning rate. Asimismo, tanto en el experimento 2 como en el experimento 3 las pérdidas oscilan relativamente poco ya que se mantienen entorno al 0.6 y 0.7. Por otra parte, las gráficas del experimento 4 no arroja resultados importantes ya que se ha probado para ver como funciona una GAN con un modelo discriminador implementado con una arquitectura preconfigurada en Keras.

7. Experimentación y resultados



7.2.1.2. Modelo WGAN-GP

Se han llevado a cabo los mismos experimentos con el mismo número de imágenes, épocas y mismos optimizadores para poder comparar ambos modelos. Después de varias pruebas se ha decidido entrenar 5 veces al discriminador entre cada actualización del generador, entonces `discriminator_extra_steps` tomará el valor de 5. Al contrastar las imágenes generadas por ambos modelos podemos observar como WGAN-GP obtienen mejores resultados que DCGAN.

A continuación, vemos las tablas con los valores de las funciones de pérdida del discriminador (D) y del generador (G) tanto en el entrenamiento como en la evaluación de los distintos experimentos.

Experimento	Loss D	Loss G
1: MNIST BÁSICO	-1.1622	6.5486
2: MNIST MEJORADO	-1.76464	2.2456
3: MNIST COMPLEJO	-0.84	1.88
4: MNIST MOBILENET	80.1070	-0.3094

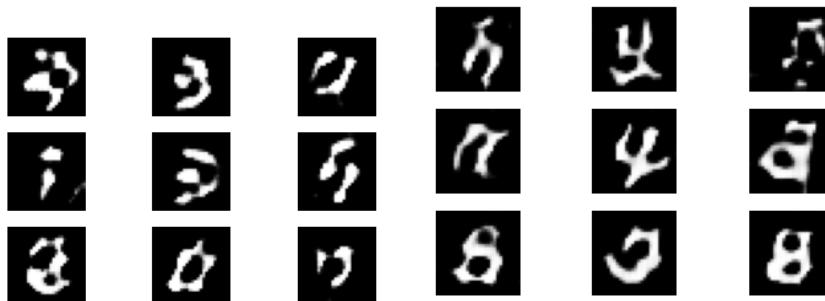
Tabla 7.3.: Entrenamiento

En la evaluación del discriminador se obtienen los siguientes valores.

Experimento	Loss real	Loss fake
1: MNIST BÁSICO	-7.27	-7.30
2: MNIST MEJORADO	-1.02	-1.46
3: MNIST COMPLEJO	-1.72	-1.20
4: MNIST MOBILENET	-0.77	0.22

Tabla 7.4.: Evaluación del discriminador

Al evaluar cada generador en los cuatro experimentos vemos como se generan las siguientes imágenes.



(a) Experimento 1: MNIST BÁSICO

(b) Experimento 2: MNIST MEJORADO

Figura 7.17.: Experimentos WGAN-GP

7. Experimentación y resultados

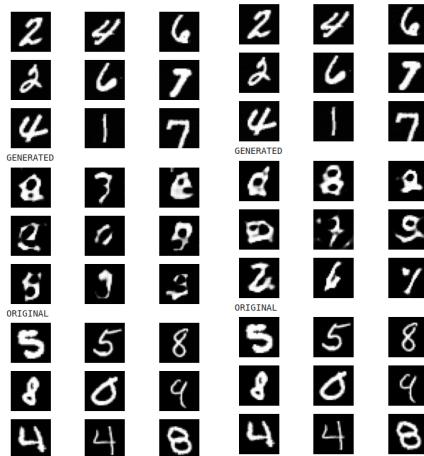


(a) Experimento 3: MNIST COMPLEJO

(b) Experimento 4: MNIST MOBILENET

Figura 7.18.: Experimentos WGAN-GP

Podemos observar como nuestro modelo es capaz de aprender a reconocer dígitos y a generar dígitos falsos a partir un entrenamiento con pocas imágenes. A continuación, veamos los dígitos que es capaz de producir los generadores del experimento 2 y 3.



(a) MNIST MEJORADO (b) MNIST COMPLEJO

Figura 7.19.: Imágenes generadas con WGAN-GP

Haciendo uso de la herramienta wandb podemos observar como se comportan las funciones de pérdida del generador y discriminador durante el entrenamiento. Veamos que ocurre la misma situación que con los experimentos realizados para el modelo DCGAN. Primero, la pérdida del generador empieza a incrementar cuando la pérdida del discriminador tiende a cero y a partir del experimento 2 conseguimos que las pérdidas se estabilicen y tomen valores más cercanos.

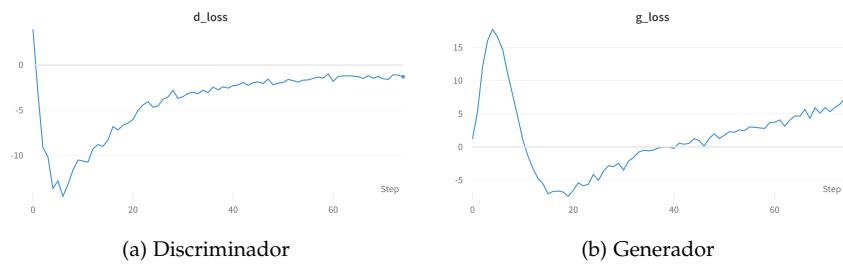


Figura 7.20.: Experimento 1: MNIST BÁSICO

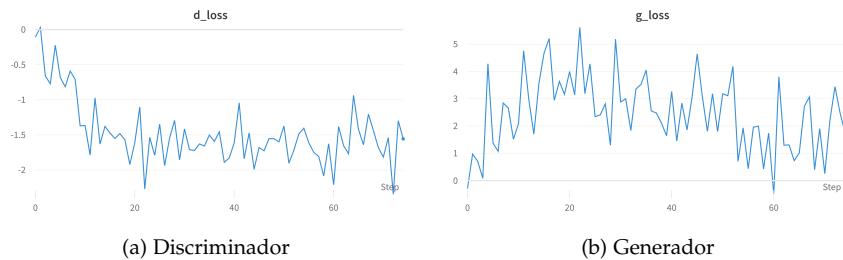


Figura 7.21.: Experimento 2: MNIST MEJORADO

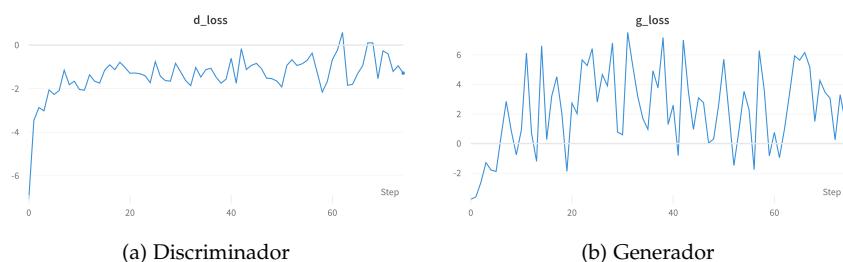


Figura 7.22.: Experimento 3: MNIST COMPLEJO

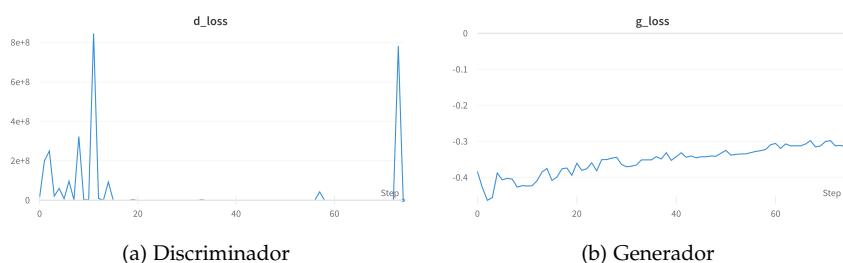


Figura 7.23.: Experimento 4: MNIST MOBILENET

7. Experimentación y resultados

7.2.2. CelebA

Se va a ejecutar el modelo WGAN-GP con 6332 imágenes de entrenamiento, se entrenará durante 30 épocas en los dos primeros experimentos y durante 100 épocas en el tercer experimento, haciendo uso del optimizador Adam en todos los experimentos. Además, se ha decidido entrenar 5 veces al discriminador entre cada actualización del generador.

La decisión de usar este modelo se basa en que con MNIST las imágenes generadas con WGAN-GP presentan una calidad superior a las generadas con el modelo DCGAN.

Primero, se probó el experimento 1, CELEBA BÁSICO, entrenando a nuestro modelo con 100 épocas pero se observó que la pérdida del generador empezaba a oscilar en rangos cada vez mayores conforme se iban ejecutando las épocas, obteniendo en las últimas épocas los siguientes valores para el experimento 1

Época	Loss D	Loss G
95	-0.57	-357.68
96	-0.81	57.48
97	-0.76	572.94
98	-0.43	898.31
99	-1.15	920.25
100	-0.4946	694.43

Tabla 7.5.: Entrenamiento Experimento 1:CELEBA BÁSICO para 100 épocas

Entonces, se optó por disminuir el número de épocas a 30 con objeto de que la pérdida del generador oscile en un rango mejor de valores. Con 30 épocas el rango de valores donde oscila es mucho menor como puede verse en la imagen [7.26a](#).

En segundo lugar, se entrena durante 30 épocas la versión mejorada donde se reduce la tasa de learning rate y se aumenta la tasa de dropout, CELEBA MEJORADO_1 .

Como sabemos que esta mejora consigue que las pérdidas del generador y discriminador se acerquen, entonces hemos optado por entrenar también con 100 épocas, CELEBA MEJORADO_2, ya que la pérdida del generador no oscilará tanto como en el experimento 1. De hecho, conseguimos que las pérdidas oscilen en un rango menor al entrenar con un número mayor de épocas como se puede ver en [7.28b](#) y [7.28a](#). Podemos apreciar como en CELEBA MEJORADO_1 la pérdida del generador oscila en $[-20, 40]$ y la del discriminador en $[-30, 10]$ mientras que en CELEBA MEJORADO_2 la pérdida del generador oscila en el intervalo $[-4, 4]$ y la del discriminador oscila en $[-2, 4]$. Por tanto, el tercer experimento ha conseguido reducir las pérdidas al entrenar en un número mayor de épocas el segundo experimento.

7.2.2.1. Modelo WGAN-GP

Vamos a ver dos tablas donde se presentan los valores de las funciones de pérdida durante el entrenamiento de nuestro modelo WGAN-GP y durante la evaluación.

7.2. Resultados

Experimento	Loss D	Loss G
1: CELEBA BÁSICO	-0.6376	52.5069
2: CELEBA MEJORADO_1	0.2730	-5.5414
3: CELEBA MEJORADO_2	0.0107	-0.5502

Tabla 7.6.: Entrenamiento

Experimento	Loss real	Loss fake
1: CELEBA BÁSICO	-4270762202497024.0	-298.57
2 CELEBA MEJORADO_1	-853829566332928.0	-2.64
3: CELEBA MEJORADO_2	1596498131288064.0	3.03

Tabla 7.7.: Evaluación del discriminador

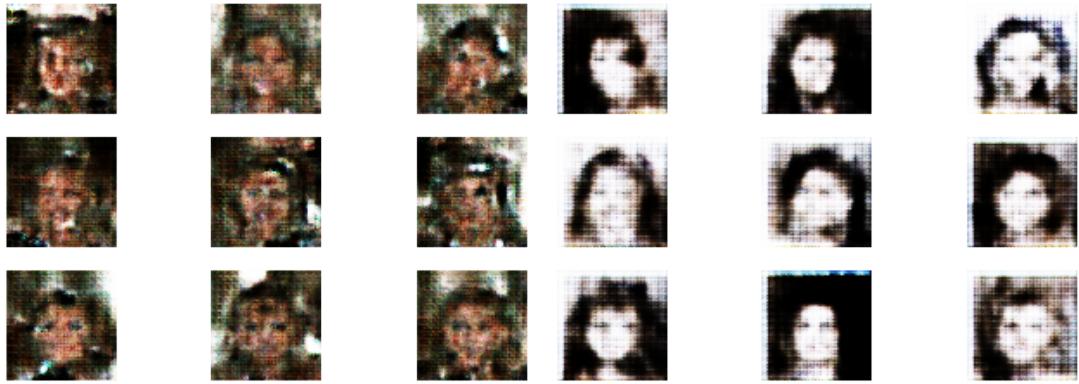
Al evaluar cada generador en los tres experimentos vemos como se generan las siguientes imágenes. De forma visual no se aprecian las mejoras entre los distintos experimentos, por eso se ha optado por estudiar los valores de las funciones de pérdidas del generador y discriminador durante el entrenamiento.



(a) Experimento 1: CELEBA BÁSICO

Figura 7.24.: Experimentos con WGAN-GP

7. Experimentación y resultados

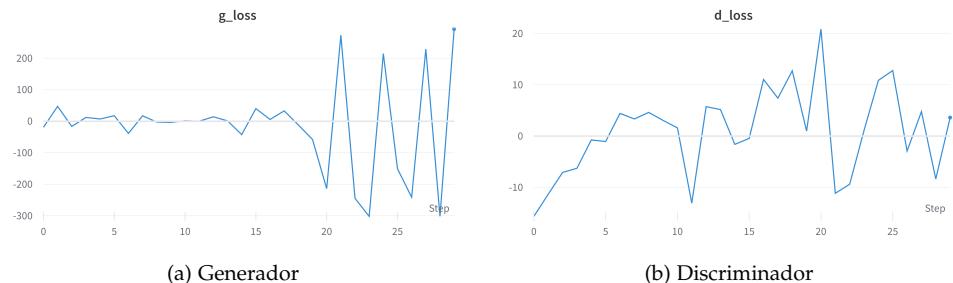


(a) Experimento 2: CELEBA MEJORADO_1

(b) Experimento 3: CELEBA MEJORADO_3

Figura 7.25.: Experimentos con WGAN-GP

Las siguientes gráficas generadas con la herramienta wandb nos permite observar como va variando las funciones de pérdida del discriminador y del generador. En el experimento 1 vemos como la pérdida del generador es inicialmente nula y luego va a oscilar, mientras que el discriminador presenta una pérdida negativa que tenderá a cero y luego comenzará a oscilar. La diferencia que presenta el experimento 2 es que al aumentar la tasa de dropout y reducir la tasa de learning rate se consigue lo que se desea, esto es, conseguimos que las pérdidas se estabilicen y tomen valores más cercanos. De hecho, se puede apreciar como las pérdidas del generador oscilan en un rango mucho menor en el experimento 2 y finalmente toman valores más cercanos, esto es, se logra que se igualen las pérdidas del generador y del discriminador. Con respecto al experimento 3 se consiguen que las pérdidas oscilen en un rango inferior de valores desde el inicio hasta el fin del entrenamiento.



(a) Generador

(b) Discriminador

Figura 7.26.: Experimento 1: CELEBA BÁSICO

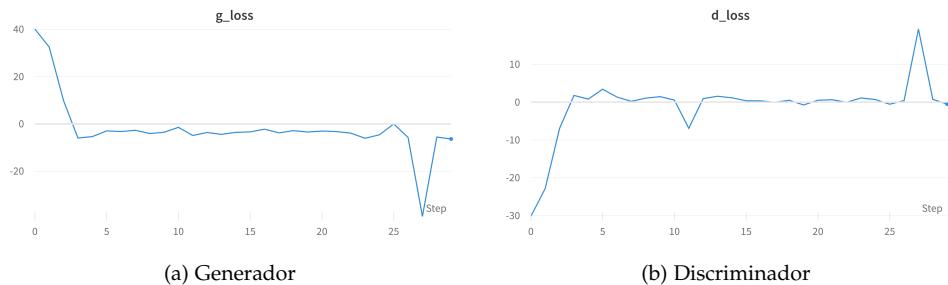


Figura 7.27.: Experimento 2: CELEBA MEJORADO_1

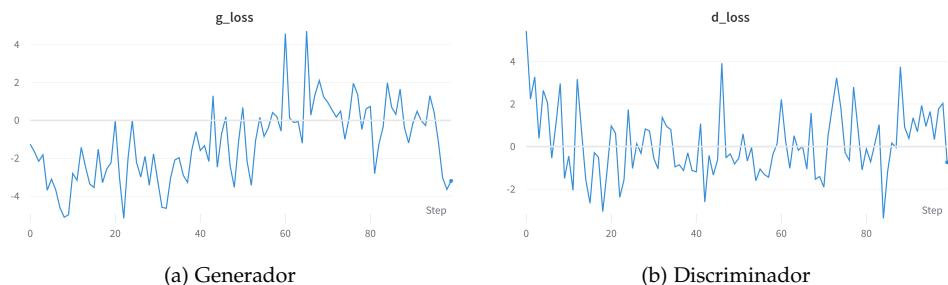


Figura 7.28.: Experimento 3: CELEBA MEJORADO_2

Parte IV.

Conclusiones y trabajo futuro

8. Conclusiones y Trabajo futuro

Finalmente, se van a exponer las conclusiones obtenidas al finalizar este trabajo y una serie de propuestas interesantes que se podría llevar acabo más adelante.

Este trabajo nos ha proporcionado todas las herramientas para la creación de contenido falso a través de modelos GANs. Se ha explicado el funcionamiento y el entrenamiento de las redes generativas adversarias con objeto de comprender totalmente este tipo de red neuronal para poder usarla en la generación de contenido sintético. Luego, se ha estudiado distintas técnicas de manipulación de imágenes faciales como son los métodos de Deepfakes y los métodos para detectar tales manipulaciones. Asimismo, se han expuesto distintos tipos de GANs y se han realizado experimentos con dos tipos de GANs: DCGAN y WGAN-GP.

El apartado matemático nos ha demostrado la existencia de solución para la formulación minmax de una GAN. Así, tenemos asegurado que siempre se alcanza el punto de equilibrio, esto es, un punto donde ambas redes neuronales minimizan su función de pérdida y el modelo converge. Además, se ha detallado el algoritmo de entrenamiento de una GAN básica y su convergencia. Estos aspectos son fundamentales a la hora de desarrollar la implementación de cualquier modelo GAN ya que necesitamos entrenarlo y que exista la convergencia a un punto de equilibrio donde finalice su entrenamiento.

Podemos concluir que este modelo generativo presentan un alto potencial en la creación de contenido falso como se ha experimentado en el capítulo 7 y se ha visto en el capítulo 4 donde se estudia el estado del arte de las creaciones y manipulaciones de Deepfakes. También puede apreciarse que existen distintos tipos de GANs con distinta arquitectura y distintas funciones de pérdida, en nuestra experimentación se optó por usar dos de ellos pero puede implementarse otros tipos de GANs y seguramente se obtendrá resultados bastante notables.

Por otra parte, se podrían usar las bases de datos citadas en el capítulo 4 y así probar a generar contenido falso haciendo uso de cualquier tipo de GANs. Como vías de trabajo futuro se propone realizar otras experimentaciones cambiando las bases de datos y el tipo de GANs con objeto de conseguir distintas manipulaciones faciales. En efecto, se propone generar contenido falso haciendo uso de distintas bases de datos y de distintos modelos GANs donde se podrán hacer distintas pruebas cambiando la función de pérdida, los optimizadores, así como la arquitectura del generador y discriminador.

A. Otras aproximaciones de tipo generativo

Para la generación de contenido sintético con redes neuronales vamos a estudiar también otras aproximaciones de tipo "generativo", como la transferencia de estilo, los autoencoders o las redes generativas estocásticas.

A.1. Transferencia de estilo

Inicialmente pensemos que transferir el estilo de una imagen a otra puede considerarse un problema de transferencia de textura cuyo objetivo es sintetizar una textura de una imagen origen a una imagen destino y restringir la síntesis de texturas para preservar el contenido semántico en la imagen destino. Los algoritmos de transferencia de textura solo usan las características de bajo nivel de la imagen destino, sin embargo un algoritmo de transferencia de estilo debe ser capaz de obtener el contenido semántico de la imagen destino y luego proponer un mecanismo para reproducir la textura de la imagen origen en la imagen destino. Por tanto, necesitamos encontrar representaciones de imágenes que modelen de forma independiente el contenido semántico de su estilo. Las redes neuronales convolucionales consiguen este objetivo y el método de transferencia de estilo se reduce a un problema de optimización dentro de una sola red neuronal. Más bien, nuestro algoritmo de transferencia de estilo combina un modelo de textura paramétrica basado en Redes Neuronales Convolucionales con un método para invertir sus representaciones de imagen.

A.1.1. Representación del contenido semántico

Sea $F^l \in \mathbb{R}^{N_l \times M_l}$ el tensor obtenido en la capa l de nuestra CNN y F_{ij}^l es la activación del i -ésimo filtro en la posición j en la capa l .

Para obtener la representación del contenido de la imagen se va a realizar un descenso de gradiente en una imagen de ruido blanco. Definimos p como la imagen original con P_l su representación de características de la capa l y x como la imagen que se genera con F_l su representación de características en la capa l . Siguiendo esta notación tenemos que la función de pérdida entre las dos representaciones de características se define como

$$\mathcal{L}_{content}(p, x, l) = \frac{1}{2} \sum_{i,j} (F_{ij}^l - P_{ij}^l)^2$$

A continuación, podemos apreciar la derivada de la pérdida con respecto a las activaciones en la capa l que nos será de utilidad para calcular el gradiente con respecto a la imagen x utilizando retropropagación y así ir cambiando la imagen aleatoria hasta conseguir representar en x todo el contenido semántico.

$$\frac{\partial \mathcal{L}_{content}}{\partial F_{ij}^l} = \begin{cases} (F^l - P^l)_{ij} & \text{si } F_{ij}^l > 0 \\ 0 & \text{si } F_{ij}^l < 0 \end{cases}$$

A. Otras aproximaciones de tipo generativo

A.1.2. Representación del estilo

Vamos a usar la matriz de Gram $G^l \in \mathbb{R}^{N_l \times N_l}$ donde G_{ij}^l es el producto interno entre los mapas de características i y j en la capa l .

$$G_{ij}^l = \sum_k F_{ik}^l F_{jk}^l$$

Sea a la imagen original de la que queremos extraer el estilo y x la imagen generada, entonces la contribución de la capa l a la pérdida total es

$$E_l = \frac{1}{4N_l^2 M_l^2} \sum_{i,j} (G_{ij}^l - A_{ij}^l)^2$$

Si definimos unos pesos w_l que hacen referencia a la contribución de cada capa entonces la pérdida total de estilo es

$$\mathcal{L}_{style}(a, x) = \sum_{l=0}^L w_l E_l$$

Al igual que antes, la derivada puede ser calculada y se obtiene

$$\frac{\partial E_l}{\partial F_{ij}^l} = \begin{cases} \frac{1}{N_l^2 M_l^2} ((F^l)^T (G^l - A^l))_{ji} & \text{si } F_{ij}^l > 0 \\ 0 & \text{si } F_{ij}^l < 0 \end{cases}$$

A.1.3. Algoritmo de transferencia de estilo

Consiste en minimizar conjuntamente las dos funciones de pérdida \mathcal{L}_{style} y $\mathcal{L}_{content}$, entonces la función de pérdida que minimizamos es

$$\mathcal{L}_{total}(p, a, x) = \alpha \mathcal{L}_{content}(p, x) + \beta \mathcal{L}_{style}(a, x)$$

Los valores α y β son factores de ponderación de la representación del contenido y del estilo.

Finalmente, vamos a ver los resultados que se obtienen al aplicar este modelo generativo a una fotografía de la orilla del río Neckar en Tubingen(Alemania) y que ha sido extraído de [GEB16] donde se le han aplicado distintos estilos, en concreto cinco representaciones de estilo distintas.



Figura A.1.: Transferencia de Estilo

A.2. Autoencoders

Los autoencoders se componen de dos partes: codificador y decodificador. Veamos la siguiente imagen extraída del libro [Bok19] y que se usará para explicar el funcionamiento básico de este modelo generativo.

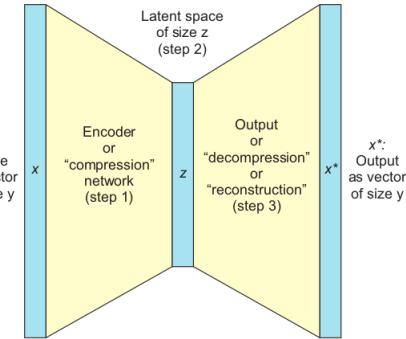


Figura A.2.: Autoencoder

- El espacio latente es una representación oculta y mucho más simple ya que presenta menor dimensión, por ejemplo puede ser un vector de ruido aleatorio. Actúa como paso intermedio en el modelo generativo.
- La red del codificador se encarga de reducir la dimensión de una entrada x que normalmente suele ser una imagen y construye una instancia z del espacio latente. Esto lo lleva a cabo a partir de una red neuronal.
- La red del decodificador reconstruye la imagen original con su dimensión original a partir de una instancia del espacio latente que claramente presenta una dimensión menor. Este es el paso de z a x^* que lo realiza una red neuronal cuya misión es invertir el proceso del codificador.
- Algoritmo de entrenamiento
 1. Tomamos imágenes x y las ejecutamos a través de nuestro autoencoders
 2. Obtenemos la reconstrucción x^*
 3. Calculamos el valor de la función de pérdida, esto es, la diferencia entre x y x^* , $\|x - x^*\|$. Por tanto, hemos determinado nuestra función objetivo que será optimizada haciendo uso del gradiente descendente.

Nuestro objetivo es encontrar los parámetros de las redes del codificador y del decodificador que minimicen la función de pérdida que iremos actualizando a través del algoritmo de descenso del gradiente.

Sabemos que las redes neuronales se pueden aproximar por funciones, entonces imaginemos que tenemos una función de codificador f que genera una representación z del espacio latente, $z = f(x)$, y tenemos una función de decodificador g tal que $g(f(x)) = x^* \approx x$. A continuación, notaremos $r(x) := g(f(x))$ y vamos a estudiar diferentes autocodificadores extraídos del artículo [AB12] donde aparecen estudiados con mucha más profundidad.

A. Otras aproximaciones de tipo generativo

■ Autoencoder contractivo/CAE

Es un autocodificador regularizado que se entrena para minimizar el siguiente error de reconstrucción regularizado:

$$\mathcal{L}_{CAE} = \mathbb{E}[\ell(x, r(x)) + \lambda \left\| \frac{\partial f(x)}{\partial x} \right\|_F^2]$$

■ Autoencoder de eliminación de ruido/DAE

Sea $N(x)$ es una corrupción estocástica de x , este autoencoder se encarga de minimizar la siguiente función de pérdida:

$$\mathcal{L}_{DAE} = \mathbb{E}[\ell(x, r(N(x)))]$$

■ Reconstrucción autoencoder contractiva/RCAE

Es una variación del CAE donde la función de pérdida es la pérdida de reconstrucción al cuadrado más la penalización de la reconstrucción contractiva:

$$\mathcal{L}_{RCAE} = \mathbb{E}[\|r(x) - x\|_2^2 + \lambda^2 \left\| \frac{\partial r(x)}{\partial x} \right\|_F^2]$$

A.3. Otros

A.3.1. Redes generativas estocásticas(GSN)

Las redes generativas estocásticas generalizan a los codificadores automáticos de eliminación de ruido (DAE) y se basan en el aprendizaje del operador de transición de una cadena de Markov donde la distribución estacionaria determina la distribución de datos. El estudio de este modelo generativo puede seguirse en [ABY⁺16].

A.3.2. GPT-3

GPT-3 es un modelo creado por OpenAI que permite generar lenguaje escrito, esto es, el usuario escribe un párrafo y el sistema se encarga de completarlo de la forma más coherente posible. Su aplicación se puede ver en las tecnologías del lenguaje humano, por ejemplo procesamiento de lenguaje, habla, traducción, búsqueda o asistentes virtuales, en la generación de imágenes o audio e incluso en las ayudas a la programación de 'software'. [Dal21]

A.3.3. Dall-E

Fue creado por OpenAI y consiste en un modelo que genera imágenes sintéticas originales correspondientes a un texto de entrada, utilizando un conjunto de datos de parejas texto e imagen. Básicamente, consiste en una versión de 12 mil millones de parámetros de GPT-3. Tiene un conjunto diverso de utilidades donde se incluyen la creación de versiones antropomórficas de animales y objetos, la combinación de conceptos no relacionados de manera plausible, la representación de texto y la aplicación de transformaciones a imágenes existentes. [RPG⁺21]

Bibliografía

Las referencias se listan por orden alfabético. Aquellas referencias con más de un autor están ordenadas de acuerdo con el primer autor.

- [AB12] Guillaume Alain and Yoshua Bengio. What regularized auto-encoders learn from the data generating distribution, 2012. [Citado en pág. 121]
- [ABY⁺16] Guillaume Alain, Yoshua Bengio, Li Yao, Jason Yosinski, Éric Thibodeau-Laufer, Saizheng Zhang, and Pascal Vincent. GSNs: generative stochastic networks. *Information and Inference: A Journal of the IMA*, 5(2):210–249, 03 2016. [Citado en pág. 122]
- [ACB17] Martin Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein generative adversarial networks. In Doina Precup and Yee Whye Teh, editors, *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 214–223. PMLR, 06–11 Aug 2017. [Citado en pág. 29]
- [AI18] AI. 100,000 faces generated by ai. <https://generated.photos/>, 2018. Recurso online. Accedido el 23 de marzo de 2022. [Citado en pág. 40]
- [AI19] Google AI. Contributing Data to Deepfake Detection Research. <https://ai.googleblog.com/2019/09/contributing-data-to-deepfake-detection.html>, 2019. Recurso online. Accedido el 23 de marzo de 2022. [Citado en pág. 42]
- [BGA⁺22] Amit H. Bermano, Rinon Gal, Yuval Alaluf, Ron Mokady, Yotam Nitzan, Omer Tov, Or Patashnik, and Daniel Cohen-Or. State-of-the-Art in the Architecture, Methods and Applications of StyleGAN, 2022. [Citado en págs. 27, 34, and 35]
- [Bok19] Vladimir Bok. *GANs in Action*. Manning Publications, City, 2019. [Citado en págs. 19, 20, 22, 23, 27, and 121]
- [Cho18] Francois Chollet. *Deep learning with Python*. Manning Publications Co, Shelter Island, NY, 2018. [Citado en pág. 97]
- [CS] CS. Cs 60050 machine learning. <https://cs60050.github.io>. Recurso online. Accedido el 23 de marzo de 2022. [Citado en pág. 6]
- [Cyb89] G. Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals, and Systems*, 2(4):303–314, 1989. [Citado en pág. 10]
- [Dal21] Robert Dale. Gpt-3: What's it good for? *Natural Language Engineering*, 27(1):113–118, 2021. [Citado en pág. 122]
- [DHP⁺19] Brian Dolhansky, Russ Howes, Ben Pflaum, Nicole Baram, and Cristian Canton Ferrer. The Deepfake Detection Challenge (DFDC) Preview Dataset, 2019. [Citado en pág. 42]
- [DLS⁺19] Hao Dang, Feng Liu, Joel Stehouwer, Xiaoming Liu, and Anil Jain. On the Detection of Digital Face Manipulation, 2019. [Citado en pág. 40]
- [Fos19] David Foster. *Generative deep learning : teaching machines to paint, write, compose, and play*. O'Reilly Media, Sebastopol, CA, 2019. [Citado en págs. 18 and 27]
- [GAA⁺17] Ishaan Gulrajani, Faruk Ahmed, Martin Arjovsky, Vincent Dumoulin, and Aaron Courville. Improved Training of Wasserstein GANs, 2017. [Citado en págs. 30, 87, and 95]
- [GEB16] Leon A. Gatys, Alexander S. Ecker, and Matthias Bethge. Image style transfer using convolutional neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016. [Citado en pág. 120]

Bibliografía

- [Goo16] Ian Goodfellow. *Deep learning*. The MIT Press, Cambridge, Massachusetts, 2016. [Citado en págs. 3, 19, and 70]
- [GPAM⁺14] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 27. Curran Associates, Inc., 2014. [Citado en pág. 67]
- [KALL17] Tero Karras, Timo Aila, Samuli Laine, and Jaakkko Lehtinen. Progressive Growing of GANs for Improved Quality, Stability, and Variation, 2017. [Citado en pág. 30]
- [Kim17] Phil Kim. *MATLAB deep learning : with machine learning, neural networks and artificial intelligence*. Apress, New York, NY, 2017. [Citado en págs. 4, 14, and 15]
- [KLA19] Tero Karras, Samuli Laine, and Timo Aila. A Style-Based Generator Architecture for Generative Adversarial Networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019. [Citado en pág. 40]
- [KM18] Pavel Korshunov and Sebastien Marcel. DeepFakes: a New Threat to Face Recognition? Assessment and Detection, 2018. [Citado en pág. 42]
- [LCL18] Yuezun Li, Ming-Ching Chang, and Siwei Lyu. In Ictu Oculi: Exposing AI Generated Fake Face Videos by Detecting Eye Blinking, 2018. [Citado en pág. 42]
- [LYS⁺20] Yuezun Li, Xin Yang, Pu Sun, Honggang Qi, and Siwei Lyu. Celeb-DF: A Large-Scale Challenging Dataset for DeepFake Forensics. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2020. [Citado en pág. 42]
- [MO14] Mehdi Mirza and Simon Osindero. Conditional Generative Adversarial Nets, 2014. [Citado en pág. 32]
- [Nie19] Frank Nielsen. On the jensen–shannon symmetrization of distances relying on abstract means. *Entropy*, 21:485, 05 2019. [Citado en pág. 73]
- [NTVR⁺20] Joao Neves, Ruben Tolosana, Ruben Vera-Rodriguez, Vasco Lopes, Hugo Proen  a, and Julian Fierrez. GANprintR: Improved Fakes and Evaluation of the State of the Art in Face Manipulation Detection, 04 2020. [Citado en pág. 40]
- [PCo8] Fernando Perez-Cruz. Kullback-leibler divergence estimation of continuous distributions. In *2008 IEEE International Symposium on Information Theory*, pages 1666–1670, 2008. [Citado en pág. 73]
- [RCV⁺19] Andreas R  ssler, Davide Cozzolino, Luisa Verdoliva, Christian Riess, Justus Thies, and Matthias Nie  ner. FaceForensics++: Learning to Detect Manipulated Facial Images, 2019. [Citado en pág. 42]
- [RMC15] Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks, 2015. [Citado en pág. 28]
- [RPG⁺21] Aditya Ramesh, Mikhail Pavlov, Gabriel Goh, Scott Gray, Chelsea Voss, Alec Radford, Mark Chen, and Ilya Sutskever. Zero-shot text-to-image generation, 2021. [Citado en pág. 122]
- [SGZ⁺16] Tim Salimans, Ian Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, Xi Chen, and Xi Chen. Improved Techniques for Training GANs. In D. Lee, M. Sugiyama, U. Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 29. Curran Associates, Inc., 2016. [Citado en pág. 31]
- [Tao11] Terence Tao. *An introduction to measure theory*. American Mathematical Society, Providence, R.I, 2011. [Citado en pág. 51]
- [TVRF⁺20] Ruben Tolosana, Ruben Vera-Rodriguez, Julian Fierrez, Aythami Morales, and Javier Ortega-Garcia. Deepfakes and beyond: A survey of face manipulation and fake detection. 2020. [Citado en pág. 39]

Bibliografía

- [Wan20] Yang Wang. A mathematical introduction to generative adversarial nets (GAN), 2020. [Citado en págs. [67](#), [80](#), and [82](#)]
- [ZPIE17] Jun-Yan Zhu, Taesung Park, Phillip Isola, and Alexei A. Efros. Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks, 2017. [Citado en pág. [33](#)]