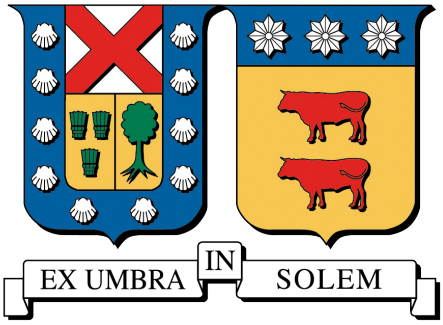
UNIVERSIDAD TÉCNICA FEDERICO SANTA MARÍA
DEPARTAMENTO DE ELECTRÓNICA
VALPARAÍSO - CHILE



REDES GENERATIVAS ADVERSARIAS HIPERBÓLICAS

DIEGO FACUNDO LAZCANO ARCOS

MEMORIA PARA OPTAR AL TÍTULO DE
INGENIERO CIVIL ELECTRÓNICO Y MAGÍSTER EN CIENCIAS DE LA INGENIERÍA ELECTRÓNICA

PROFESOR GUÍA : Dr. Werner Creixell

MARZO 2021

TÍTULO DE LA TESIS:

Redes Generativas Adversarias Hiperbólicas

AUTOR:

Diego Facundo Lazcano Arcos

TRABAJO DE TESIS, presentado en cumplimiento parcial de los requisitos para el título de Ingeniero Civil Electrónico y el grado de Magíster en Ciencias de la Ingeniería Electrónica de la Universidad Técnica Federico Santa María.

Dr. Werner Creixell _____

Dr. Mauricio Araya _____

Dr. Moulay Akhloufi _____

Valparaíso, Marzo de 2021.

Dedicado a mi familia y amigos

AGRADECIMIENTOS

Agradezco a mi familia, ya que sin ellos no estaría donde estoy ahora, gracias por toda su entrega y amor, además de sus enseñanzas de vida, de perseverar ante las adversidad, responsabilidad, respeto y moral.

También agradezco de todo corazón a Paula, mi fuente de paz, y amor. Gracias por estar ahí, en los momentos más difíciles y en los más lindos.

Por otro lado, agradezco a todos mis amigos tanto dentro como fuera de la universidad, entre ellos destaco a mi grupo de amigos más espaciales, Felipe Arancibia, Pablo "Papito"Hernández, y al gran Nicolás Fredes. Donde, al que más agradezco es a Nicolas, compañero de innumerables batallas, de derrotas y victorias, gracias por ayudarme en el desarrollo de la tesis y en la escritura de la publicación.

Finalmente, agradezco a mi profesor Guía Werner Creixell, por su confianza, enseñanzas e incansable ayuda. Gracias maestro por creer en mí.

RESUMEN

Recientemente, los espacios hiperbólicos en el contexto de *Deep Learning* no euclidiano han ganado popularidad debido a su capacidad para representar datos jerárquicos. Este trabajo propone que es posible aprovechar la característica jerárquica presente en las imágenes mediante el uso de redes neuronales hiperbólicas en arquitecturas de redes generativas adversarias (GAN), lo cual mejorará el proceso de generación. En este estudio, se prueban diferentes configuraciones que utilizan capas hiperbólicas *feedforward* en las arquitecturas GAN, CGAN y WGAN, en lo que llamamos HGAN, HCGAN y HWGAN, respectivamente. Luego se dispone a utilizar capas neuronales hiperbólicas en la arquitectura StyleGAN2, que es una GAN donde la red generadora se fundamenta en conceptos de transferencia de estilos. El generador de la StyleGAN y StyleGAN2 a diferencia de las otras arquitecturas GAN estándar, consta de dos redes neuronales. Una de ellas es la red de mapeo de estilo f, cuya función es generar una representación vectorial de las características de alto nivel de la imagen. Por ende, se implementa una versión hiperbólica de la red de mapeo para explotar la jerarquía intrínseca de la información de alto nivel de la imagen a generar. En general, la GAN hiperbólica o HGAN depende de cómo estén distribuidas las capas neuronales hiperbólicas en la red y la curvatura del espacio. De esta forma, para cada arquitectura GAN propuesta se logran mejores resultados en su versión hiperbólica propuesta que su contraparte euclidiana. Todas las arquitecturas HGAN se miden utilizando las métricas de *Inception Score* (IS) y *Fréchet Inception Distance* (FID), y se prueban con el conjunto de datos MNIST para las arquitecturas HGAN, HWGAN y HCGAN, y CIFAR-10 para la arquitectura StyleGAN2 con mapeo hiperbólico.

Palabras Clave. GAN, Deep Learning, Hyperbolic Deep Learning, Poincaré Ball.

ABSTRACT

Recently, hyperbolic spaces in the non-Euclidean deep learning context have gained popularity due to their ability to represent hierarchical data. In this work, it is proposed that it is possible to take advantage of the hierarchical characteristic present in the images, using neural networks through hyperbolics in adversary generative network (GAN) architectures, which improves the generation process. In this study, different configurations using feedforward hyperbolic layers are tested in GAN, CGAN, and WGAN architectures, in what we call HGAN, HCGAN, and HWGAN, respectively. It is then arranged to use hyperbolic neural layers in the StyleGAN2 architecture, a GAN where the generating network is based on style transfer concepts. The StyleGAN and StyleGAN2 generator, unlike the other standard GAN architectures, consists of two neural networks. One of them is the f style mapping network, whose function is to generate a vector representation of the image's high-level features. Therefore, a hyperbolic version of the mapping network is implemented to exploit the intrinsic hierarchy of the image's high-level information to be generated. In general, the hyperbolic GAN or HGAN depends on how the hyperbolic neural layers are distributed in the network and space's curvature. In this way, for each proposed GAN architecture, better results are achieved in its proposed hyperbolic version than its Euclidean counterpart. All HGAN architectures are measured using the Inception Score (IS) and Fréchet Inception Distance (FID) metrics and tested against the MNIST dataset for the HGAN, HWGAN HCGAN architectures, and CIFAR-10 for the StyleGAN2 architecture with hyperbolic mapping.

Keywords. GAN, Deep Learning, Hyperbolic Deep Learning, Poincaré Ball.

Índice de Contenidos

Índice de Tablas

Índice de Figuras

Índice de Algoritmos

1 | Introducción

Las arquitecturas de *Deep Learning* o aprendizaje profundo producen representaciones jerárquicas o características abstractas, a partir de los datos de entrada (Bengio et al. (2013)). En ellas, los datos se representan en forma vectorial o tensorial, y éste es procesado a través de múltiples transformaciones no lineales llamadas capas neuronales (Zeiler y Fergus (2014)). Las capas, poseen diferentes funciones e interconexiones, que determinan cómo los datos son procesados y representados consecutivamente dentro de la red. Por cada muestra de entrada, la representación o característica depende de la estructura y los parámetros de la red. Estos últimos son aprendidos mediante un proceso de optimización de la red neuronal, y así aprender los patrones de los datos de entrada y manipular su distribución a través de la red hasta obtener una salida con las propiedades estadísticas deseadas. En tal sentido, la relación y patrones establecidos por la red, se basan fuertemente sobre la suposición, los datos de entrada pueden ser representados apropiadamente en un espacio euclídeo. De tal forma, esta suposición ha tenido éxito en entornos que trata con imágenes, sonido, texto y otras señales las cuales presentan claramente una estructura euclídea subyacente. Sin embargo, para datos que presentan una naturaleza no euclídea, el rendimiento de las capas tradicionales es bajo, no obstante, se pueden esperar una mejora usando capas neuronales que exploten estas propiedades geométricas (Bronstein et al. (2017)).

El campo del *Geometric Deep Learning* (Bronstein et al. (2017)), también conocido como aprendizaje profundo no euclídeo, es un nuevo acercamiento para producir representaciones en datos no euclídeos a diferencia de las redes convencionales (Wilson et al. (2014a)). En general, hay dos tipos de datos no euclídeos. Los primeros son aquellos que evidentemente no pueden ser representados adecuadamente por un espacio euclidiano, como grafos o formas 3D. El segundo tipo son los datos típicamente representados por la geometría euclidiana, aunque, presentan una estructura subyacente no euclídea, tales como, relaciones cíclicas o jerárquicas, donde, la jerarquía o la ciclicidad de los datos se puede explotar mejor a través de incrustaciones en un espacio curvado, también llamado variedad riemanniana. En tal sentido, el concepto curvatura puede entenderse intuitivamente analizando una superficie. De tal forma, la curvatura de la superficie corresponde a cuánto se desvía cualquier punto de esta, a un plano tangente. Siendo así, para espacios que presentan curvatura constante, hay tres familias (Weisstein (2002)), aquellas con curvatura positiva o espacios elípticos, curvatura negativa o espacios hiperbólicos, y espacios euclídeos con curvatura cero (Wolfe (2012)). En concreto, una variedad riemanniana no mantiene el quinto postulado de Euclides (postulado de las paralelas) (H. y I. (1996)), este postulado es equivalente a afirmar que, en un triángulo los ángulos internos suman 180 grados (Weisstein (2002)). En cambio, dependiendo de la curvatura del espacio, la suma de los ángulos en un triángulo puede ser mayor que 180 para espacios elípticos y menos de 180 para espacios hiperbólicos.

Los espacios no euclídeos con curvatura constante tienen propiedades útiles para proveer mejores representaciones para cierto tipo de estructura en los datos. Así por ejemplo, los espacios elípticos o esféricos son adecuados para representar datos con una estructura cíclica (Gu et al. (2019) y Wilson et al. (2014b)). Por otro lado, para los datos con estructura jerárquica se utilizan los espacios hiperbólicos, particularmente la bola de Poincaré, ya que, comparte la misma estructura métrica que los árboles (Krioukov et al. (2010)).En particular, las propiedades estructurales jerárquicas se encuentran en el lenguaje escrito y grafos (Samei y Jalili (2019) y Chami et al. (2019)), lo que ha motivado diferentes trabajos utilizando la bola de Poincaré, tal como, el trabajo realizado por Tifrea et al. (2018), en el que propone una adaptación del algoritmo GloVe (Pennington et al. (2014)). Este último corresponde a un algoritmo de incrustación de palabras en un espacio vectorial, cuyos vectores representan características semánticas de la palabra, el hecho de que

también sea hiperbólico entrega además una representación jerárquica. De igual forma, Dhingra et al. (2018) desarrollaron un algoritmo de incrustación de texto en espacio hiperbólico, el cual es aprendido mediante el uso de redes neuronales hiperbólicas. Análogamente, Nickel y Kiela (2017), proponen un método para realizar incrustaciones de grafos con estructura de árbol, en una bola de Poincaré de baja dimensión, para lograr mejores resultados en similitud de palabras y vinculación léxica (este último corresponde a identificar la relación semántica entre dos palabras) con el conjunto de datos WordNet (Miller (1998)). Sin embargo, además de texto y grafos, existen otros tipos de datos que tienen estructuras de árbol subyacentes o latentes, donde los espacios de Poincaré se han utilizado en menor grado. En tal sentido, las imágenes no presentan un comportamiento jerárquico aparente. No obstante, las imágenes se pueden agrupar en clases debido a sus similitudes. Estas clases también se pueden agrupar en otras clases, aumentando el nivel de abstracción o subiendo en una estructura de árbol. Khrulkov et al. (2020), afirman que la estructura semántica jerárquica de los conceptos del lenguaje también pueden estar presente en las imágenes de esos conceptos. Un claro ejemplo de este fenómeno se aprecia en el reino animal, donde la rama de los mamíferos presenta características visuales claras y se diferencia de la rama de los reptiles. Con este argumento, los autores proponen utilizar capas neuronales hiperbólicas, introducidas por Ganea et al. (2018), para mejorar las tareas de aprendizaje con pocas muestras, también conocido como *few-shot learning*. Además, las redes neuronales hiperbólicas (HNN) han sido útiles para los datos que exhiben anatomía latente jerárquica, ya que, aumentan la fidelidad de la representación de los datos, con menos distorsión y dimensionalidad.

Al igual que en las incrustaciones vectoriales, se han empleado espacios hiperbólicos para mejorar las tareas de generación mediante aprendizaje profundo, tanto en texto como en imágenes. Así, por ejemplo Dai et al. (2020) desarrollaron un modelo de generación de texto en el espacio hiperbólico, utilizando una versión del codificador automático variacional (VAE) en este espacio, el cual permite representar jerárquicamente imágenes en la bola de Poincaré como en los trabajos de Mathieu et al. (2019) y Nagano et al. (2019). En concreto, la arquitectura VAE se utiliza para aprender representaciones eficientes y también para la generación. Sin embargo, la arquitectura más popular para las tareas de generación de imágenes son las redes generativas adversarias (GAN).

Las GAN's constan de dos redes que participan en un juego adversario durante su entrenamiento. Esta arquitectura se compone de una red neuronal generador que elabora una salida similar al conjunto de datos de entrenamiento, y una red neuronal discriminador que diferencia entre los datos reales y generados. El generador se entrena de forma no supervisada, evaluando su salida con el discriminador, con el objetivo de confundir a este último. Por el contrario, el discriminador es entrenado de forma supervisada para diferenciar las imágenes reales de las falsas. Una vez que se hayan entrenado ambas redes, el generador puede producir datos similares al conjunto de datos, pero con variaciones originales, sólo a partir de una entrada de ruido. A causa del buen desempeño de las arquitecturas GAN, existen múltiples variaciones y modificaciones de la GAN original. No obstante, hasta donde se sabe, no existe una aplicación del espacio hiperbólico en estas arquitecturas que tenga como objetivo explotar las características jerárquicas de los datos, como en la arquitectura VAE.

La propuesta o hipótesis de la presente tesis consiste en aprovechar las características jerárquicas presentes en las imágenes mediante el uso de redes neuronales hiperbólicas, para mejorar el desempeño de la arquitectura GAN. De este modo, se puede lograr una mejor calidad y diversidad en las imágenes generadas. El objetivo de la tesis consiste en implementar las versiones hiperbólicas de diferentes arquitecturas GAN, como lo son: GAN de Goodfellow et al. (2014), WGAN-GP de Gulrajani et al. (2017), CGAN de Mirza y Osindero (2014) y StyleGAN2 de Karras et al. (2020). En cada caso, se realizan experimentos con diferentes arreglos y combinaciones de capas neuronales euclídeas e hiperbólicas y de valores de curvatura del espacio, donde, se espera mejorar el rendimiento de las arquitecturas originales. Estas se miden mediante las métricas para modelos generativos *Inception Score* y *Fréchet Inception Distance*.

La tesis se divide en 6 capítulos. En primer lugar, el capítulo 2 detalla algunos conceptos de *Deep Learning*, necesarios para el desarrollo de la tesis. El capítulo 3 introduce y explica brevemente las arquitecturas GAN's implementadas en los experimentos. El capítulo 4, explica los conceptos clave de espacios no euclídeos y específicamente espacios hiperbólicos, además de introducir las capas neuronales hiperbólicas utilizadas para el desarrollo de la GAN hiperbólica. El capítulo 5, se detalla el acercamiento original de la tesis, donde se especifican los experimentos y metodologías propuestas para probar la hipótesis. Finalmente,

el capítulo 6 contiene las conclusiones obtenidas a partir del desarrollo de la tesis.

2 | Deep Learning

El aprendizaje profundo o ***Deep Learning*** Goodfellow et al. (2015) corresponde a un conjunto de algoritmos inspirado en las redes neuronales del sistema nervioso, el cual en los últimos años ha revolucionado la industria, informática y electrónica, realizando tareas que se atribuyen a las personas y que hasta hace poco eran imposibles para una máquina. Por ejemplo, en la competencia de clasificación de imágenes ImageNet Deng et al. (2009), las técnicas basadas en redes neurales han liderado durante la última década, superando incluso el desempeño humano. En la actualidad, existe una gran variedad de problemas que se han automatizado gracias al aprendizaje profundo, tales como, el reconocimiento de facial, reconocimiento del habla, convertir texto a voz, transcripción de manuscritos, traducción de idiomas, asistentes digitales, diagnósticos médicos, manejo de automóviles, etc.

El aprendizaje profundo se enmarca dentro de lo que se conoce como ***Machine Learning*** o aprendizaje de máquinas, y su funcionamiento consiste en extraer y reconocer patrones en los datos de forma automática y secuencial, utilizando una gran cantidad y variedad de operaciones tensoriales llamadas capas. Estas operaciones o capas dependen de sus parámetros internos, y en general las capas poseen muchos de estos parámetros, pero existen capas con pocos parámetros o incluso ninguno, y que cumplen funciones de regularización o estabilidad. Los parámetros de la capa se modifican mediante un proceso de optimización basado en gradiente sobre una función de costo, donde esta depende ya sea del marco de trabajo, el problema a resolver, el modelo, los datos o el algoritmo. La salida de la capa es una representación tensorial de la información procesada, la cual se denomina mapa de características.

En este capítulo veremos los conceptos clave del aprendizaje profundo. Primero revisaremos lo que se denomina como **FeedFordward Networks** o **Multilayer perceptron** Nazzal et al. (2008), que corresponde a modelos sencillos de procesamiento neuronal basado en las redes neuronales biológicas. Luego se describirá el algoritmo **backpropagation**, necesario para entrenar los modelos de aprendizaje profundo. Se explicarán los algoritmos de optimización basados en gradiente que corresponden a **SGD** (Zinkevich et al. (2010)) y el algoritmo adaptativo **Adam** (Kingma y Ba (2017)). Luego métodos clásicos de regularización tales como **sanción por norma de parámetros** o también conocido como **decaimiento de pesos**, y capas de regularización como **Normalización de lote** (*Batch Normalization*) y **Dropout**. Finalmente se revisarán las **Redes Convolucionales**, que corresponden a otro tipo de capas neuronales especializadas para el procesamiento de imágenes.

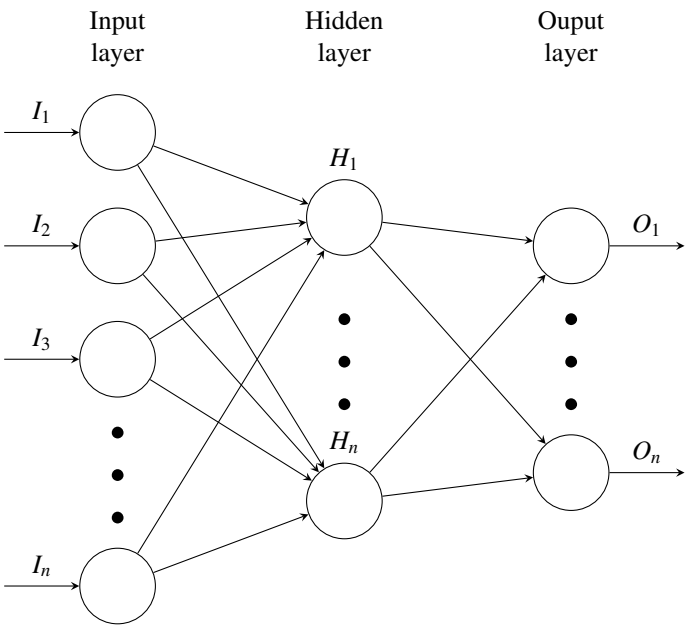


Figura 2.1: Representación red Feedforward, se aprecia que la red consiste en un grafo dirigido, donde los vectores resultantes corresponden a los nodos y las matrices de peso la matriz de afinidad.

2.1. FeedForward Networks

Las redes neuronales *FeedForward* o redes neuronales prealimentadas, son los primeros y más sencillos modelos de redes neuronales artificiales. En este tipo de red neuronal la información fluye hacia adelante y no existe ***feedback***. El objetivo de una red *feedforward* es aproximar alguna función $y = f^*(x)$, donde x es el vector de entrada e y el vector de salida, esto sirve para regresión o clasificación. La red *feedforward* define una función $\hat{y} = f(x, \theta)$, donde θ son los parámetros de la red, que se aprenden de forma tal para aproximar la función f^*.

El modelo de red neuronal *feedforward* se construye mediante la composición en cadena de capas *feedforward*. Cada capa toma la salida de la capa anterior para generar una salida y a su vez, esta pasa por otra capa hasta alcanzar la capa de salida, como se aprecia en la figura 2.1. Tomemos por ejemplo tres funciones o capas $f^{(1)}$, $f^{(2)}$ y $f^{(3)}$ conectadas en cadena, lo que constituye una composición de las funciones, de la forma $f(x) = f^{(3)}(f^{(2}(f^{(1)}(x)))$. Este tipo de estructuras es la más común al construir redes neuronales. En el caso de $f^{(1)}$, se denomina **primera capa**, $f^{(2)}$, **segunda capa**, y así sucesivamente. Al largo de la cadena completa se le denomina profundidad de la red y la última capa del modelo se conoce como capa de salida. Durante el entrenamiento, el algoritmo busca que el modelo $f(x)$ imite una función de blanco $f^*(x)$. A diferencia de algunos modelos de aprendizaje de máquinas o identificación de sistemas, la función objetivo no se tiene de forma explícita, más bien, se tiene un conjunto de ejemplos de x y el valor $y = f^*(x)$, que corresponden al conjunto de entrenamiento, donde x es la entrada e y la salida, para el problema de clasificación y es conocido como la etiqueta. El comportamiento de las capas intermedias, es decir, las capas que no son ni de salida ni de entrada, no está previamente definido por los datos del conjunto de entrenamiento, al contrario, el algoritmo de aprendizaje debe decidir cómo usar cada capa y encontrar la mejor implementación que modele $f^*(x)$. Debido a que de las capas internas, el usuario no tiene conocimiento de los valores de los parámetros, y que estas se modifican según el algoritmo de aprendizaje, son llamadas capas ocultas.

Las capas *feedfordward* consisten en una transformación afín y una posterior función de activación no lineal. Existe una gran variedad de funciones de activación, con diferentes propiedades y usos, la elección de que función de activación debe tener una capa depende de los resultados empíricos y el tipo de dato que se esté procesando. La ecuación 2.1 muestra la salida $\mathbf{z}$ de una capa en función de la entrada $\mathbf{x}$:

$$\mathbf{z} = F(\mathbf{W}\mathbf{x} + \mathbf{b}) \tag{2.1}$$
$$z_i = F\left(\sum_{j=1}^{N} w_{ij}x_j + b_i\right)$$

Donde w_{ij} corresponden a los valores i-ésimos y j-ésimos de la matriz de pesos $\mathbf{W}$, y b_i valor i-ésimos del vector de sesgo $\mathbf{b}$, x_j es el valor j-ésimo del vector de entrada, F la función de activación y z_i el valor i-ésimo del vector de salida de la capa o mapa de características. Cabe destacar que tanto $\mathbf{W}$ y $\mathbf{b}$ corresponden a los parámetros internos de la capa los cuales son aprendidos. Conectando varias capas, consecutivamente se obtiene un modelo profundo *feedforward*, y dependiendo de la profundidad del modelo es la capacidad para representar funciones no lineales que modelen la interacción de la entrada y la salida.

2.1.1. Funciones de Activación

Existe una gran variedad de funciones de activación empleadas en modelos de aprendizaje profundo, que debido a su naturaleza no-lineal permite a los modelos de *feedforward* no colapsar a una función lineal(Goodfellow et al. (2015), cap 6) . Inicialmente, las activaciones sigmoides $\sigma(\cdot)$ y tangente hiperbólica $tanh(\cdot)$ eran ampliamente utilizadas, y están relacionadas según $tanh(z) = 2\sigma(2z) - 1$. La función sigmoide se utiliza debido a que es la versión continua y diferenciable del escalón unitario y, de la misma forma, la función tangente hiperbólica con la función signo, que además son bioinspiradas. En la actualidad, estas funciones de activación han bajado su popularidad y uso en el campo de procesamiento de imágenes, debido a que, para valores muy positivos y negativos del argumento las funciones alcanzan un valor constante, provocando que el gradiente se desvanezca y en consecuencia genera un pobre comportamiento en la etapa de entrenamiento.

En los últimos años, la función de activación más utilizada en modelos de aprendizaje profundo corresponde a la ***Rectified Linear Unit*** o **ReLU**, debido a su simpleza y bajo costo computacional, que se traduce en modelos muy profundos y mayor velocidad de procesamiento tanto en inferencia como entrenamiento. Gracias a la simpleza de la ReLU, la derivada es fácil de calcular, además no existe el problema de que el gradiente se estanque en la parte positiva, favoreciendo su cálculo y la aplicación. Sin embargo, la ReLU entrega un valor cero para cualquier valor negativo, lo que provoca problemas con el gradiente. Debido a este problema, se propone la función de activación ***leaky ReLU***, el cual consiste en la ReLU pero con la parte negativa como una recta de menor pendiente. Finalmente, últimas propuestas ocupan la denominada ***Parametric ReLU***, el cual deja como parámetro la pendiente de la recta para valores negativos, por ende, corresponde a una versión generalizada que abarca tanto la ReLU como la leaky ReLU. Todas las funciones mencionadas se detallan en la tabla 2.1.

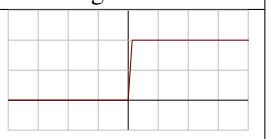
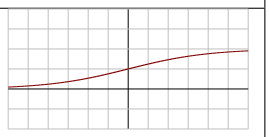
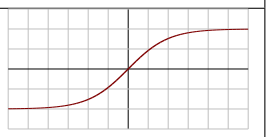
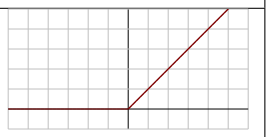
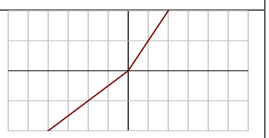
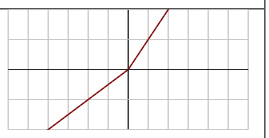
Nombre	Ecuación	gráfico
Escalón	$f(x) = \begin{cases} 0 & x \leq 0 \\ 1 & x \geq 0 \end{cases}$	
Sigmoide	$f(x) = \sigma(x) = \frac{1}{1+e^{-x}}$	
Tangente hiperbolica	$f(x) = tanh(x) = \frac{e^{x}-e^{-x}}{e^{x}+e^{-x}}$	
ReLU	$f(x) = \begin{cases} 0 & x \leq 0 \\ x & x \geq 0 \end{cases} = max\{0, x\}$	
Leaky ReLU	$f(x) = \begin{cases} 0{,}01x & x \leq 0 \\ x & x \geq 0 \end{cases}$	
Parametric ReLU	$f(\alpha, x) = \begin{cases} \alpha x & x \leq 0 \\ x & x \geq 0 \end{cases}$	

Tabla 2.1: Tabla resumen de las principales funciones de activación. Source: By Laughsinthestocks - Own work, CC BY-SA 4.0, https://commons.wikimedia.org/w/index.php?curid=44920533.

2.2. Backpropagation

Backpropagation o "propagación hacia atrás.[es] el algoritmo clave para entrenar redes neuronales basado en gradientes. Cuando se usa un modelo de redes neuronales *feedforward*, mediante una entrada x se produce una salida $\hat{y}$, la información fluye hacia adelante de la red. Este proceso se llama propagación hacia adelante. Durante el entrenamiento, se tiene el ejemplo (x, y), donde x es la entrada del modelo e y la etiqueta, se realiza la propagación hacia adelante con x para obtener la salida del modelo $\hat{y}$, con este último y la etiqueta y se calcula un costo escalar $J(\theta)$, donde θ son los parámetros internos de la red. Esto se aprecia en el algoritmo 1. El algoritmo de propagación hacia atrás, calcula los gradientes de los parámetros de cada capa, mediante un flujo de información inverso o hacia atrás, desde el valor de costo $J(\theta)$ hasta la entrada x. Esto se realiza con el cálculo analítico de la expresión del gradiente en cada capa, utilizando consecutivamente la regla de la cadena que se evalúa numéricamente de manera que el cálculo sea eficiente, como se ilustra en el algoritmo 2.

La función de costo $J(\theta)$ depende de una **función de pérdida**: $L(\hat{y}, y)$, además de los términos de regulación. La función de perdida es una métrica que indica cuanta diferencia existe entre los valores reales y la predicción $\hat{y}$. Esta depende de la aplicación, por ejemplo, en problemas de regresión es natural ocupar el error cuadrático medio como función de pérdida. Para entender como el algoritmo calcula los gradientes de los vectores internos del modelo, es necesario recordar algunos conceptos claves de cálculo como **regla de la cadena** y su respectiva generalización en $\mathbb{R}^n$.

Definición 2.2.1 (Regla de la Cadena)**.** Sea x un número real, y sea f y g ambas funciones son de $\mathbb{R} \to \mathbb{R}$. Supongamos $y = g(x)$ y $z = f(g(x)) = f(y)$, entonces la regla de la cadena establece que:

$$\frac{dz}{dx} = \frac{dz}{dy}\frac{dy}{dx} \tag{2.2}$$

Definición 2.2.2. (Generalización vectorial de regla de la cadena) Sea $x \in \mathbb{R}^m$, $y \in \mathbb{R}^n$, y las funciones $g : \mathbb{R}^m \to \mathbb{R}^n$ y $f : \mathbb{R}^n \to \mathbb{R}$. Si $y = g(x)$ y $z = f(x)$, entonces el gradiente de z respecto al vector x, $\nabla_x z$, Donde $\frac{\partial y}{\partial x}$ es la matriz Jacobiana $n \times m$ de g y $(\cdot)^T$ la operación transpuesta. es igual a:

$$\nabla_x z = \left(\frac{\partial y}{\partial x}\right)^T \nabla_y z \equiv \frac{\partial z}{\partial x_i} = \sum_j \frac{\partial z}{\partial y_j}\frac{\partial y_j}{\partial x_i} \tag{2.3}$$

Los algoritmos 1 y 2, son ejemplos para el caso de redes neuronales *feedforward*. Los modelos de aprendizaje profundo utilizan también capas otros tipos de capas como las convolucionales o recurrentes, capas de regularización ya sea normalización de lote y Dropout, o de concatenaciones u otro tipo de operaciones. Para la implementación real de estos algoritmos, se utilizan **grafos de procesamiento**, que es el concepto básico de todas las librerías de aprendizaje profundo, como lo son *Tensorflow*, *Torch*, *MxNet*, etc. Sin embargo, la idea es la misma, generar la salida del modelo a partir de los datos de entrada, a través de un paso *forward*, para posteriormente calcular los gradientes mediante la versión de *backpropagation* para grafos de computación, que también se basa en la regla de la cadena.

Algorithm 1 Propagación hacia adelante de un modelo feedforward y calculo de función de costo. La función de perdida $L(\hat{y}, y)$, el cual depende de la salida de la red $\hat{y}$ y la etiqueta y. El costo total J se considera la función de perdida más un termino de regularización $\Omega(\theta)$, donde θ consiste en los parámetros de la red (pesos y sesgos).

Require:
Red Neuronal de profundidad d
$W^{(i)}, i \in \{1, ..., d\}$, las matrices de pesos del modelo
$b^{(i)}, i \in \{1, ..., d\}$, los vectores de sesgo del modelo
$f^{(i)}, i \in \{1, ..., d\}$, las funciones de activación del modelo
x, la entrada del proceso
y, etiqueta u objetivo

$h^{(0)} = x$
for $k = 1, ..., d$ **do**
  $a^{(k)} = b^{(k)} + W^{(k)} h^{(k-1)}$
  $h^{(k)} = f^{(k)}\left(a^{(k)}\right)$
end for
$\hat{y} = h^{(d)}$
$J = L(\hat{y}, y) + \lambda\Omega(\theta)$

Algorithm 2 Propagación hacia atrás para el algoritmo 1. El algoritmo de *backpropagation* calcula los gradientes de las activaciones $a^{(k)}$ para cada capa k, desde la capa de salida hacia la primera capa oculta. Los gradientes indican como cambiar los parámetros de cada capa para reducir el error o el valor de costo J. Con los gradientes calculados es posible realizar métodos de optimización basado en gradiente

Después del cálculo hacia adelante de la red neuronal, se calcula el gradiente de la capa de salida:

$g \leftarrow \nabla_{\hat{y}} J = \nabla_{\hat{y}} L(\hat{y}, y)$

for $k = d, d-1, ..., 1$ **do**

Convierte el gradiente de la capa e salida al gradiente previo de la función no lineal(multiplicación elemento a elemento $\odot$, debido a que la función no lineal se aplica elemento a elemento):

$g \leftarrow \nabla_{a^{(k)}} J = g \odot f'(a^{(k)})$

Cálculo de gradiente de los pesos y los biases (incluido el termino de regularización):

$\nabla_{b^{(k)}} J = g + \lambda \nabla_{b^{(k)}} \Omega(\theta)$

$\nabla_{W^{(k)}} J = g + \lambda \nabla_{W^{(k)}} \Omega(\theta)$

Propagar los gradientes con respecto a las siguientes activaciones de las capas ocultas de niveles más bajos:

$g \leftarrow \nabla_{h^{(k-1)}} J = W^{(k)T} g$

end for

2.3. Optimización

El proceso de entrenamiento de un modelo de aprendizaje profundo consiste en resolver un problema de optimización. La optimización consiste en buscar los parámetros internos θ del modelo, que obtengan un bajo costo $J(\theta)$. Principalmente, los algoritmos de optimización utilizados se basan en descenso estocástico de gradiente (SGD: *Stochastic Gradient Descent*), estos mueven los parámetros en contra de la dirección del gradiente, es decir, sigue la dirección de la bajada más pronunciada del costo. Si bien, esto resulta obvio, existen muchas dificultades prácticas debido a que entrenar un modelo de aprendizaje profundo es un problema no convexo y de alta dimensión. Por ende, se encuentran fácilmente mínimos locales y puntos de silla. Como el entrenamiento es un proceso con bastante costo computacional y de tiempo de procesamiento, es necesario que los optimizadores sean capaces de adaptarse a regiones difíciles, con puntos de alto gradiente y otros de bajo, los cuales pueden provocar inestabilidad numérica o bien estancar la búsqueda en mínimos que no son globales. Una mala elección del optimizador o los parámetros de este, entregará un pobre desempeño en el entrenamiento. En consecuencia, el modelo no cumplirá con los estándares deseados.

2.3.1. Descenso de Gradiente Estocástico (SGD)

El Descenso de Gradiente Estocástico o *Stochastic Gradient Descent* (SGD) y sus variantes, son los optimizadores más utilizados en modelos de aprendizaje de máquinas y aprendizaje profundo. Este optimizador tiene un parámetro fundamental para su funcionamiento, que corresponde a la tasa de aprendizaje, la cual escala el gradiente para regular el tamaño de los pasos. En la práctica, la tasa de aprendizaje se programa para que vaya decreciendo mientras avanza las iteraciones, ya que cuando el entrenamiento se encuentra en un estado avanzado es necesario pasos más cortos. Se denota estocástico, debido a que resulta intratable estimar el gradiente de todo el conjunto de datos de entrenamiento, es por ello que se entrena mediante pequeños lotes de datos, el cual se denomina *minibatch training*. Estos lotes son muestras del conjunto de datos escogidos de forma aleatoria y de tamaño fijo, como se ilustra en el algoritmo 3.

Algorithm 3 Descenso de Gradiente Estocástico (SGD)

Require:
Tasa de aprendizaje ϵ.
Parámetros iniciales θ.
Modelo de aprendizaje profundo $D(\cdot, \theta)$.

1: **while** Criterio de detención no se cumpla **do**
2: Lote de datos de tamaño m, a partir del conjunto de entrenamiento $\left\{x^{(1)}, ..., x^{(m)}\right\}$ con las etiquetas correspondientes $y^{(i)}$.
3: Calculo del costo: $J(\theta)^{(i)} \leftarrow L(D(x^{(i)}, \theta), y^{(i)}) + \Omega(\theta)$
4: Estimación del gradiente (**Backpropagation**): $\hat{g} \leftarrow \frac{1}{m}\nabla_\theta \sum_i J(\theta)^{(i)}$
5: Calculo de delta: $\Delta\theta \leftarrow -\epsilon\hat{g}$ ▷ Parte Optimización
6: Actualizar parámetros:$\theta \leftarrow \theta + \Delta\theta$
7: **end while**

2.3.2. Descenso de Gradiente con Momento

Si bien el optimizador SGD presenta buenos resultados en el entrenamiento, el proceso de búsqueda es lento. Es por ello, que se incluye el concepto de **momento** con el objetivo de acelerar el aprendizaje, especialmente para regiones con alta curvatura y de gradientes pequeños o ruidosos. El momento es un concepto de la física clásica, el cual indica la cantidad de movimiento de un objeto, éste depende únicamente de su velocidad v y de su masa m. Las fuerzas externas pueden modificar el momento, lo que conlleva a un cambio en su velocidad y desplazamiento. La analogía de fuerzas externas de la física clásica a aprendizaje profundo corresponde al gradiente, el cual modifica la velocidad de caída (desplazamiento a regiones con bajo valor de costo), que posteriormente cambiará el desplazamiento que en este caso son los parámetros θ. Para integrar este concepto al algoritmo SGD, el cual se detalla en el algoritmo 4, se agrega una variable v de velocidad, esta depende de la velocidad anterior y del gradiente mediante un decaimiento exponencial. El parámetro $\alpha \in [0, 1)$ determina la contribución de la velocidad, y la tasa de aprendizaje la contribución de los gradientes, como se ve en el algoritmo 4.

Algorithm 4 Descenso de Gradiente con Momento: Parte Optimización

Require:
Tasa de aprendizaje ϵ.
Hiperparámetro $\alpha \in [0, 1)$.
Parámetros actuales θ.
Estimación del gradiente $\hat{g}$.
Velocidad actual v.

1: Actualizar velocidad: $v \leftarrow \alpha v - \epsilon\hat{g}$
2: Actualizar parámetros: $\theta \leftarrow \theta + v$

2.3.3. Algoritmo Momentos adaptivos (Adam)

La tasa de aprendizaje es el hiperparámetro que posee mayor influencia en como las redes neuronales aprenden. Por ende, escoger una tasa de aprendizaje resulta en la tarea más importante después de obtener un conjunto de datos de entrenamiento saneado y diseñar un modelo. Buenos modelos fallan con una mala elección de la tasa de aprendizaje. Es por esto, que es necesario diseñar optimizadores capaces de adaptar la taza de aprendizaje con respecto a la región de búsqueda mientras entrena. **AdaGrad** por sus siglas en inglés (Adaptative Gradients), posee propiedades teóricas deseables para optimización convexa, y es un gran acercamiento para resolver el problema. El optimizador controla la tasa de aprendizaje mediante la acumulación del histórico de los gradientes al cuadrado, r, que inicialmente es cero. Como se aprecia en

Algorithm 5 AdaGrad: Parte Optimización

Require:
Tasa de aprendizaje ϵ.
Constante pequeña δ, alrededor de 10^{-7}, para estabilidad numérica.
Parámetros actuales θ.
Estimación del gradiente $\hat{g}$.
Acumulación gradientes actual r, inicialmente 0.

1: Acumulación gradientes al cuadrado: $r \leftarrow r + \hat{g} \odot \hat{g}$
2: Cálculo actualización: $\Delta\theta \leftarrow -\frac{\epsilon}{\delta + \sqrt{r}} \odot \hat{g}$
3: Aplicar actualización: $\theta \leftarrow \theta + \Delta\theta$

Algorithm 6 RMSprop: Parte Optimización

Require:
Tasa de aprendizaje ϵ.
Constante pequeña δ, alrededor de 10^{-7}, para estabilidad numérica.
Coeficiente de decaimiento exponencial ρ
Parámetros actuales θ.
Estimación del gradiente $\hat{g}$.
Acumulación gradientes actual r, inicialmente 0.

1: Acumulación gradientes al cuadrado: $r \leftarrow \rho r + (1-\rho)\hat{g} \odot \hat{g}$
2: Cálculo actualización: $\Delta\theta \leftarrow -\frac{\epsilon}{\sqrt{\delta + r}} \odot \hat{g}$
3: Aplicar actualización: $\theta \leftarrow \theta + \Delta\theta$

algoritmo 5.

Sin embargo, AdaGrad presenta problemas en regiones no convexas, ya que, existe un prematuro decrecimiento de la tasa de aprendizaje antes de caer en una región convexa. Esto reduce la efectividad de la tasa de aprendizaje. Para resolver este problema, el optimizador **RMSprop** el cual es una modificación de AdaGrad, en lugar de usar todo el histórico de los gradientes usa un promedio con decaimiento exponencial para descartar el histórico que está muy en el pasado.

Finalmente, Kingma y Ba (2017), proponen el optimizador de momentos adaptivos o **adaptive moments** (**Adam**), que corresponde a la combinación de RMSprop con SGD momentum, pero con pequeñas distinciones, véase algoritmo 7. Primero, se estiman los momentos de primer y segundo orden del gradiente mediante un decaimiento exponencial controlado por los hiperparámetros ρ_1 y ρ_2, cuya entrada es el gradiente al cuadrado, donde ambos momentos inicialmente son 0. Segundo, Adam incluye una corrección de sesgo para estimar ambos momentos de primer orden y el momento de segundo orden, a diferencia de RMSprop donde este puede tener un alto sesgo en etapas tempranas del entrenamiento. Adam es mucho más robusto en la elección de los hiperparámetros, aunque igualmente es necesario una correcta elección de la taza de aprendizaje.

Algorithm 7 Adam: Parte Optimización

Require:
Tasa de aprendizaje ϵ.
Parámetros actuales θ.
primer y segundo momentos actuales s y r, ambos incialmente 0.
Tazas de decaimiento exponencial para estimación de momentos $\rho_1, \rho_2 \in [0, 1)$.
Constante pequeña δ, alrededor de 10^{-7}, para estabilidad numérica.

1: Actualización sesgada de estimación momento de primer orden: $s \leftarrow \rho_2 s + (1 - \rho_1)\hat{g} \odot \hat{g}$
2: Actualización sesgada de estimación momento de primer orden:$r \leftarrow \rho_2 r + (1 - \rho_2)\hat{g} \odot \hat{g}$
3: Corrección de sesgo estimación momento primero orden: $\hat{s} \leftarrow \frac{s}{1-\rho_1^t}$
4: Corrección de sesgo estimación momento segundo orden:$\hat{r} \leftarrow \frac{r}{1-\rho_2^t}$
5: Cálculo actualización: $\Delta\theta \leftarrow -\epsilon\frac{\hat{s}}{\sqrt{\hat{r}}+\delta}$
6: Aplicar actualización: $\theta \leftarrow \theta + \Delta\theta$

2.4. Regularización

Uno de los principales retos en el área de aprendizaje de máquinas, es encontrar algoritmos tales que tengan buen desempeño tanto en el conjunto de datos de entrenamiento como con datos nuevos nunca antes vistos. Es por ello que en muchos algoritmos de aprendizaje de máquinas se divide el conjunto de datos total en dos conjuntos, el conjunto de entrenamiento y el conjunto de prueba cuya proporción generalmente es de 80-20. Conceptos como ***overfitting*** (sobre ajuste) o ***underfitting*** (bajo ajuste) entran en juego. El *overfitting* ocurre cuando el modelo presenta un excelente desempeño en el conjunto de entrenamiento, pero un pobre desempeño en el conjunto de prueba. Cuando este fenómeno ocurre, muchas veces se dice que el modelo está modelando el ruido del conjunto de entrenamiento. Por otro lado, *underfitting* es el concepto contrario, es decir, pobre desempeño en el conjunto de entrenamiento, pero aceptable en el conjunto de prueba. Si se juntan ambos conceptos en una sola métrica, se identifica una curva en forma de U, alto *overfitting* implica bajo *underfitting* y viceversa. Por ende, es necesario encontrar el punto en el cual el modelo se comporte bien tanto el conjunto de entrenamiento como el conjunto de prueba.

Los modelos de aprendizaje profundo o de redes neuronales artificiales no quedan fuera de esto, donde, el poder de predicción de un modelo está relacionado directamente a la cantidad de parámetros y a su profundidad. Por ende, modelos con muchos parámetros tienden a caer en el *overfitting*, y modelos con pocos en *underfitting*. Existen diversos métodos para tener un modelo de gran cantidad de parámetros sin caer en el *overfitting*, manteniendo así, el poder de predicción del modelo hasta cierto grado.

2.4.1. Sanciones por Norma de los parámetros

Muchas de las técnicas de regularización se basan en reducir la capacidad del modelo. La técnica consiste en agregar una sanción por la norma de los parámetros $\Omega(\theta)$ a la función de costo u objetivo J, obteniendo una nueva función de perdida $\tilde{J}$:

$$\tilde{J}(\theta; X, y) = J(\theta; X, y) + \alpha\Omega(\theta) \tag{2.4}$$

Donde $\alpha \in [0, \infty)$ es el hiperparámetro que escala la contribución de la sanción de norma Ω. Cuando α es cero, se tiene la función de pérdida original, y para valores grandes de α, indica que existe mucha penalización. Cabe destacar que sólo se penaliza los pesos del modelo, mientras que los términos de sesgo se mantienen sin regularización.

Las normas más comunes para la penalización son la norma L^1 y L^2, donde la norma L^2 es conocida como **decaimiento de pesos**. El termino de regularización que se adhiere es de forma cuadrática, cabe

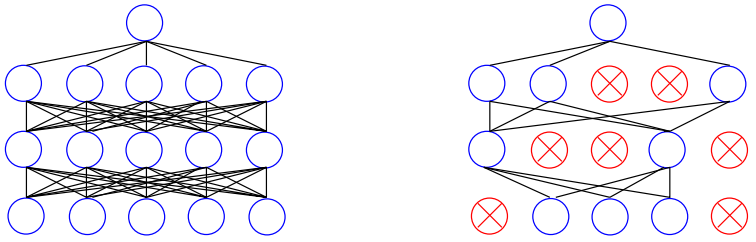
destacar que en otras áreas de investigación se relaciona con la energía. El decaimiento de pesos provoca que los pesos se encuentren cerca de origen, independiente del signo. Por otro lado, la norma L^1 corresponde a la suma de los pesos en valor absoluto, esto provoca que las matrices de pesos sean **ralas**. Ralo significa que muchos valores de la matriz sean igual a cero, lo que reduce el costo computacional. Las estrategias de regularización L^2 y L^1, se ilustran en las ecuaciones (2.6) y (2.5) respectivamente.

$$\tilde{J}(W; X, y) = J(\theta; X, y) + \frac{\alpha}{2} W^T W \tag{2.5}$$

$$\tilde{J}(W; X, y) = J(\theta; X, y) + \alpha \|W\|_1 \tag{2.6}$$

2.4.2. Dropout

El *Dropout* es un método de bajo costo computacional, que permite regularizar una gran cantidad de familias de modelos. Esta técnica se basa en el método de ***bagging***, el cual consiste en definir k modelos diferentes, construir k conjuntos de datos mediante el muestreo del conjunto de entrenamiento con reemplazo, para finalmente entrenar cada modelo i con cada conjunto de datos i. Un ejemplo de implementación del *bagging* es el algoritmo *Random Forest*. Sin embargo, para los modelos de aprendizaje profundo el método de *bagging* resulta poco práctico y excesivamente caro. La técnica *Dropout* entrega una aproximación barata para entrenar y evaluar un conjunto de *bagging* de una cantidad exponencial de modelos de redes neuronales. El *Dropout* se implementa mediante una capa ubicada después de una capa de activaciones, cuya función es ignorar ciertas unidades que son elegidas aleatoriamente. Ignorar se refiere a que la unidad no es considerada tanto en la etapa de *forward*, como la de *backward*, como se aprecia en la figura 2.2 .



(a) Red feedforward sin Dropout. **(b)** Red feedforward con Dropout.

Figura 2.2: Ejemplo del funcionamiento de las capas *Dropout*.

Al momento de entrenar, la capa de *Dropout* multiplica elemento a elemento la capa de activación previa con una máscara que implementa un proceso de Bernoulli, que toma valores 0 o 1 con probabilidad p y $1 - p$ respectivamente. El *Dropout* obliga a todas las neuronas de la capa a aprender, debido a que las neuronas que están apagadas no aportarán en la activación resultante. Por ende, el resultado final de la red depende de las neuronas activadas, lo cual es aleatorio. Una vez entrenado, se ocupan todas las activaciones, pero estas se reducen por un factor igual a la probabilidad del proceso de Bernoulli $1 - p$, que corresponde al promedio de las activaciones prendidas durante el entrenamiento.

2.4.3. Normalización de lote (Batch Normalization)

El algoritmo de **normalización de lote**, al igual que el *Dropout*, es uno de los más utilizados para la regularización de modelos de aprendizaje profundo. Esto se debe a que además de regularizar, estabiliza la entrada a las capas neuronales, permitiendo ocupar optimizadores con tasas de aprendizaje más altas. El algoritmo de normalización de lote se implementa mediante una capa que se conecta antes de la capa neuronal, el cual emplea una normalización estadística provocando que la salida se asemeje a una distribución gaussiana de media 0 y varianza 1. Sea H un lote de datos de mapa de características de la capa a normalizar,

que posee estructura de tensor, donde cada elemento del tensor corresponde a un mapa de características. El procedimiento es el siguiente:

$$H'_{ij} = \frac{H_{ij} - \mu_j}{\sigma_j}, \tag{2.7}$$

donde μ es el tensor que contiene el promedio de cada unidad y σ es el vector que con la desviación estándar de cada unidad. Los vectores μ y σ se aplican por cada fila del tensor H. Dentro de la fila, la aritmética es elemento a elemento, es decir, H_{ij} se normaliza substrayendo μ_j y dividiendo por σ_j. Los vectores μ y σ se calculan en cada paso en el entrenamiento, también estos pueden ser aprendidos durante el entrenamiento.

$$\mu_j = \frac{1}{m}\sum_i H_{ij} \tag{2.8}$$

$$\sigma_j = \sqrt{\delta + \frac{1}{m}\sum_i (H_{ij} - \mu_i)^2} \tag{2.9}$$

Donde δ es una constante positiva igual a 10^{-8} para estabilidad numérica, debido a que el gradiente de $\sqrt{z}$ no está definido cuando $z = 0$.

2.5. Redes Convolucionales

A partir de las redes *feedforward*, existen diversas variaciones dependiendo de la topología de los datos. En principio la redes *feedforward* procesan la información representada de forma vectorial, existen versiones tensoriales para procesar más dimensiones, pero el costo computacional es elevado debido a la cantidad de conexiones y operaciones. Es por ello que dependiendo del problema se han diseñado nuevas capas bio-inspiradas en las células neuronales especializadas en visión. En esta sección se introducen las **redes neuronales convolucionales** ideales para procesar imágenes.

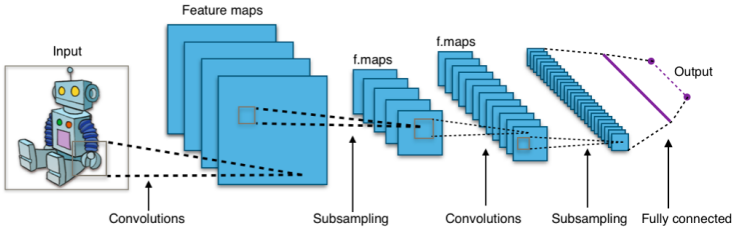


Figura 2.3: Implementación típica de una red neuronal convolucional. Source: By Aphex34 - Own work, CC BY-SA 4.0, https://commons.wikimedia.org/w/index.php?curid=45679374

Las redes convolucionales son especializadas en procesar datos en forma de grilla. Por ejemplo, grillas 1-D como series de tiempo, grillas 2-D imágenes y grillas 3-D vídeos. Como su nombre lo indica, las redes convolucionales se basan en la operación matemática de la convolución, dependiendo de la dimensión de la grilla es la convolución que se emplea. La convolución es una operación sobre dos funciones de argumentos reales, supóngase que se posee una señal $x(t)$ contaminada con ruido, una solución común es filtrar esta señal mediante un filtro que elimine las bandas de frecuencia donde se encuentra el ruido. El proceso de filtrar

corresponde a realizar la convolución entre la señal $x(t)$ y la respuesta impulso del filtro $w(t)$ (la salida del filtro a una entrada impulso). La salida filtrada resultante es igual a:

$$s(t) = (x * w)(t) = \int x(a)w(t-a)da \tag{2.10}$$

Este ejemplo corresponde a dos señales continuas. En la práctica las señales que se tienen son representaciones discretas del mundo continuo, ya que se obtienen mediante el muestreo y conversión digital, para luego ser procesadas mediante un computador. Es por ello, que se debe utilizar la versión discreta de la convolución la cual se define de la siguiente forma:

$$s(t) = (x * w) = \sum_{a=-\infty}^{\infty} x(a)w(t-a) \tag{2.11}$$

En términos de redes convolucionales, $x(t)$ corresponde a la entrada, $s(t)$ se denomina mapa de características y $w(t)$ **kernel** de la convolución. La entrada generalmente es un tensor de datos y el kernel es un tensor de parámetros de la capa neuronal, los cuales son modificados durante el entrenamiento. Esto implica que muchos términos de la suma infinita son cero, debido a que los tensores tienen tamaño fijo, en consecuencia se acota la suma para los términos de los tensores. Finalmente, la definición de convolución se puede extender a más dimensiones. Por ejemplo, si la entrada es una imagen I, entonces el kernel K será de dos dimensiones al igual que la salida.

$$S(i,j) = (I * K)(i,j) = \sum_m \sum_n I(m,n)K(i-m, j-n) \tag{2.12}$$

Por razones prácticas, muchas librerías de aprendizaje de máquina implementan la convolución como la correlación cruzada. Además, la salida se restringe sólo a las posiciones donde el kernel cabe completamente en la imagen, como se ve en la figura 2.4. La correlación cruzada se define como:

$$S(i,j) = (I * K)(i,j) = \sum_m \sum_n I(i+m, j+n)K(m,n) \tag{2.13}$$

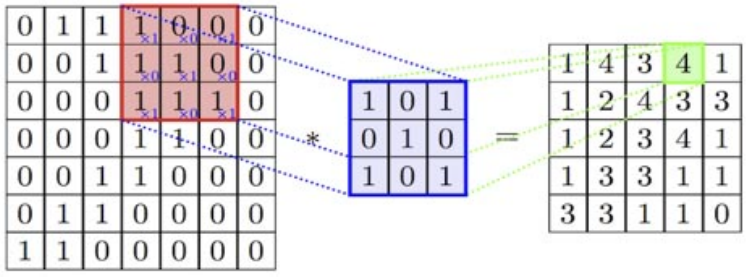


Figura 2.4: Ejemplo de implementación de la operación convolución dentro de una red neuronal convolucional. Source: https://rohanverma.net/blog/2018/10/14/convolutional-neural-network-basics/

La convolución aprovecha tres ideas importantes que ayudan a mejorar el rendimiento de los sistemas de aprendizaje de máquinas: **interacciones escasas**, **compartir parámetros** y **representación equivariante**, además, permite trabajar con entradas de tamaño variable. La interacción escasa se refiere a que los kernels son más pequeños que las imágenes de entrada, es por ello que requieren menos memoria para almacenarlos y además consumen menos potencia de cómputo. Estas mejoras en la eficiencia del algoritmo son muy importantes. En una red *feedforward* donde el tamaño de la entrada es m y la salida n, la multiplicación matricial tiene una complejidad computacional de $O(m \times n)$. Como los kernels son más pequeños que las matrices de peso, de tamaño k, se limita la complejidad operacional a $O(k \times n)$, lo que mejora la velocidad

y el consumo de memoria. Por ende, es posible crear redes neuronales más profundas, más rápidas y con menos parámetros. Por otro lado, compartir parámetros se refiere a usar los mismos parámetros para más de una función. En modelos *feedforward*, cada elemento de la matriz de pesos se usa una vez cuando se calcula la salida de la capa. En cambio, los kernels de una capa convolucional se usan por cada posición de la imagen (excepto las posiciones en que se sale del límite de la entrada). Finalmente, la representación equivariante se refiere a que, si la entrada cambia, la salida cambia de la misma forma. En este sentido, las capas convolucionales tienen una propiedad llamada **equivariancia a la traslación**, útiles para procesamiento de imagen.

Finalmente, las implementaciones modernas de capas convolucionales tienen los siguientes los hiperparámetros: cantidad de kernels, dimensión de los kernels que generalmente son cuadrados, método de ***padding*** que se refiere a agregar ceros ya que el tamaño de la salida disminuye debido al proceso de convolución (el método de *padding* permite mantener un tamaño constante), ***stride*** que es la cantidad de pasos en que se aplica la correlación cruzada, **dilatación** que indica la cantidad de espacios entre los distintos valores del kernel, sesgo y función de activación.

2.5.1. Pooling

La función o capa de ***pooling*** realiza un muestreo del mapa de características resumiendo las características cercanas mediante una estadística simple. Por ejemplo, la capa de ***max pooling***, devuelven el máximo elemento de un vecindario rectangular. Esto permite crear representaciones aproximadamente invariantes a pequeñas traslaciones de la entrada, por ende, las capas de *pooling* son extremadamente útiles después de aplicar capas de convolución y por lo general siempre se usan en modelos de neuronas convolucionales. Existen diversos tipos de capas *pooling*, todas operan dentro de pequeñas vecindades rectangulares. Las más utilizadas son ***average pooling***, que promedia las características cercanas, norma L^2 y *average pooling* con peso basado en la distancia al pixel central. Por último, también existen capas *pooling* globales tales como, ***Global Max Pooling*** y ***Global Average Pooling***, las cuales no operan sobre pequeñas vecindades, sino más bien, a lo largo de toda la entrada. Estas capas son utilizadas como capa final de los modelos, reemplazando el uso de capas *feedforward* con funciones *softmax*.

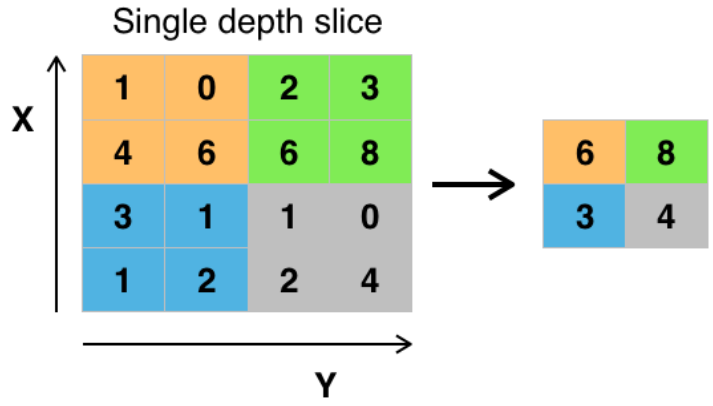


Figura 2.5: Ejemplo de capa de Max Pooling. Source: By Aphex34 - Own work, CC BY-SA 4.0, https://commons.wikimedia.org/w/index.php?curid=45673581

3 | Redes Adversarias Generativas (GAN)

En el área del aprendizaje profundo, los modelos que más han llamado la atención de los investigadores en los últimos años corresponden a los modelos generativos. Estos modelos como su nombre lo indica, permiten generar nuevos datos que no se encuentran en el conjunto de datos de entrenamiento, aprendiendo la distribución de los datos de manera implícita. Esto resulta bastante útil debido a que es posible crear contenido nuevo y original de manera autónoma sin la intervención humana, además de refinar otros procesos como asistencia en la edición o creación. Dentro de los modelos generativos, las redes adversarias generativas o GAN (*Generative Adversarial Netowrk*) de Goodfellow et al. (2014), han ganado gran popularidad y uso debido a los buenos resultados que han obtenido en diferentes dominios, como generación de texto, voz, imágenes, música, vídeos, etc. Las GAN's se han ubicado como el estado del arte en muchas tareas de generación en diversos marcos de trabajo y conjunto de datos tales como MNIST, CIFAR-10, LSUN, ImageNet, etc. Todo gracias a su funcionamiento que es sencillo y al mismo tiempo elegante. Este se basa en dos modelos de redes neuronales profundas que participan en un juego adversario, donde lo aprendido por un modelo influye en el otro. El entrenamiento busca el **equilibrio de Nash**, que consiste en un estado de equilibrio en un juego con dos o más jugadores, donde cada jugador a adoptado la estrategia que le entregue más beneficios considerando todas las estrategias de los rivales.

En este capítulo revisaremos brevemente la arquitectura GAN. Luego los diferentes modelos con los que se trabajó durante el desarrollo de la tesis. Finalmente se mostrará las diferentes métricas que existe en la actualidad para medir el desempeño de una arquitectura GAN.

3.1. Arquitectura GAN

La arquitectura GAN consiste en dos redes neuronales que compiten entre sí, la red **generador** $\mathcal{G}$ cuya función es generar datos que se asimilen al conjunto de entrenamiento a partir de ruido, y la red **discriminador** $\mathcal{D}$ el cual indica la probabilidad de que el dato de entrada sea real, como se muestra en la figura 3.1. Para ejemplificar el entrenamiento considere la generación de imágenes, supóngase que la red $\mathcal{G}$ corresponde a un falsificador de pinturas, en cambio, la red $\mathcal{D}$ es un experto en detectar falsificaciones. El falsificador $\mathcal{G}$ intentara crear pinturas que se asemejen mucho a las reales, para lograr engañar al experto $\mathcal{D}$, pero, $\mathcal{G}$ no tiene acceso a las pinturas reales. Por otro lado, el experto $\mathcal{D}$ aprenderá a identificar las falsificaciones de las reales, teniendo acceso tanto a las pinturas reales como las falsificadas. Entonces, mientras el experto $\mathcal{D}$ sea más capaz de diferenciar las falsificaciones de las reales, el falsificador $\mathcal{G}$ tendrá que aprender a generar falsificaciones que se asemejen más a las reales, y así, engañar al experto. En Teoría de juegos, esta competencia entre dos agentes se denomina juego adversario o **Min-Max**, debido a que el Discriminador busca maximizar los aciertos, en cambio el Generador busca minimizarlo. El entrenamiento consiste en resolver:

$$\min_{\mathcal{G}} \max_{\mathcal{D}} \; V(\mathcal{D}, \mathcal{G}), \tag{3.1}$$

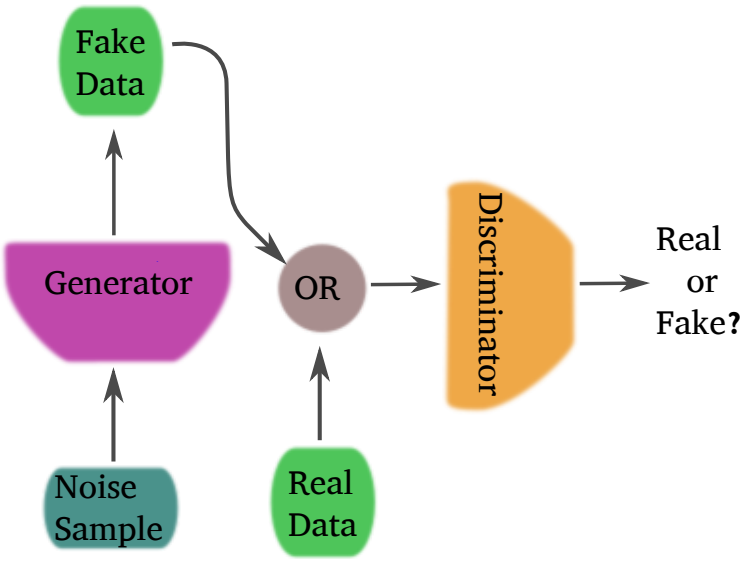


Figura 3.1: Diagrama general de arquitectura GAN, donde se aprecia los modelos de redes neuronales generador y discriminador, con sus respectivas entradas y salidas.

tal que

$$V(\mathcal{D},\mathcal{G}) = \mathbb{E}_{x\sim p_{\text{data}}(x)}[\log(\mathcal{D}(x))] + \mathbb{E}_{z\sim p_z(z)}[\log(1-\mathcal{D}(\mathcal{G}(z)))]. \tag{3.2}$$

Donde, $p_{data}(x)$ y $p_z(z)$ corresponden a la función de densidad probabilística de los datos y del ruido, este último se denomina **espacio latente** y $\mathbb{E}[\cdot]$ corresponde al operador esperanza. Durante el entrenamiento, los parámetros de una red se actualizan mientras la otra permanece fija. En las arquitecturas GAN para un Generador fijo, el discriminador óptimo cumple la relación:

$$\mathcal{D}^*(x) = \frac{p_{data}(x)}{p_{data}(x) + \mathcal{G}(p_z(z))}. \tag{3.3}$$

Por otro lado, la ecuación (3.2) muestra que el generador es óptimo cuando $\mathcal{G}(p_z(z)) = p_{data}(x)$, que es equivalente que el discriminador óptimo sea $\mathcal{D}^*(x) = 0{,}5$. Es decir, el discriminador no puede distinguir entre los datos reales y los falsos, por ende, retorna una probabilidad de 0,5 a cualquier dato de entrada. Idealmente, el discriminador es entrenando hasta alcanzar el óptimo para un respectivo generador, para luego actualizar este último. Sin embargo, en la práctica este procedimiento tardaría mucho tiempo en entrenar el discriminador hasta alcanzar el óptimo por cada iteración del generador. Por ello se entrena el generador por una cantidad de iteraciones del discriminador, generalmente esto se realiza de forma simultánea. Debido a que los modelos se entrenan mediante optimizadores basados en gradientes, existen muchos desafíos o problemas para encontrar la solución de un juego min-max, en consecuencia, resulta difícil entrenar una arquitectura GAN (Arjovsky y Bottou (2017)). Posteriores propuestas buscan solucionar este problema mediante modificaciones en la función de perdida, que se verán con más detalle en las próximas secciones.

3.2. GAN Condicional

La arquitectura GAN presenta un problema, resulta imposible controlar la imagen a generar. Por otro lado, muchos conjuntos de datos vienen con etiquetas para entrenamiento supervisado, el cual no es aprovechado en el entrenamiento de la GAN. Es por ello, la GAN condicional (Mirza y Osindero (2014)) permite explotar esta información adicional para mejorar el desempeño de la arquitectura GAN, y además proveer un método para controlar el tipo de imagen a generar. Una arquitectura GAN puede ser extendida a

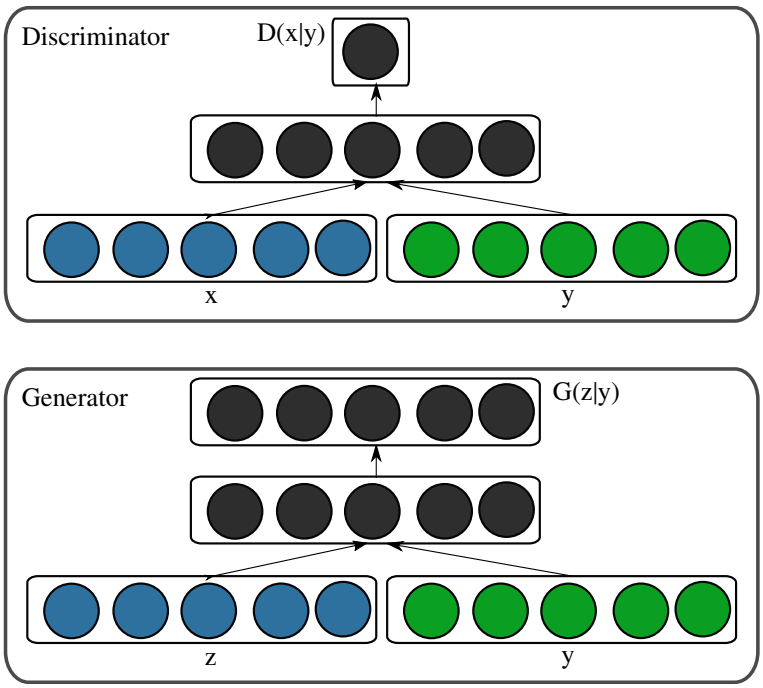


Figura 3.2: Diagrama simplificado de la arquitectura GAN Condicional (CGAN).

un modelo condicional, si ambas redes, generador y discriminador son condicionados con alguna información extra y. La información adicional puede ser cualquier tipo de información auxiliar, tal como etiquetas o datos de otras modalidades, como por ejemplo marcas de rostros para condicionar la generación de rostros. Es posible realizar el condicionamiento, alimentando la información extra y al generador y al discriminador como una entrada adicional.

En el generador la entrada de ruido z, que es una muestra de la distribución $p_z(z)$, se combina con y mediante una representación oculta. Dependiendo del marco de trabajo y el tipo de información auxiliar, es posible diseñar un mecanismo de interacción entre y y el ruido z. Sin embargo, lo más sencillo sería concatenarlos en la capa de entrada. Por otro lado, el dato x con la información auxiliar y entran como entradas en el discriminador, permitiendo condicionar la función de discriminación del dato x. Ambos mecanismos de condicionamiento para el generador y el discriminador se muestran en la figura 3.2. La ecuación de la función objetivo del juego min-max de la GAN condicional es:

$$V(\mathcal{D},\mathcal{G}) = \mathbb{E}_{x\sim p_{\text{data}}(x)}[\log(\mathcal{D}(x|y))] + \mathbb{E}_{z\sim p_z(z)}[\log(1-\mathcal{D}(\mathcal{G}(z|y)|y))]. \tag{3.4}$$

3.3. GAN Wasserstein

Entrenar una arquitectura GAN es una tarea difícil, debido a que se busca el equilibrio de Nash de ambos agentes en un juego adversario mediante optimización basada en gradientes. Esto provoca varias dificultades, entre los mayores problemas consisten: no convergencia de los modelos, colapso de modo, es decir, el generador colapsa y genera una limitada variedad de ejemplos, y finalmente, desvanecimiento y explosión de gradientes. La **GAN Wassestein** o **WGAN** propuesta por Arjovsky et al. (2017), permite resolver estos problemas mediante una modificación en su función de costo utilizando la **distancia Wassertein**, la cual estaba basada en conceptos de teoría de la información, y permite que los gradientes de la GAN sean más suaves

3.3.1. Distancia Wasserstein

La distancia Wasserstein es una función de distancia definida entre distribuciones probabilísticas, que surge de la idea del **transporte óptimo**. Supóngase que se tiene dos densidades de probabilidad $p(x)$ y $q(y)$, y se desea encontrar el plan de transporte que traslade la masa de $p(x)$ a $q(y)$, esto es posible debido a que ambas distribuciones tienen la misma masa total (igual a 1). Sin embargo, transportar una unidad de masa desde el punto x al punto y, tiene un costo $c(x, y) \mapsto [0, \infty)$, el cual puede ser simplemente la distancia entre los puntos $||x - y||$. Un plan de transporte que mueva la masa de $p(x)$ hacia $q(y)$ se describe como una función $\gamma(x, y)$, la cual mueve una cantidad de masa desde x hacia y. Para visualizar esta tarea, imagine que una pila de tierra de forma $p(x)$, se traslada hasta obtener otra pila de tierra de forma $q(y)$. El plan de transporte cumple las siguiente propiedades:

$$\int \gamma(x, y)dy = p(x) \tag{3.5}$$

$$\int \gamma(x, y)dx = q(y) \tag{3.6}$$

Esto quiere decir, la masa total movida desde una región infinitesimal alrededor de x, debe ser igual a $p(x)dx$, por otro lado, la masa movida hacia una región infinitesimal alrededor de y, debe ser igual a $q(y)dy$. Como se trabaja con distribuciones de probabilidad, el plan de trasporte $\gamma(x, y)$ es una distribución de probabilidad conjunta, cuyas distribuciones marginales son $p(x)$ y $q(y)$. Por lo tanto, el costo total de trasportar la masa de $p(x)$ a $q(y)$ es equivalente a:

$$\int c(x, y)d\gamma(x, y) = \int ||x - y||d\gamma(x, y) = \int\int ||x - y||\gamma(x, y)dxdy = \mathbb{E}_{(x,y)\sim\gamma}\left[||x - y||\right] \tag{3.7}$$

La distancia Wasserstein se define como el costo más bajo de todos los planes de transporte, es decir el costo del plan de transporte óptimo.

$$W(p, q) = \inf_{\gamma\in\Pi(p,q)} \mathbb{E}_{(x,y)\sim\gamma}\left[||x - y||\right] \tag{3.8}$$

Donde, $\Pi(p, q)$ denota el conjunto de todas las distribuciones conjuntas $\gamma(x, y)$, cuyas marginales son $p(x)$ y $q(y)$ respectivamente.

3.3.2. WGAN

La arquitectura GAN se basa en optimizar la función de costo mostrada en la ecuación 3.2, la que está estrechamente relacionada la divergencia de Jensen-Shanon, que a su vez, se basa en la divergencia de Kullback-Leibler, esto se aprecia en las siguientes ecuaciones:

$$KL(p||q) = \int p(x)\log\left(\frac{p(x)}{q(x)}\right)dx \tag{3.9}$$

$$JS(p||q) = \frac{1}{2}KL\left(p\left\|\frac{p+q}{2}\right.\right) + \frac{1}{2}KL\left(q\left\|\frac{p+q}{2}\right.\right) \tag{3.10}$$

Es posible demostrar que cuando se obtiene el discriminador óptimo $\mathcal{D}^*$, la función de perdida resulta:

$$V(\mathcal{G}, \mathcal{D}^*) = 2JS(p_{data}(x)||\mathcal{G}(p_z(z))) - 2\log 2 \tag{3.11}$$

Debido a la naturaleza de la divergencia de Jensen-Shanon, las cuales se saturan, introduce graves problemas en la optimización, tales como: baja convergencia, desvanecimiento de gradiente, o colapso de modo (Arjovsky y Bottou (2017)). En cambio, la distancia Wassestein debido a su naturaleza, que no se satura, no presenta desvanecimiento o explosión de gradiente, por ello, resulta como la mejor opción para definir el costo de la arquitectura GAN. Sin embargo, la ecuación de la distancia Wasserstein resulta intratable (3.8), pero, es posible simplificar su cálculo usando la **dualidad de Kantorovich-Rubinstein**, que se expresa en la ecuación (3.12).

$$W(p,q) = \sup_{\|f\|_L \leq 1} \mathbb{E}_{x\sim p}[f(x)] - \mathbb{E}_{x\sim q}[f(x)] \tag{3.12}$$

Donde sup indica el supremo, el cual es el mínimo elemento que, al aplicar la función el resultado es mayor o igual a todos los elementos del recorrido de la función. f corresponde a una función **1-Lipschitz**, aquellas que cumplen la siguiente restricción:

$$|f(x_1) - f(x_2)| \leq |x_1 - x_2| \tag{3.13}$$

Entonces, para calcular la distancia de Wasserstein solo se requiere encontrar una función 1-Lipschitz. Sin embargo, no es necesario encontrarla analíticamente, más bien, es posible aprender esta función usando el paradigma del aprendizaje profundo. En este caso, el discriminador es el candidato ideal para hacer esta tarea, pero con algunas modificaciones. La modificación consiste en sacar la función de activación sigmoidea en la capa final del discriminador, para que el recorrido del discriminador abarque todos los números reales. El valor nuevo que retorna el discriminador no corresponde a una probabilidad de que el dato sea real, en cambio, se interpreta como el puntaje que indica que tan real es el dato. Por ejemplo, para una entrada x, si el discriminador entrega un valor grande, x tiene mayor grado de realidad, en cambio, si entrega un valor bajo (negativo), x tiene menor grado de realidad, es decir, x es falso. En algunos estudios, el discriminador pasa a ser llamado crítico, ya que realiza la función de criticar los datos de entrada. Finalmente, para forzar la restricción 1-Lipschitz en el discriminador, se aplica una función ***clipping*** o saturación a los pesos del discriminador, donde el umbral de saturación o corte pasa a ser un hiperparámetro del modelo. El nuevo juego min-max se modela como:

$$\max_{\mathcal{G}} \min_{\mathcal{D}} \mathbb{E}_{z\sim p_z(z)}[\mathcal{D}(\mathcal{G}(z))] - \mathbb{E}_{x\sim p_{data}(x)}[\mathcal{D}(x)] \tag{3.14}$$

3.3.3. WGAN con Penalización de Gradiente (WGAN-GP)

Si bien, la WGAN funciona, y aplicar la función clipping a los pesos del discriminador permite forzar la restricción de 1-Lipschitz, en la práctica esto presenta muchas dificultades para el entrenamiento. La principal causa es la sensibilidad al hiperparámetro de corte de la función *clipping*, ya que, si el umbral de corte es muy grande los pesos del discriminador tomaran mucho tiempo en alcanzar este umbral, por ende, no se forzará la restricción. Por otro lado, si el umbral de corte es muy pequeño, puede provocar problemas de desvanecimiento de gradiente cuando se tiene una gran cantidad de capas o no se usan capas de normalización de lotes (Gulrajani et al. (2017)).

Sin embargo, una función diferenciable f es 1-Lipchitz si y solo si, esta tiene gradiente con norma menor o igual a 1 en todo el dominio (Gulrajani et al. (2017)). Es por ello, que en lugar de aplicar la función *clipping* a los pesos, es más conveniente aplicar una penalización en la norma del gradiente en la función de costo del discriminador, para forzar la restricción. Esta modificación se expresa en la función de costo de ambas redes que quedan de la siguiente forma:

$$\min_{\mathcal{D}} \mathbb{E}_{z\sim p_z(z)}[\mathcal{D}(\mathcal{G}(z))] - \mathbb{E}_{x\sim p_{data}(x)}[\mathcal{D}(x)] + \lambda \mathbb{E}_{\hat{x}\sim p_{\hat{x}}(\hat{x})}[(\|\nabla_{\hat{x}}\mathcal{D}(\hat{x})\|_2 - 1)^2] \tag{3.15}$$

(a) Santa María **(b)** Terraza de cafe de Van Gogh **(c)** Santa María Van Gogh

Figura 3.3: Ejemplo de transferencia de estilo, en (a) se encuentra una fotografía de la Universidad Santa María Casa Central, (b) es el famoso cuadro de Van Gogh, Terraza de café en la noche, y (c) es la Santa María Casa Central al estilo de Van Gogh. Imágenes proporcionadas por el Dr. Werner Creixell.

$$\max_{\mathcal{G}} \mathbb{E}_{z\sim p_z(z)}[\mathcal{D}(\mathcal{G}(z))] \tag{3.16}$$

Donde $\hat{x}$ es un muestreo de los datos generados $\mathcal{G}(z)$ y los datos reales x, con t una muestra uniforme entre 0 y 1.

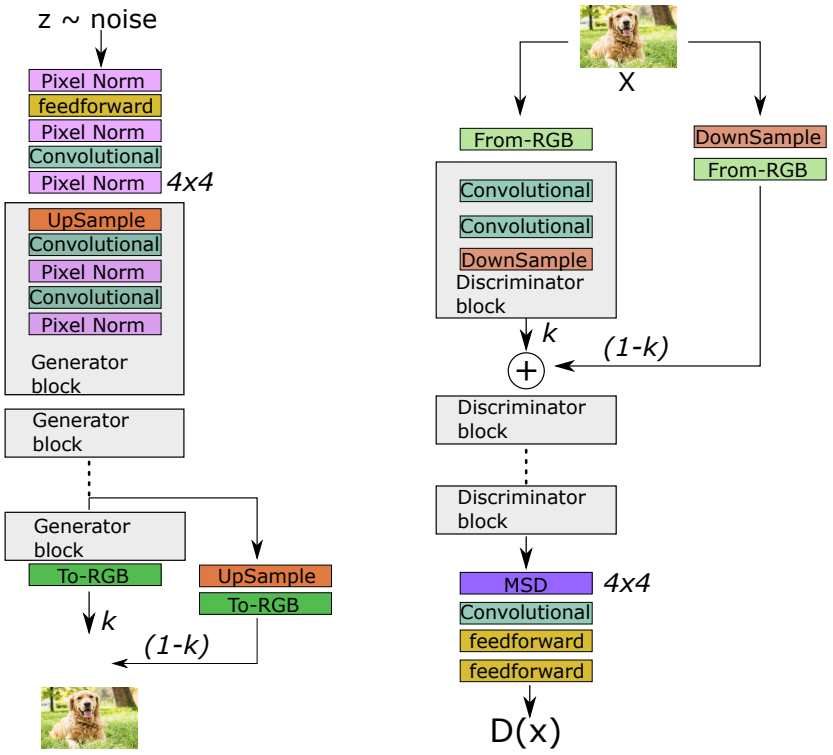
$$\hat{x} = t\mathcal{G}(z) + (1-t)x, \quad t \sim U(0,1) \tag{3.17}$$

3.4. GAN basada en estilo (StyleGAN)

La StyleGAN es una arquitectura GAN, que se cimenta en los conceptos y técnicas de **transferencia de estilos**. Estas técnicas consisten en crear una representación del estilo de una imagen, para traspasar esta información a una imagen cualquiera, véase 3.3. La StyleGAN utiliza este concepto para la generación de imágenes, separando la contribución del espacio latente en dos partes. El estilo, que indica los aspectos de alto nivel de una imagen como forma y color (ej. Un auto rojo). Las variaciones estocásticas en la imagen, que corresponden a las pequeñas diferencias y detalles en la imagen de carácter aleatorio, como los poros de la piel o el flequillo de pelo. Es por ello, la arquitectura StyleGAN separa el generador en dos sub redes neuronales, la red de mapeo de estilo f y la red de síntesis g, ambas cumplen que $\mathcal{G} = g \circ f$. De la misma forma, duplica las fuentes de ruido, una que entra por la red de mapeo f para generar una representación neural del estilo, la otra que se introduce en la red de síntesis g a través de inyecciones de ruido aditivo entre los bloques de capas. La información neural del estilo entra a la red de síntesis mediante capas de transferencia de estilos. Como resultado se obtienen imágenes de alta variabilidad y realismo.

3.4.1. Arquitectura de la StyleGAN

La arquitectura de la StyleGAN se asienta en la **GAN con crecimiento progresivo** (Karras et al. (2017)), cuya principal característica es el método de entrenamiento que le otorga el nombre a la arquitectura, el cual consiste en agregar bloques de capas neuronales que cambian la resolución (aumentan o disminuyen



(a) Discriminador **(b)** Generador

Figura 3.4: Arquitecturas del Discriminador (a) y Generador (b) de la red GAN con crecimiento progresivo, la cual es la base para la StyleGAN. *Pixel Norm* corresponde a una capa que no posee parámetros entrenables, cuya funcionamiento es similar a la capa de normalización de lote, y se encarga de normalizar con respecto a la norma por cada canal de la imagen. *To-RGB* y *From-RGB*, son capas convoluciones cuyo kernel es de 1 y 3×1, y su función es pasar de mapa de característica que posee un canal a imagen a color de tres canales y viceversa. MSD o *Minibatch Standard Derivation*, es una capa sin parámetros entrenables y su función es calcular la desviación estándar del mapa de característica, y anexarla como un canal.

dependiendo si son bloques de generador o discriminador respectivamente), mientras entrena. Esto le permite generar imágenes de alta resolución. Además, todas las capas con parámetros entrenables, como *feedforward* o convolucional, se encuentran ecualizadas. Esto quiere decir, presentan algunas alteraciones para cambiar el comportamiento original de las capas, en primer lugar, todos los parámetros entrenables están escalados por un factor cuya función es escalar el valor efectivo de la tasa de aprendizaje de los optimizadores. Por otro lado, se escalan las matrices de pesos y kernels dependiendo de la dimensión de entrada y salida como se aprecia en la ecuación (3.18), además de agregar una ganancia final a la capa.

$$W_f = W_i \sqrt{\frac{2}{k * k * c}}, \tag{3.18}$$

donde k es la dimensión de entrada y c la cantida de canales.

La StyleGAN propuesta por Karras et al. (2019), utiliza el método de entrenamiento progresivo, además de los mismos bloques neuronales y capas ecualizadas a excepción de las capas de *upsample* y *downsample*, que se reemplazan por **muestreos bilineales**, como se aprecia en la figura 3.5. Se modifica la arquitectura del generador $\mathcal{G}$ dividiéndolo en dos redes, **síntesis** g y **mapeo** f. La red de síntesis se encarga de generar la imagen, y se construye al igual que el generador de la GAN con crecimiento progresivo, pero con capas **AdaIN** (Huang y Belongie (2017)) entre las convoluciones encargadas de la transferencia de estilo. Además, se agrega ruido aditivo antes de cada moduló de AdaIN, que consiste en ruido gaussiano no correlacionado que entra a un módulo previo de escalamiento por canal entrenable B. Finalmente, la entrada

de la red de síntesis es una constante que se aprende y corresponde a la materia prima para la imagen.

Por otra parte, la capa de mapeo consiste en ocho capas *feedforward*, cuya entrada es el ruido proveniente del espacio latente $z \in \mathcal{Z}$, y la salida el vector de estilo que vive en un espacio latente intermedio $w \in \mathcal{W}$. Todas las capas de la red de mapeo de estilo f son de 512 nodos. El vector de estilo w pasa por una transformación afín A, entrenable (capa *feedforward* sin función de activación), para obtener el tensor $y = (y_s, y_b)$ compuesto por dos vectores que controlan el comportamiento de la capa AdaIN, cuya operación es:

$$AdaIN(x, y) = y_s \frac{x - \mu(x)}{\sigma(x)} + y_b. \tag{3.19}$$

Esta ecuación se asemeja a la normalización por lote con una transformación afín. Donde x es la entrada, $\mu(x)$ y $\sigma(x)$ son el promedio y la desviación estándar respectivamente, e $y = (y_s, y_n)$ los vectores que controlan la transformación afín.

La función de perdida de la StyleGAN es similar a la de la WGAN-GP, pero las salidas del discriminador entran a una función llamada **softplus**, que corresponde a una versión suave de la función ReLU. Además, utiliza dos métodos regularización, la **regularización R1** (Mescheder et al. (2018)) en el discriminador, que reemplaza la penalización de gradiente en la WGAN-GP y se diferencia de este debido a que solo utiliza las imágenes reales, como se muestra en la ecuación (3.21), donde R1 es el término que depende del gradiente. El otro método de regularización corresponde a la **mezcla de estilos**, cuyo funcionamiento es tomar dos muestras w_1 y w_2 del espacio latente $\mathcal{W}$, a partir del procesamiento de la red de mapeo f con dos muestras de ruido z_1 y z_2. Para luego, elegir un punto en la red de síntesis de forma aleatoria nombrado punto de *crossover*, y alimentar las capas AdaiIN antes del punto de *crossover* con w_1, y las capas después con w_2. Esta técnica de regulación ayuda a prevenir que la red suponga que los estilos adyacentes estén correlaciones.

$$softplus(x) = sp(x) = ln(1 + e^x) \tag{3.20}$$

$$\min_{\mathcal{D}} \mathbb{E}_{z\sim p_z(z)}[sp(\mathcal{D}(\mathcal{G}(z)))] - \mathbb{E}_{x\sim p_{data}(x)}[sp(\mathcal{D}(x))] + \frac{\gamma}{2}\mathbb{E}_{x\sim p_x(x)}[\|\nabla_x \mathcal{D}(\hat{x})\|_2^2] \tag{3.21}$$

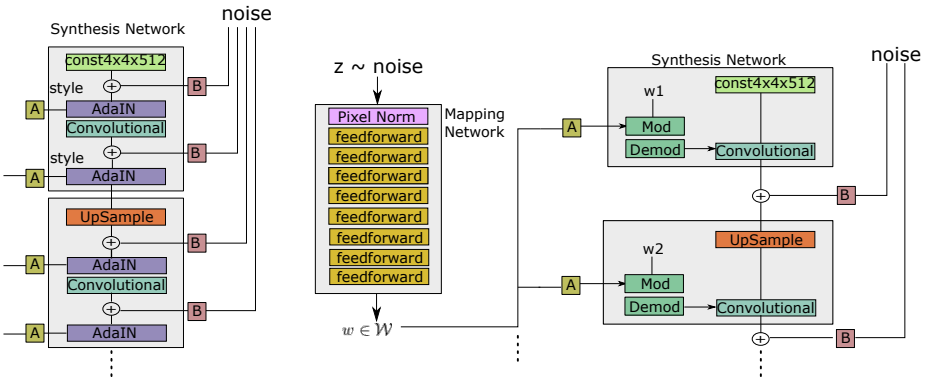
$$\min_{\mathcal{G}} \mathbb{E}_{z\sim p_z(z)}[sp(-\mathcal{D}(\mathcal{G}(z)))] \tag{3.22}$$

3.4.2. StyleGAN2

La arquitectura GAN basada en estilo ha significado en un gran avance con respecto a la generación de imágenes sin información condicional. Sin embargo, es posible mejorar la arquitectura StyleGAN respecto a los métodos de procesamiento y entrenamiento. Karras et al. (2020) analizaron esta arquitectura y proponen distintas configuraciones para mejorar el rendimiento final. El primer tópico revisado es el bloque de transferencia de estilo, la capa AdaIN, que se puede dividir en dos procesos, una normalización estadística por media y varianza, y una transformación afín por la entrada y como se aprecia en la ecuación (3.19). Se reemplaza la capa AdaIN por la modulación directa de los pesos de las capas de convolución, como se muestra en la figura 3.5. El primer proceso consiste en modular los pesos de la capa convolucional de la siguiente forma:

$$w'_{ijk} = s_i w_{ijk} \tag{3.23}$$

Donde w y w' corresponden a el peso original y modulado respectivamente, el factor de escalamiento s_i (que depende del estilo), i indica el i-ésimo mapa de características de entrada, j y k enumeran el mapa de característica de salida y la marca espacial de la convolución respectivamente. Luego es necesario remover los



(a) StyleGAN **(b)** StyleGAN2

Figura 3.5: Arquitecturas de la red de Síntesis para la StyleGAN (a) y la StyleGAN2 (b), además de la red de mapeo que para ambas arquitecturas es la misma, 8 capas *feedforward*.

efectos estadísticos de s en la convolución, por ello se implementa una normalización estadística considerando media cero, con ϵ constante para no caer en problemas numéricos.

$$w''_{ijk} = \frac{w'_{ijk}}{\sqrt{\sum_{ik}(w'_{ijk})^2 + \epsilon}} \tag{3.24}$$

Otro punto estudiado es la regularización diferida, que consiste en aplicar los términos de regularización cada n cantidad de iteraciones, logrando los mismos resultados y reduciendo costo computacional. Además, se agrega regularización de longitud de la ruta, la cual se aplica en la función de pérdida del generador, ya que, se encontró una correlación entre la métrica de longitud de la ruta de percepción (véase en la sección de métricas 3.5.3) y la calidad de las imágenes, su cálculo se detalla en la ecuación (3.25). También, se descartó el entrenamiento con crecimiento progresivo debido a que no presentaba una mejora en el rendimiento. Finalmente, ya que no se requiere una arquitectura en forma de bloques consecutivos necesaria para el método de crecimiento progresivo, se estudiaron nuevas arquitecturas para la red de síntesis, tales como arquitecturas basadas en **omisión de conexiones** y **redes residuales**(He et al. (2016)).

$$\mathbb{E}_{w\sim f(z),y\sim\mathcal{N}(0,I)}[\|J_w^T y\|_2 - a]^2. \tag{3.25}$$

Donde J_w corresponde a $\nabla_w \mathcal{G}(z)$, con y que se distribuye $y \sim \mathcal{N}(0, I)$, y a es el promedio móvil de la perdida de ruta anteriores con factor $\beta_{pl} = 0{,}99$.

3.5. Métricas para GAN

Tener una forma de medir que tan bueno es el desempeño de un modelo generativo, es una tarea necesaria para su investigación y desarrollo. En primera instancia, los modelos generativos se median con la ayuda de voluntarios, los cuales comparaban las imágenes generadas con las reales, donde los investigadores comparaban estos resultados y realizaban estadísticas. Este procedimiento dificulta el desarrollo, ya que, se requiere fijar a los voluntarios en días específicos, y no siempre cuando los investigadores lo requieran. Por ello, metodologías para medir de forma automática los modelos generativos han provocado un impacto importante en esta área, reemplazando el uso de personas voluntarias por la inteligencia artificial, en específico modelos de clasificación de imágenes. Las métricas propuestas han permitido evaluar el desempeño de muchos modelos generativos, y dar un punto de comparación para los investigadores.

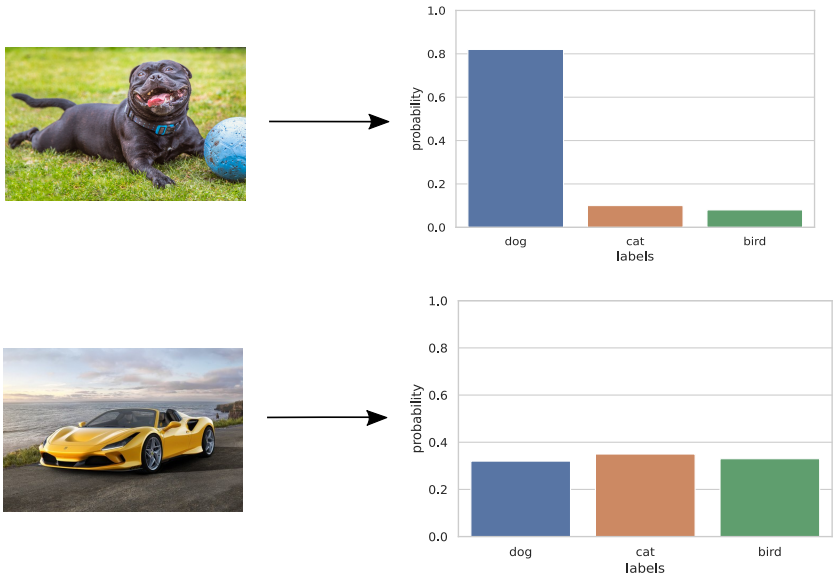


Figura 3.6: Ilustración resultado de clasificación obtenido de red Inception.

3.5.1. Inception Score

La métrica *Inception Score* (IS) es una de las más populares para medir cuan realista es el resultado de las imágenes generadas de las arquitecturas GAN's (Salimans et al. (2016a)). Esto, gracias a la correlación presente entre los resultados del IS y evaluaciones humanas, por ende, esta métrica es una gran alternativa para automatizar el proceso de evaluación de los modelos generativos. El IS mide simultáneamente dos propiedades en las imágenes generadas, la variabilidad y la calidad. Si se cumplen ambas propiedades en las imágenes generadas, el valor arrojado por la métrica IS será alto, en contraparte, si existen problemas con uno o ambas propiedades el valor final será bajo. Entonces, para dos modelos generativos se dice que un modelo es mejor a otro si presenta un valor de IS más alto. La medición entregada por la métrica IS, tiene como cota inferior cero y como cota superior teóricamente infinito, pero en la práctica los resultados con un buen IS se acercan al valor IS del conjunto de datos real.

La métrica IS se le debe a su nombre a que utiliza el clasificador neuronal de imágenes ***Inception***, desarrollado por Salimans et al. (2016b), la cual entrega la distribución discreta correspondiente a la probabilidad de las clases presentes en la imagen. Si se introduce una imagen de un perro a la red *Inception*, la distribución resultante tendrá un pico en la clase perro, a diferencia del resto de las clases que tendrán baja probabilidad. Por otro lado, si la imagen contiene un objeto fuera del conjunto de las clases utilizadas para entrenar la red, la distribución resultante será muy cercana a una distribución uniforme, ya que no existe certeza con respecto a la clase presente en la imagen. Esto se ejemplifica en la figura 3.6. Gracias a esto, utilizando el modelo *Inception* es posible medir las propiedades de las imágenes generadas, sin embargo, el modelo debe estar entrenado con el mismo conjunto de datos con el cual se entrenó la red GAN.

Para determinar la variabilidad de las imágenes, es necesario estudiar las distribuciones de clase obtenidas desde la red *Inception* de todas las imágenes de la muestra (lo común es utilizar una cantidad de 10.000 imágenes). Sumando las distribuciones obtenidas de cada imagen se crea una nueva distribución, denominada distribución marginal, donde si esta se asemeja a una uniforme significa que las clases presentes en las imágenes están balanceadas, tal como ilustra la figura 3.7. Note que, es deseable tener formas opuestas entre la distribución de una clase, de forma estrecha, y la distribución marginal, de forma uniforme. Por ende, midiendo la diferencia entre la distribución de clase de cada imagen con la distribución marginal, es posible cuantificar el realismo de la muestra. Para ello, se utiliza la divergencia KL que se ilustra en la ecuación (3.9), tal que, si la divergencia es alta, entonces las distribuciones son muy diferentes, caso contrario, las distribuciones son similares. Finalmente, el valor del *Inception Score* se consigue promediando

Figura 3.7: Comparación entre la distribución marginal para un conjunto de imágenes balanceado (izquierda) y un conjunto de datos des balanceado (derecha), en este caso hacia la clase perro.

las divergencias KL entre cada distribución de clase de las imágenes con respecto a la distribución marginal, en función exponencial.

3.5.2. Fechet Inception Distance

La técnica de *Inception Score* es un gran avance para las arquitecturas GAN, debido a que permite medir el desempeño de los diferentes modelos, determinando la calidad de las imágenes generadas sin intervención humana. Sin embargo, el IS presenta desventajas claras debido a que no utiliza muestras reales o del conjunto de datos de entrenamiento, para comparar las estadísticas de los ejemplos generados (Heusel et al. (2017)). Esto se soluciona con la introducción de la métrica **Fréchet Inception Distance** (FID), el cual es una distancia entre las distribuciones de las imágenes de entrenamiento y la distribución de las imágenes generas, y no un simple puntaje.

Definición 3.5.1 (Igualdad entre distribuciones)**.** Sea $p(x)$ la distribución de los datos generados y $p_w(x)$ la distribución de los datos reales. La igualdad $p(x) = p_w(x)$ se mantiene excepto para un conjunto no medible, si y solo si:

$$\int p(x)f(x)dx = \int p_w(x)f(x)dx, \tag{3.26}$$

para una base $f(x)$ del espacio de funciones donde pertenecen $p(x)$ y $p_w(x)$, donde $f(x)$ son polinomios de la variable x. Es deseable trabajar con características relevantes en visión, por ello, en lugar de trabajar directamente con los datos o imágenes, x corresponde al mapa de características antes de la capa final de la red *Inception*. Por razones prácticas, solo se consideran los dos primeros momentos que corresponden a la media y la varianza. Por otra parte, la distribución con máxima entropía para una media y varianza dada corresponde a la distribución gaussiana, entonces para el cálculo de la distancia se supone que ambas distribuciones son gaussianas multidimensionales. La diferencia entre las dos distribuciones gaussianas se mide mediante la **distancia Fréchet**, o también conocida como **distancia Wasserstein-2**. La distancia Fréchet entre dos distribuciones gaussianas con media y varianza (m, C) obtenida desde $p(x)$ y la gaussiana con (m_w, C_w) obtenida de $p_w(x)$, es llamada **Fréchet Inception Distance**(FID), la cual es dada por:

$$d^2((m, C), (m_w, C_w)) = \|m - m_w\|_2^2 + Tr(C + C_w - 2(CC_w)^{1/2}) \tag{3.27}$$

3.5.3. Largo perceptivo de ruta o *Perceptual Path Lenght*(PPL)

El PPL es una métrica introducida en la publicación GAN basada en estilo o StyleGAN (Karras et al. (2019)), que indica si una imagen cambia suavemente desde el punto de vista **perceptivo**. Específicamente,

esta métrica indica si la imagen cambia en la ruta *perceptiva* más corta en el espacio latente. Supóngase que se tiene tres muestras del espacio latente z_1, z_2 y z_3, que posteriormente se introducen al generador. La muestra z_1 hace que la imagen generada sea un perro blanco, z_2 genera la imagen de un perro negro, y z_3 genera la imagen de un auto azul. Dado que los elementos generados a partir de z_1 y z_2 no cambian (ambos son perros, pero de distinto color), el camino dentro del espacio latente entre z_1 y z_2 resulta en el camino más corto desde el punto de vista *perceptivo*. Si se selecciona una muestra z_4, que este en el camino más corto, entonces la imagen generada a partir de z_4 resulta en un perro de otro color. Por otro lado, la distancia que genera el cambio de forma de los elementos (perro a auto), resulta en una distancia *perceptiva* más larga. Esto se cuantifica mediante la métrica PPL, la cual se calcula con la siguiente ecuación:

$$l_{\mathcal{W}} = \mathbb{E}\left[\frac{1}{\epsilon^2} d\left(g(lerp(f(z_1), f(z_2); t)),\ \ g(lerp(f(z_1), f(z_2); t+\epsilon))\right)\right]. \tag{3.28}$$

Donde g es la red de síntesis del generador, f es la red de mapeo (recuerde $\mathcal{G} = g \circ f$), $lerp$ es una interpolación lineal, t variable aleatoria que distribuye $t \sim U(0,1)$ y $d(\cdot,\cdot)$ es la distancia perceptiva, que se implementa como una diferencia con pesos sobre los mapas de características de alto nivel obtenidos del clasificador neural **VGG16** (Simonyan y Zisserman (2014)) entrenada con el conjunto de datos ImageNet. En la práctica, la esperanza se calcula a partir de 100000 muestras y épsilon se escoge de un valor igual a 10^{-4}.

4 | Aprendizaje Profundo Hiperbólico

El enfoque del aprendizaje profundo consiste en crear características abstractas o representaciones jerárquicas a partir de los datos de entrada, mediante transformaciones no lineales llamadas capas. Las relaciones y patrones que pueden establecer las capas neuronales están fuertemente determinados por la suposición que los datos de entrada son representados en el **espacio euclídeo**. Esta suposición ha sido exitosa en medida en que los datos, tales como imágenes, audio, texto, y otras señales, tienen una estructura euclídea subyacente. Los datos euclídeos se pueden representar como muestras de una grilla, donde es posible aplicar métricas del espacio euclídeo. Sin embargo, existen datos que presentan una pobre representación en el espacio euclídeo, por ende, las capas neuronales no funcionan correctamente, este tipo de datos se les denomina datos no euclídeos.

El campo del **aprendizaje profundo geométrico** o **aprendizaje profundo no euclídeo**, es un nuevo enfoque para producir mejores representaciones para datos no euclídeos que las redes neuronales convencionales. En general, se identifican dos tipos de problemas distintos dependiendo del tipo de dato con el cual se trabaja (Bronstein et al. (2017)), en primer lugar, se encuentran los datos que poseen una clara naturaleza no euclídea, por ejemplo, grafos o formas 3D. El segundo tipo de datos corresponden a datos que se encuentran incrustados en un espacio euclídeo de gran dimensionalidad, pero poseen una estructura latente no euclídea de baja dimensión. Por ejemplo, si los datos presentan estructura cíclica o jerárquica, espacios no euclídeos de curvatura constante son perfectos para lograr una buena representación, en estos casos corresponden a espacios esféricos para la estructura cíclica e hiperbólicos para estructuras jerárquicas(Gu et al. (2019)). El enfoque de esta tesis radica en estos últimos tipos de problemáticas, específicamente en los datos los cuales presentan una estructura jerárquica, por ende, se dispondrá a utilizar espacios hiperbólicos los cuales presentan una curvatura constante negativa.

En este capítulo se revizaran los conceptos de **Espacios Hiperbólicos**, luego se detallara el modelo de **Bola de Poincaré**, el cual es uno de los espacios hiperbólicos mas estudiados y uno de los mas utilizados en el campo de la aprendizaje automático, para finalmente ver como explotan estas propiedades en el Deep Learning con las **Redes Neuronales Hiperbólicas** y algunas aplicaciones.

4.1. Geometría No Euclidiana y Variedades Rimannianas

La geometría euclidiana es el sistema geométrico definido sobre un espacio que cumple los cinco postulados de Euclides expresando en su libro **Los Elementos** (Weisstein (2002)). Cuando un sistema geométrico no cumple o difieren en algún postulado se le denomina geometría no euclídea. El cumplimiento de los postulados de Euclides se debe a que la geometría es plana, no presenta curvatura, donde el concepto de curvatura se puede entender como la cantidad en que una figura geométrica difiere de su contra parte plana, por ejemplo, una curva de una línea recta. La generalización matemática de un espacio curvado o con curvatura se denomina **variedad riemanniana** o **variedad diferencial**, y los sistemas geométricos que se definen en una variedad riemanniana se le conoce como geometría rimanniana.

Una variedad riemanniana $(\mathcal{M}, g)$, donde $\mathcal{M}$ es un conjunto de puntos z que es localmente similar a un espacio euclídeo y g el tensor métrico. Para cada punto $z \in \mathcal{M}$ se adjudica un espacio vectorial denominado **espacio tangente** $\mathcal{T}_z\mathcal{M}$. El espacio tangente corresponde al espacio que contiene todas las direcciones

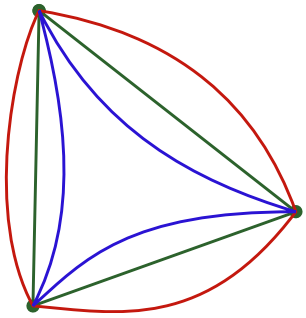


Figura 4.1: Ejemplo de triángulos, objeto geométrico formado por tres puntos y sus tres geodésicas, en distintos espacios. Triangulo en euclídeo (verde), hiperbólico (azul) y elíptico (rojo). Se ejemplifica que la suma de los ángulos interiores en los diferentes triángulos son menor, igual y superior, para los espacios hiperbólicos, euclídeo y elíptico respectivamente.

posibles o derivadas direccionales que pude pasar z. Para definir el producto interno en un punto $z \in \mathcal{M}$, entra en juego el otro elemento que define a una Variadad Riemmaniana, que es la función **tensor métrico** g, que se define como:

$$g(z) = \langle \cdot, \cdot \rangle_z : \mathcal{T}_z\mathcal{M} \times \mathcal{T}_z\mathcal{M} \to \mathbb{R}. \tag{4.1}$$

El tensor métrico es un producto interno de espacio tangente de z, por ende, cumple las propiedades de **simetría** ($g(z, y) = g(y, z)$) y **positiva definida** ($g(x, x) > 0$, si $x > 0$). Además, si se tiene el par $(u, v) \in \mathcal{T}_z\mathcal{M} \times \mathcal{T}_z\mathcal{M}$ el tensor métrico se puede definir de forma matricial $G(z)$ como:

$$\langle u, v \rangle_z = u^t G(z) v. \tag{4.2}$$

La norma inducida por el producto interno sobre $\mathcal{T}_z\mathcal{M}$ corresponde a:

$$\| \cdot \|_z = \sqrt{\langle u, v \rangle_z}. \tag{4.3}$$

La Variedad Riemanniana queda completamente definida como el par $(\mathcal{M}, g)$. Con el tensor métrico es posible dar una noción local de ángulo, largo de las curvas, volumen y área de una superficie, donde es posible obtener cantidades globales integrando las contribuciones locales. El largo de una curva $\gamma : t \mapsto \gamma(t) \in \mathcal{M}$ es dado por:

$$L(\gamma) = \int_{t_a}^{t_b} \sqrt{\| \frac{d(\gamma(t))}{dt} \|_{\gamma(t)}} dt \tag{4.4}$$

El concepto de lineas rectas se puede generalizar mediante las **geodésicas**, que son curvas de velocidad constante que dan el camino más corto entre los puntos z e y en la variedad: $\gamma^* = argmin L(\gamma)$ con $\gamma(t_a) = z$, $\gamma(t_b) = y$ y $\|\gamma'(t)\|_{\gamma(t)} = 1$. El concepto de estar moviéndose a lo largo de una curva recta con velocidad constante lo entrega el **mapeo exponencial**. En particular, para un vector de velocidad $v \in \mathcal{T}_z\mathcal{M}$, hay una única geodésica γ_v que satisface que $\gamma_v(t_a) = z$, cuyo vector tangente o de velocidad inicial es igual a $\gamma_v'(t_a) = v$, entonces se define el mapeo exponencial $exp_z(v) = \gamma(t_b)$. El mapa logarítmico es la función inversa y se define como $log_z = exp_z^{-1} : \mathcal{M} \to \mathcal{T}_z\mathcal{M}$.

Las variedades riemannianas se caracterizan por tener curvatura, la cual se encuentra fuertemente relacionada con las geodésicas de la varianza. Por ejemplo, no existe curvatura si la geodésica corresponde a una línea plana, en contra parte, se aprecia la existencia de una curvatura si la geodésica difiere de la línea plana. Un espacio particular en las variedades rimanniana, son los llamados espacios **homogéneos** que se definen como espacios que poseen curvatura constante. En esta categoría encontramos los espacios hiperbólicos, elípticos y euclídeo (plano), que tienen curvatura constante negativa, positiva y cero respectivamente. Los

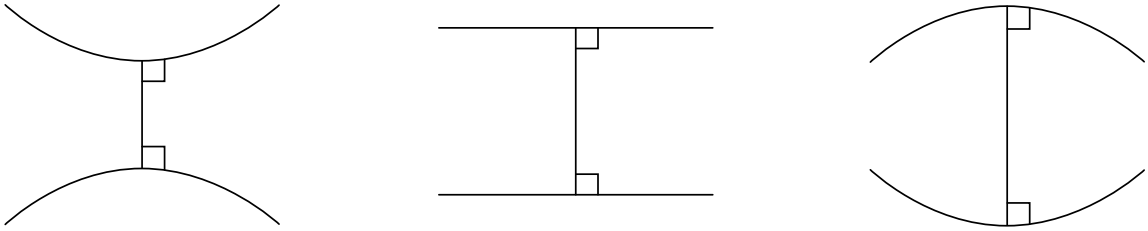


Figura 4.2: Los tres casos de una recta que incide sobre dos rectas formando ángulos rectos, en los tres diferentes espacios, hiperbólico, euclídeo y elíptico. Fuente: De derivative work: Pbroks13 (talk)Noneuclid.png: Original uploader was Joshuabowman at en.wikipedia - Noneuclid.png, CC BY-SA 3.0, https://commons.wikimedia.org/w/index.php?curid=4373343

sistemas geométricos derivados de los espacios hiperbólicos y elípticos se consideran geometría no euclídea debido a que no cumplen el **quinto postulado de Euclides**, el cual dicta que: *Para un punto exterior a una recta dada, es posible trazar una única recta paralela*. Este postulado es equivalente a: *La suma de los ángulos interiores de un triángulo es igual a 180 grados*. En los espacios elípticos la suma de los ángulos interiores de un triángulo suma más de 180 grados, y para una recta y un punto que no pertenece a la recta, no es posible trazar una recta paralela que pase por el punto. Por otro lado, en los espacios hiperbólicos la suma de los ángulos interiores de un triángulo suma menos de 180 grados y para una recta y un punto que no pertenece a esta, existen más de una recta paralela que pase por el punto. Las figuras 4.1 y 4.2 ilustran estas propiedades para los espacios elíptico, hiperbólico y euclídeo.

4.2. Espacios Hiperbólicos

Un espacio Hiperbólico es una variedad Riemanniana que posee curvatura constante y negativa. Esto se aprecia gráficamente observando las geodésicas, que al tener curvatura negativa estas tenderán a divergir entre sí (es por ello los ángulos interiores de un triángulo suman menos de 180). La naturaleza de los espacios hiperbólicos les permite representar de forma fidedigna estructuras con forma de árbol, que corresponde a un objeto matemático discreto especial para modelar la jerarquía entre elementos. Donde los nodos más cercanos a la raíz poseen un mayor grado de jerarquía que los nodos cercanos a las hojas. Por este motivo, se afirma que los espacios hiperbólicos poseen naturaleza jerárquica intrínseca, y resultan útiles en el campo de procesamiento de datos para cuantificar su jerarquía mediante representaciones vectoriales hiperbólicas. Existen múltiples modelos de espacios hiperbólicos, los cuales se relacionan a través de isometrías (transformaciones que preservan las medidas en las métricas en ambos espacios), donde en el campo del aprendizaje de máquinas los espacios ampliamente utilizados son: El espacio hiperbólico de **Lorentz** o **Hiperboloide**, y el modelo de **bola de Poincaré**.

Un árbol es un grafo acíclico que no presenta saltos entre niveles, cuya distancia entre los nodos a y b del grafo es la cantidad mínima de saltos entre nodos para llegar desde a hasta b. Supóngase un árbol con un factor de ramificación b, si se considera la analogía de circunferencia o disco dentro del árbol, entonces estos corresponden a los nodos que se encuentran a una distancia menor o igual r saltos desde la raíz. Entonces el largo y el área de la circunferencia estarían dados por los números $(b+1)b^{r-1}$ y $[(b+1)b^r - 2]/(b-1)$ respectivamente, donde ambos crecen de forma b^r con r. Por otro lado, considérese un espacio hiperbólico de dos dimensiones cualquiera $\mathbb{H}^2_\zeta$, cuya curvatura constante negativa es igual a $K = -\zeta^2 < 0, \zeta > 0$. El largo L y área A de una circunferencia de radio r en el espacio $\mathbb{H}^2_\zeta$, está dado por las expresiones (Krioukov et al. (2010)):

$$L(r) = 2\pi sinh(\zeta r). \tag{4.5}$$

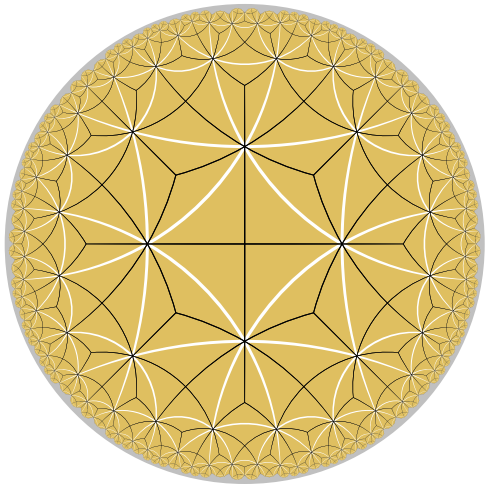


Figura 4.3: Teselación de cuadrados (polígono regular de cuatro lados) en la bola de Poincaré 2D. Los trazos blancos indican una diagonal del cuadrado.

$$A(r) = 2\pi(cosh(\zeta r) - 1). \tag{4.6}$$

Ambas expresiones de largo y área de una circunferencia en un espacio hiperbólico crecen respecto $e^{\zeta r}$ con r. Si $\zeta = ln(b)$, entonces $\mathbb{H}^2_{ln(b)}$ y un árbol con factor de ramificación b son equivalentes con respecto a la métrica, es por ello que informalmente es factible decir *un árbol es una discretización del espacio hiperbólico* o *un espacio hiperbólico es la versión continua a un árbol*. Un ejemplo de esta propiedad radica, en que cualquier teselación en un plano hiperbólico define naturalmente una incrustación isométrica, para una clase de árbol formado por ciertos subconjuntos de lados de un polígono, como se ilustra en la figura 4.3. Para generar una incrustación de un árbol en un espacio cualquiera, es necesario que el espacio crezca de forma exponencial al igual que las ramificaciones del árbol, esto se cumple en un espacio hiperbólico a diferencia de un espacio euclídeo cuyo crecimiento es cuadrático.

4.3. Modelo de Bola de Poincaré

El modelo de bola de Poincaré es un modelo de espacio hiperbólico n-dimensional que se define dentro de un espacio Euclídeo de misma dimensión n. Este modelo corresponde a n-bola en un espacio Euclídeo, donde el radio R es igual a $1/\sqrt{c}$

La bola de Poincaré se define como la variedad de Riemanniana $(\mathbb{D}^n_c, g^{\mathbb{D}})$, donde:

$$\mathbb{D}^n_c = \{u \in \mathbb{R}^n : c\|u\|^2 < 1, c \geq 0\} \tag{4.7}$$

Cuyo tensor métrico esta determinado por:

$$g^{\mathbb{D}_c}_u = \lambda^c_u g^E, \quad Donde\ \lambda^c_u := \frac{2}{1 - c\|u\|^2} \tag{4.8}$$

λ^c_u se conoce como el factor conformal del espacio, y g^E corresponde al tensor métrico del espacio euclídeo de orden n en bases canónicas, que es igual a la matriz identidad de orden n. Desde el tensor métrico se deriva la función de distancia que corresponde a:

$$d(\cdot,\cdot) : \mathbb{D}_c^n \times \mathbb{D}_c^n \to \mathbb{R}$$
$$d(x,y) = cosh^{-1}\left(1 + \frac{\lambda_x \lambda_y}{2}\|x-y\|^2\right) \tag{4.9}$$

Sea $\mathcal{T}_u$ el operador de tangencia en el punto $u \in \mathbb{D}_c^n$, entonces el espacio tangente en un punto u es igual a $\mathcal{T}_u\mathbb{D}_c^n \cong \mathbb{R}^n$, es decir, el espacio plano. Debido a que el espacio tangencial de la bola de Poincaré corresponde a todo el espacio real n-dimensional, entonces la operación del tensor métrico está definida en $\mathbb{R}^n \times \mathbb{R}^n$. Esto resulta útil, ya que, mediante los operadores de mapeo exponencial y mapeo logarítmico es posible pasar desde el espacio euclídeo a la bola de Poincaré y viceversa. El mapeo exponencial y el logarítmico se definen como:

$$\exp_{\mathbf{u}}^c(\mathbf{x}) := (\mathbf{u}) \oplus_c \left(\tanh\left(\sqrt{c}\,\frac{\|\mathbf{x}\|\lambda_u^c}{2}\right)\frac{\mathbf{x}}{\sqrt{c}\|\mathbf{x}\|}\right) \tag{4.10}$$

$$\log_{\mathbf{u}}^c(\mathbf{v}) := \frac{2}{\sqrt{c}\,\lambda_{\mathbf{u}}^c}\tanh^{-1}(\sqrt{c}\|-\mathbf{u}\oplus_c\,\mathbf{v}\|)\frac{-\mathbf{u}\oplus_c\mathbf{v}}{\|-\mathbf{u}\oplus_c\mathbf{v}\|} \tag{4.11}$$

Donde $\oplus_c$ corresponde a la suma de Mobiüs, la cual se verá más adelante. Si se considera el mapeo exponencial y el mapeo logarítmico con respecto al punto $\vec{0}$ se obtiene:

$$\mathbf{v} = \exp_0^c(\mathbf{x}) := \tanh(\sqrt{c}\|\mathbf{x}\|)\frac{\mathbf{x}}{\sqrt{c}\|\mathbf{x}\|} \tag{4.12}$$

$$\mathbf{x} = \log_0^c(\mathbf{v}) := \tanh^{-1}(\sqrt{c}\|\mathbf{v}\|)\frac{\mathbf{v}}{\sqrt{c}\|\mathbf{v}\|} \tag{4.13}$$

Tanto en el mapeo $\exp_0^c(x)$ como el mapeo $\log_0^c(\mathbf{v})$, se aprecia que el ángulo del vector de entrada no se modifica, a excepción de la magnitud. La magnitud cambia a razón de las funciones, *tanh* y $tanh^{-1}$ de la norma sobre el radio de la bola. En la bola de Poincaré no están definidos los vectores con norma igual o superior a $1/\sqrt{c}$, y la circunferencia $\{v \in \mathbb{R}^n : \|v\|^2 = 1/c\}$ corresponde a infinito (análogo a la esfera de Riemman, que el punto $(0,0,1)$ representa infinito). Cabe destacar, si el vector se encuentra en la circunferencia el cálculo del factor conformal (4.8) queda indefinido.

4.3.1. Espacios Gyrovectoriales de Möbius

Para trabajar con información numérica se requiere un sistema algebraico, el cual se define como un conjunto no nulo y un conjunto de operaciones. A través de un sistema algebraico es posible realizar álgebra en algún conjunto de elementos cualquiera, donde el sistema algebraico que define el álgebra lineal corresponde al espacio vectorial. Un espacio vectorial se constituye por un conjunto de vectores V, una operación binaria $\oplus$ denominada **adición vectorial**, y una operación externa $\otimes$ sobre un cuerpo K, llamada **producto por escalar**. El sistema algebraico $(V, \oplus)$ cumple con los requisitos de grupo conmutativo o abeliano. Estos requisitos son: propiedad asociativa, propiedad conmutativa y existencia elemento neutro e inverso.

Por otro lado, un cuerpo K es otro sistema algebraico formado por un conjunto K no vació, dotado de las operaciones binarias + y $\cdot$ llamadas adición y multiplicación respectivamente. Donde, los pares $(K, +)$ y $(K - \{inverso\,aditivo\}, \cdot)$ son grupos conmutativo, pero la multiplicación, además debe cumplir la propiedad distributiva con respecto la adición $(c \cdot (a + b) = c \cdot a + c \cdot b)$. Por ejemplo, los números reales $\mathbb{R}$ con las operaciones adición y multiplicación comunes son un cuerpo. Con el cuerpo K, es posible definir la operación multiplicación por escalar, para finalmente tener todas las propiedades de un espacio vectorial. La operación producto por escalar es una operación externa, ya que, es una operación binaria que se aplica sobre dos conjuntos distintos, el cuerpo K y el conjunto de vectores V, de la forma $\otimes : V \times K \to V$. La multiplicación

por escalar cumple con las propiedades: asociativa, existencia de elemento neutro y distributiva respecto a la suma vectorial $\oplus$ y suma escalar $+$. En resumen, un espacio vectorial se construye con un grupo conmutativo y un producto escalar sobre un cuerpo K.

El espacio girovectorial propuesto por Ungar (2008), corresponde a la extensión del sistema algebraico espacio vectorial, y permite definir álgebra sobre un espacio no euclídeo consistente con el tensor métrico. El primer acercamiento a los girovectores es definir el sistema girogrupo, que es una relajación del sistema grupo, y para ello es necesario introducir los conceptos algebraicos que permitirán construirlo.

Definición 4.3.1 (Grupoide)**.** Un grupoide o **magma**, consiste en un par $(G, +)$, donde G es un conjunto no vacío y $+$ una operación binaria.

Definición 4.3.2 (Automorfismo de un Grupoide)**.** Un automorfismo ϕ de un grupoide $(G, +)$ es una función auto mapeo de G biyectiva (mapeo uno a uno) $\phi : G \to G$, que preserva la operación del grupoide, que es, $\phi(a + b) = \phi(a) + \phi(b)$.

La operación binaria en un conjunto dado se conoce como **operación de conjunto**. Por otra parte, el conjunto de todos los automorfismos de un grupoide $(G, +)$ se denota como $Aut(A, +)$, donde la función identidad I, tal que $I(x) = x$, corresponde al automorfismo trivial. Con todas estas definiciones, Ungar (2008) generalizo el concepto de grupo relajando algunos axiomas, para así definir un sistema algebraico en un espacio no euclídeo, el cual nombro girogrupo.

Definición 4.3.3 (Girogrupo)**.** Un girogrupo es un par $(G, \oplus)$, donde este debe cumplir las siguientes propiedades:

- Existencia del elemento neutro e.
- Existencia del inverso por la izquierda: $\forall a \in G, \exists \bar{a} \in G : \bar{a} \oplus a = e$.
- Propiedad de giroasociatividad: $a \oplus (b \oplus c) = (a \oplus b) \oplus gyr[a, b]c, \forall a, b, c \in G$, donde, $gyr[a, b]c \in G$ y es único.
- El mapeo $gyr[a, b] : G \to G$ dado por $c \to gyr[a, b]c$ es un automorfismo del grupoide $(G, \oplus)$, es decir, $gyr[a, b] \in Aut(G, \oplus)$. $gyr[a, b]$ de G se le llama giromorfismo, o giro de G generado por $a, b \in G$. El operador $gyr : G \times G \to Aut(G, \oplus)$ se denomina giro de G.
- Propiedad de bucle: $gyr[a, b] = gyr[a \oplus b, b]$.

Definición 4.3.4 (Girogrupo conmutativo)**.** Un girogrupo $(G, \oplus)$ es conmutativo si el operador binario cumple con la propiedad de gyroconmutatividad: $a \oplus b = gyr[a, b](b \oplus a)$.

Claramente un grupo (conmutativo), consiste en un girogrupo (giroconmutativo) cuyo giroautomorfismo es trivial. La estructura algebraica de los girogrupos es en consecuencia más rica que un grupo. Con respecto al modelo de la bola de Poincaré se encuentra el girogrupo de Möbius, que se define como el par $(\mathbb{D}_c^n, \oplus_c)$, donde $\oplus_c$ corresponde a **la adición de Möbius**. Para $x, y \in \mathbb{D}_c^n$, la adición de Möbius $\oplus_c$ es una operación binaria tal que:

$$\oplus_c : \mathbb{D}_c^n \times \mathbb{D}_c^n \to \mathbb{D}_c^n$$
$$x \oplus_c y := \frac{\left(1 + 2c\langle x, y\rangle + c\|y\|^2\right) x + \left(1 - c\|x\|^2\right) y}{1 + 2c\langle x, y\rangle + c^2\|x\|^2\|y\|^2} \tag{4.14}$$

Donde el operador de giro (gyr) del grupo de Möbius se muestra en la ecuación (4.15). En particular, cuando $c = 0$ se obtiene la suma ordinaria de dos vectores en $\mathbb{R}^n$. La adición de Möbius no cumple con la propiedades conmutativas y asociativas, pero satisface que $x \oplus_c 0 = 0 \oplus_c x = x$. Además, para cualquier $x, y \in \mathbb{D}_c^n$, se cumple que $(-x) \oplus_c x = x \oplus_c (-x) = 0$ y $(-x) \oplus_c (x) \oplus y = y$ (inverso por la izquierda). Con esto es posible definir la substracción de Möbius en la ecuación (4.16).

$$gyr : \mathbb{D}_c^n \times \mathbb{D}_c^n \to Aut(\mathbb{D}_c^n, \oplus_c)$$
$$gyr[a, b]z = \ominus_c(a \oplus_c b) \oplus_c (a \oplus_c (b \oplus_c z))) \tag{4.15}$$

$$x \ominus_c y := x \oplus_c (-y). \tag{4.16}$$

Definición 4.3.5. (Espacio girovectorial) Sea la tupla $(V, \oplus, \otimes)$, donde V es un conjunto de elementos no vació denominados vectores, $\oplus$ es una operación binaria sobre V y $\otimes$ una operación externa sobre un cuerpo $(K, +, \cdot)$. Se dice que V es un espacio girovectorial si y solo si $(V, \oplus)$ es un girogrupo giroconmutativo, y la operación externa multiplicación por escalar $\otimes : K \times V \to V$ cumple las propiedades de producto por escalar.

En la descripción del espacio girovectorial en la bola de Poincaré ya se tiene el girogrupo de Möbius, sin embargo, se necesita dotarlo de una operación producto por escalar. Se define la operación externa **multiplicación escalar de Möbius**, el cual opera sobre el cuerpo de los reales $\mathbb{R}$. Sea $r \in \mathbb{R}$ y $x \in \mathbb{D}_c^n$, entonces la multiplicación escalar de Möbius $\otimes_c$ corresponde a:

$$\begin{gathered} \otimes_c : \mathbb{R} \times \mathbb{D}_c^n \to \mathbb{D}_c^n \\ r \otimes_c x := exp_0^c\left(r \cdot log_0^c(x)\right) = \frac{x}{\|x\|\sqrt{c}} tanh(r \cdot tanh^{-1}(\sqrt{c}\|x\|)) \end{gathered} \tag{4.17}$$

Finalmente, el espacio girovectorial de Möbius corresponde a la tupla $(\mathbb{D}_c^n, \oplus_c, \otimes_c)$ sobre el cuerpo $\mathbb{R}$. Por ende, ya es posible aplicar conceptos de álgebra lineal en la bola de Poincaré, requisito para definir capas neuronales hiperbólicas.

4.4. Redes Neuronales Hiperbólicas

Los espacios hiperbólicos han ganado impulso en el contexto de aprendizaje de máquinas, gracias a su capacidad de representación de datos con estructura jerárquica. Ganea et al. (2018) introdujo el concepto de capas neuronales hiperbólicas, estas consisten en capas neuronales que trabajan dentro del espacio girovectorial de Möbius. El cual, es necesario proveerlo de un operador de funciones conocido como *versión de Möbius*. Las capas hiperbólicas aprenden a generar representaciones incrustadas en la bola de Poincaré, es decir, representaciones vectoriales con jerarquía intrínseca.

4.4.1. Capas feedforward hiperbólicas

Las capas *feedforward* contiene los parámetros W y b, los pesos y el sesgo respectivamente, además de una función no lineal $f(\cdot)$ como activación. Con esto, la capa feedforward realiza la transformación en la entrada expresada en la ecuación (2.1), que consiste en una transformación afín y posteriormente la aplicación de la función no lineal. La versión hiperbólica de la capa *feedforward* debe realizar un símil de la transformación de su contraparte euclídea en la bola de Poincaré. Para esto se define el operador versión de Möbius, que va desde el espacio de funciones sobre el espacio euclídeo al espacio de funciones sobre la bola. Este se define como:

Definición 4.4.1 (Versión de Möbius)**.** Para una función $f : \mathbb{R}^n \to \mathbb{R}^m$ y un vector $x \in \mathbb{D}_c^n$, se define como $f^{\otimes_c}$ la versión de Möbius de f como:

$$\begin{gathered} f^{\otimes_c} : \mathbb{D}_c^n \to \mathbb{D}_c^m \\ f^{\otimes_c}(x) := exp_0^c(f(log_0^c(x)) \end{gathered} \tag{4.18}$$

Note que esta definición se fundamenta de la multiplicación por escalar de Möbius (4.17), cuyo procedimiento es parecido. Primero el mapeo logarítmico lleva el vector, que originalmente está en la bola de Poincaré, al espacio euclídeo, luego aplica la función euclídea, para finalmente volver a la bola mediante el mapeo exponencial. Esto es posible debido a que todas las funciones quedan bien definidas, ya que el espacio tangencial de la bola de Poincaré es $\mathcal{T}_0\mathbb{D}_c^n = \mathbb{R}^n$. Cabe destacar que $\lim_{c\to 0} f^{\otimes_c} \to f$,

además cumple con la **propiedad del morfismo** $(f \circ g)^{\otimes_c} = f^{\otimes_c} \circ g^{\otimes_c}$, y la **preservación de la dirección** $f^{\otimes_c}(x)/\|f^{\otimes_c}(x)\| = f(x)/\|f(x)\|$.

La capa *feedforward* hiperbólica, en primera instancia realiza la multiplicación entre los pesos $M \in \mathbb{R}^{m\times n}$ y un vector de entrada $x \in \mathbb{R}^n$. Esta multiplicación corresponde a una transformación lineal $T : \mathbb{R}^n \to \mathbb{R}^m$, tal que $M(x) := M \cdot x$. Con el uso de la operación versión de Möbius, se define la multiplicación matriz vector de Möbius:

Definición 4.4.2 (multiplicación matriz vector de Möbius)**.** Se define $M^{\otimes_c} : \mathbb{D}^n \to \mathbb{D}^m$, tal que:

$$M^{\otimes_c}(x) = M \otimes_c x = \frac{Mx}{\|Mx\|\sqrt{c}} tanh\left(\frac{\|Mx\|}{\|x\|} tanh^{-1}(\sqrt{c}\|x\|)\right) \tag{4.19}$$

Note que, $M \otimes_c x = 0$ si $Mx = 0$, por otro lado si se tiene otra matriz $W \in \mathbb{R}^{k\times m}$, es posible demostrar que se cumple la propiedad de la **asociatividad del producto matricial**, es decir $(W \cdot M)\otimes_c x = W \otimes_c (M \otimes_c x)$, además para un $r \in \mathbb{R}$, se cumple la **asociatividad de matriz escalar**, es decir $(rM)\otimes_c x = r\otimes_c (M \otimes_c x)$.

Finalmente, la capa feedforward hiperbólica requiere realizar la suma del sesgo y aplicar la activación no lineal ϕ. Se utiliza la adición de Möbius para la suma del sesgo y para aplicar la función no lineal se utiliza la versión de Möbius de ϕ, es decir $\phi^{\otimes_c}$.

Definición 4.4.3 (Capa feedforward hiperbólica)**.** Se define la capa feedforward hiperbolica como una función $H : \mathbb{D}_c^n \to \mathbb{D}_c^m$, la cual está determinada por los parámetros $M \in \mathbb{R}^{m\times n}$ y $b \in \mathbb{R}^m$ y una función no lineal $\phi : \mathbb{R}^m \to \mathbb{R}^m$, que realiza la siguiente operación:

$$y = H(x) := \phi^{\otimes_c}\left(M \otimes_c x \oplus_c exp_0^c(b)\right) \tag{4.20}$$

5 | Redes Adversarias Generativas Hiperbólicas (HGAN)

El aprendizaje profundo consiste en crear de manera secuencial representaciones abstractas jerárquicas de los datos de entrada. En el caso del procesamiento de imágenes, los datos se representan como un tensor de 3D de píxeles: ancho, alto, y mapa de color (rojo, verde y azul). Este tensor presenta una baja jerarquía, en consecuencia, una representación euclídea es suficiente. Mientras este tensor se adentra por la red neuronal, la información representada por los mapas de características comienza a adquirir mayor nivel de abstracción, por ejemplo, presencia de ojos, patas, pelo, color. En etapas más profundas de la red, la información representada consiste en los elementos presentes en la imagen, tales como: perros, gatos, personas. Estas características se encuentran ya fuertemente relacionadas con el lenguaje y tiene que ver con representaciones simbólicas de los diversos elementos que componen la imagen. Es por ello, la capacidad de las redes neuronales de responder que objeto está presente en la imagen.

Por este motivo, a pesar de que las imágenes no presentan un comportamiento jerárquico, existe una jerarquía intrínseca en ellas, la cual se logra apreciar a mayores niveles de abstracción. Es por ello, que han existido recientes iniciativas y estudios para incorporar geometría hiperbólica en el procesamiento de imágenes, específicamente, en capas de abstracción donde esta jerarquía se aprecia. En este punto radica la hipótesis de la presente tesis, el cual consiste en: ***Es posible explotar la jerarquía intrínseca de las imágenes, para mejorar el proceso de generación a través del uso de capas neuronales hiperbólicas en la arquitectura GAN***. Esto abre un camino dentro de las arquitecturas GAN, que consiste en combinar espacios hiperbólicos y euclídeos.

En este capítulo se revisará la metodología empleada en la investigación, además de las diferentes arquitecturas GAN estudiadas. En primer lugar, se profundizará sobre la jerarquía de las imágenes, luego se revisará la versión hiperbólica de arquitecturas GAN simples, que corresponden a GAN, WGAN-GP y CGAN. Finalmente se propone utilizar capas hiperbólicas en la arquitectura StyleGAN2, que es una arquitectura GAN moderna, la cual es el estado del arte en varios conjuntos de datos que se utilizan como punto de referencia para modelos generativos.

5.1. Jerarquía en las Imágenes

La estructura jerárquica se encuentra fuertemente en grafos y texto, he ahí los primeros acercamientos del uso de geometría hiperbólica. El uso de espacios hiperbólicos radica en algoritmos que crean incrustaciones vectoriales en espacios hiperbólicos, tanto para grafos (Nickel y Kiela (2017)) como para texto (Dhingra et al. (2018)). Además, existen acercamientos donde se procesa la información mediante capas neuronales en el espacio hiperbólico, tal como las capas neuronales *feedforward* hiperbólicas estudiadas en el capítulo anterior, las cuales se probaron en problemas de procesamiento natural de lenguaje, específicamente en tareas de reconocimiento de prefijos ruidosos y vinculación de textos. Además, existen capas hiperbólicas especializadas para trabajar con grafos, como, las capas propuestas por Liu et al. (2019) y Chami et al. (2019), que son adaptaciones de capas neuronales de grafos e introducen procesamiento en el espacio hiperbólico, mejorando tareas de predicción de nuevos enlaces o clasificación de grafos.

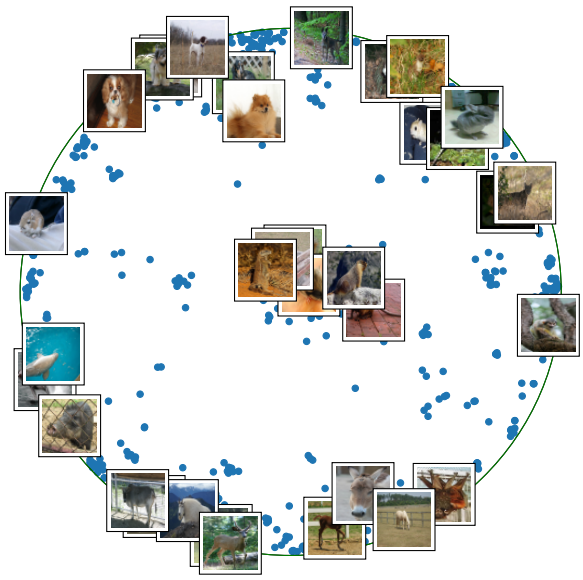
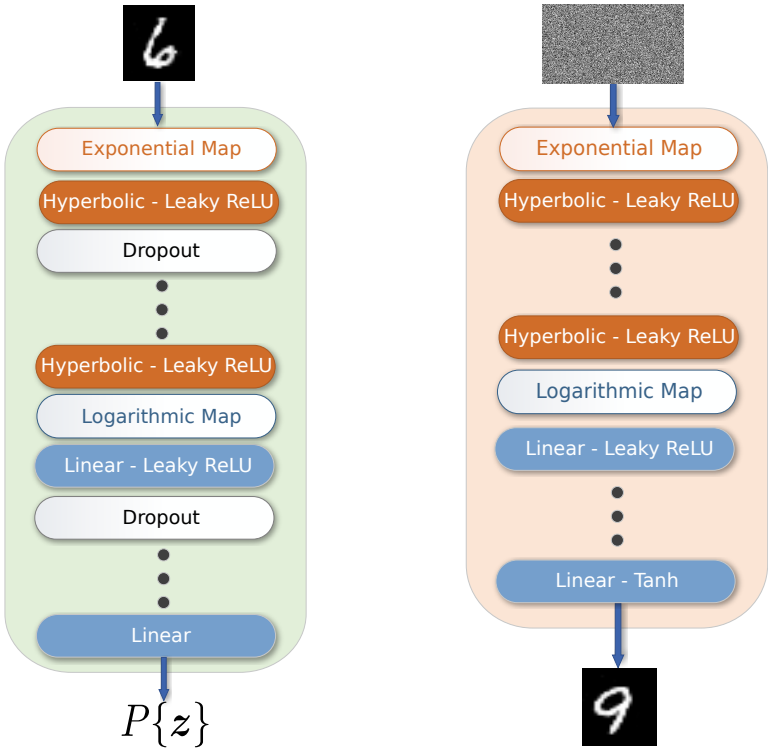


Figura 5.1: Visualización de la jerarquia presenta en la rama de los mamiferos de WordNet, enlazada con la respectiva imagen de ImageNet. Se implementa el algoritmo de incrustación de grafos en la esfera de Poincaré de dos dimensiones, para un $c = 1$ y 50 épocas.

Si bien, existen muchos estudios respecto a la utilización de la geometría hiperbólica para diversos tareas y tipos de datos, para las imágenes aún se encuentra en un estado reciente. Esto se debe a que la jerarquía en las imágenes no aparece de forma directa. Sin embargo, existen diversos indicios donde se intuye la existencia de jerarquía en las imágenes. El principal argumento empleado en esta tesis, consiste en que es posible agrupar diferentes imágenes según la similitud entre los elementos presentes. Estos elementos pueden ser agrupados dentro de otras categorías aumentando el nivel de abstracción. Si se realiza esta tarea sucesivamente, los elementos se organizan de forma natural en una estructura de árbol. Suponga el caso del reino animal, existen características físicas apreciables a la vista las cuales son propias de la evolución. Por ejemplo, resulta relativamente fácil diferenciar un animal que pertenece a la rama de los mamíferos de uno de la rama de los reptiles, debido a que los mamíferos comparten características similares, como, el pelaje. Dentro de la rama de los mamíferos, es posible seguir agrupando a los animales por sus características visuales, por ejemplo, los canes se constituyen por perros, lobos y hienas, a diferencia de los felinos como los gatos, leones y tigres. Por ende, se aprecia una relación entre la taxonomía de los animales y el lenguaje, el cual es posible ilustrar con el famoso conjunto de datos ImageNet (Deng et al. (2009)), que corresponde a un conjunto de imágenes ordenados según el conjunto de datos WordNet (Miller (1998)), donde este último es una red de palabras que se organiza en forma de árbol. Por ende, ImageNet es una agrupación jerárquica de las imágenes, y cabe destacar que en su página web existen herramientas para buscar imágenes navegando a través de las ramas del árbol. En la figura 5.1, se visualiza la jerarquía en las imágenes de ImageNet, donde se implementa el algoritmo de incrustación de grafos en la bola de Poincaré, propuesto por Nickel y Kiela (2017), sobre la rama de los mamíferos del conjunto de datos WordNet, para posteriormente enlazar cada palabra con su respectiva imagen obtenida de ImageNet.

Por lo tanto, las imágenes presentan una estructura jerárquica, gracias a la relación entre los elementos existentes y la palabra que los simboliza. Este argumento, junto con otros dos más, son propuestos por Khrulkov et al. (2020). El primer argumento consiste en la estructura jerárquica que presenta la tarea de recuperación de imágenes. Se tiene una fotografía global, que se constituye de múltiples fotografías en primer plano de distintos detalles. Existe una jerarquía ya que las fotografías globales se encuentran cerca a la raíz, en cambio las fotografías de detalles específicos corresponden a las hojas del árbol. El segundo argumento se aprecia en algunas tareas, donde imágenes más genéricas son las que contienen menos información, por ende, son más ambiguas. Por ejemplo, en reconocimiento de rostros, imágenes de rostros borrosos o de menor resolución (fotografía tomada desde lejos), se pueden relacionar con rostros de distintas personas que



(a) Discriminador **(b)** Generador

Figura 5.2: Ejemplo que muestra las arquitecturas de la HGAN, discriminador y generador y discriminador con capas hiperbólicas y euclídeas

estén en alta resolución. Por ello, imágenes de rostros borrosas se encuentran cerca de la raíz del árbol, y las de mayor resolución, tal que es posible identificar fácilmente a la persona, estarán cerca de las hojas. El trabajo realizado por Khrulkov et al. (2020), es la principal inspiración para la presente investigación, ya que utilizaron operaciones y las capas definidas en el capítulo anterior para realizar una incrustación jerárquica de las imágenes en la bola de Poincaré. Este acercamiento mejora tareas de aprendizaje profundo con pocas muestras.

5.2. GAN, CGAN y WGAN-GP, Hiperbólicas

El supuesto central de la presente tesis radica en que las redes neuronales hiperbólicas pueden mejorar el desempeño de la arquitectura GAN, ya sea en el proceso de generación o discriminación, debido a la capacidad de las capas neuronales hiperbólicas para explotar las características jerárquicas de las imágenes. La versión hiperbólica de la GAN (HGAN), consiste en combinar el espacio hiperbólico y euclídeo tanto en el generador como en el discriminador, reemplazando algunas de las capas *feedforward* normales o euclídeas por capas *feedforward* hiperbólicas. Para combinar ambos espacios, es necesario utilizar capas que implementen los mapeos exponenciales y logaritmos, y así mover los vectores resultantes desde el espacio euclídeo al hiperbólico y viceversa. El uso de diferentes espacios en las redes neuronales, permite que se creen mapas de características abstractas que representen diferentes estructuras implícitas en los datos, como la estructura jerárquica ya mencionada. Este enfoque agrega más grado de libertad al diseño de las arquitecturas de la GAN, en este caso la HGAN se define por la cantidad de capas hiperbólicas, la ubicación de estas y la curvatura del espacio hiperbólico (el valor de c de la bola de Poincaré). La HGAN es una familia de arquitecturas derivadas de la modificación de la arquitectura GAN originalmente propuesta por Goodfellow et al. (2014), la figura 5.2 muestra la arquitectura general qué se implementó. De manera similar a la HGAN,

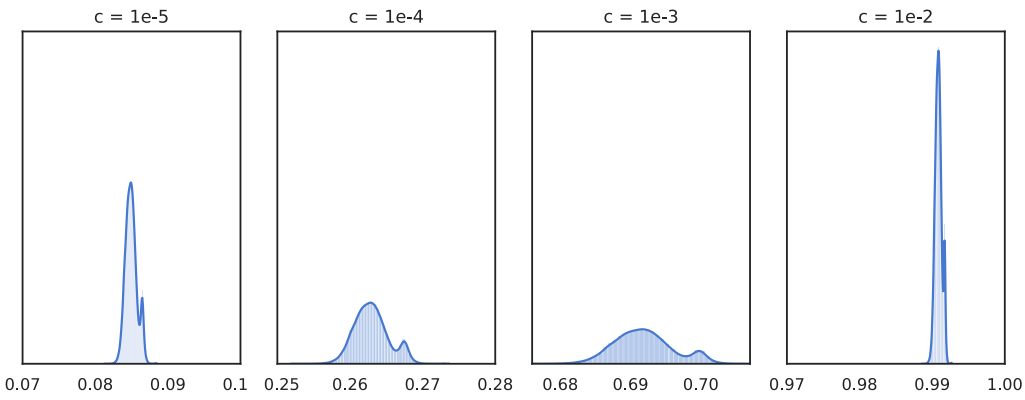


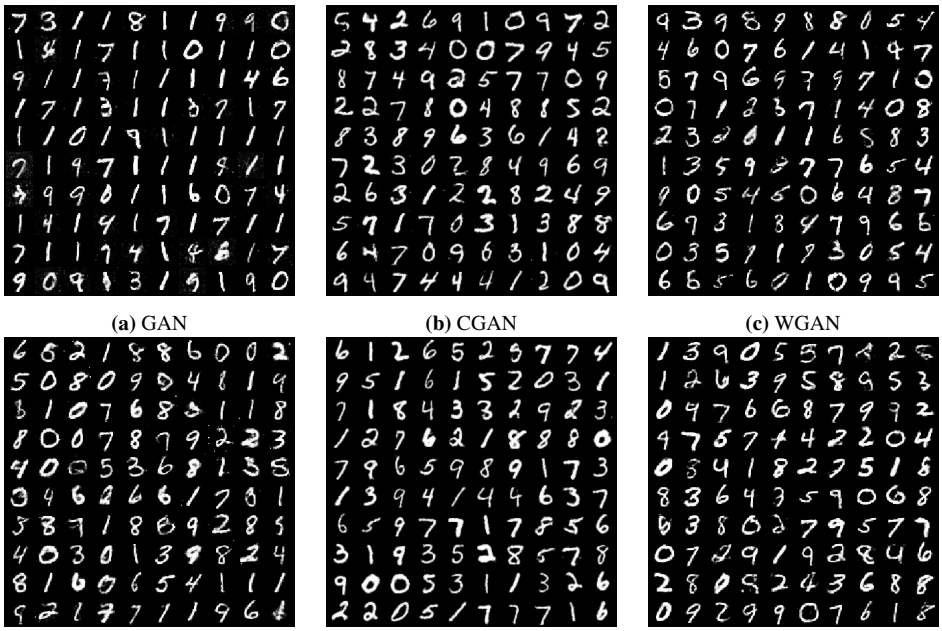
Figura 5.3: Distribución de la norma de los vectores del conjunto de datos MNIST, mapeados a la bola de Poincaré normalizado con respecto al radio de la bola r

las arquitecturas HWGAN y HCGAN, corresponden a modificaciones de las arquitecturas originales de la WGAN-GP Gulrajani et al. (2017) y la CGAN Mirza y Osindero (2014) respectivamente, al reemplazar capas euclídeas por capas hiperbólicas.

Las arquitecturas de la familia HGAN puede tener tanto capas euclídeas como hiperbólicas en diferentes disposiciones, con el correspondiente mapeo exponencial y logarítmico centrados en cero, necesarios para pasar entre los espacios. La notación de las diferentes configuraciones de la arquitectura corresponde a: $\mathcal{D}_{hhee}$, para el discriminador, y $\mathcal{G}_{hhee}$, para el generador, los subíndices indican de que capas están compuestas las redes neuronales. Por ejemplo, $\mathcal{G}_{hhee}$ significa que la red generador está comprendida por un mapa exponencial, dos capas hiperbólicas, un mapa logarítmico y dos capas euclidianas, el orden de lectura de izquierda a derecha corresponde al orden desde las capas de entrada hacia las de salida de la red. Para simplificar, tanto las capas de mapeo exponencial y logarítmico se omiten de la notación. Con esta notación, la arquitectura GAN original se representa como $\mathcal{D}_{eeee}\mathcal{G}_{eeee}$.

La ubicación de las capas hiperbólicas en la red tiene una influencia significativa en el rendimiento de la HGAN. Se probaron tres configuraciones generales y diferentes. Primero, La HGAN con configuración **euclídea-hiperbólica** (EH), que consta de capas euclidianas, un mapa exponencial y capas hiperbólicas al final de la red. La configuración EH genera primero un vector de características en el espacio euclídeo, y luego estas representaciones se procesan en la bola de Poincaré, generando representaciones abstractas jerárquicas. Se cree que las configuraciones EH pueden mejorar el rendimiento de la red discriminador, porque, la estructura jerárquica de la imagen es relevante en las capas que se encuentran a mayor profundidad en la red, debido a que tienen mayor grado de abstracción. La segunda configuración corresponde a la **hiperbólico-euclídeo** (HE), que consiste en un mapa exponencial, seguido por capas hiperbólicas, mapa logarítmico y capas euclídeas en la salida de la red. Esta secuencia, HE, podría mejorar el rendimiento creando representaciones jerárquicas, en primera instancia en el espacio hiperbólico que posteriormente ayudaría a las capas euclídeas a realizar la tarea deseada. Finalmente, se tienen las configuraciones **euclídea-hiperbólica-euclídea** (EHE) que consisten en redes que comienzan con capas euclidianas, seguidas por el mapeo exponencial, capas hiperbólicas, mapa logarítmico y capas euclídeas nuevamente hasta el final de la red. La configuración EHE primero crea una representación en el espacio euclídeo enriquecida con características abstractas, para luego explotarlas mediante el uso de las capas hiperbólicas, y por último, mapear la salida al espacio euclídeo y procesarla a través de capas euclídeas. El generador puede aprovechar esta configuración, ya que, su objetivo es crear una imagen completa a partir de ruido, el cual no posee una estructura jerárquica.

Otro grado de libertad es el valor de la curvatura o el factor c de la bola de Poincaré, el cual posee una influencia significativa en el rendimiento de la HGAN. El radio r de la bola de Poincaré se relaciona con c de la forma $c = 1/\sqrt{c}$. Por lo tanto, c representa cuan grande es la bola de Poincaré, y esto afecta en las funciones de mapeo y en las operaciones de las capas hiperbólicas. Por ejemplo, en la imagen 5.3 se



(a) GAN **(b)** CGAN **(c)** WGAN

(d) HGAN **(e)** HCGAN **(f)** HWGAN

Figura 5.4: Images of the best results generated by each architecture, on MNIST and measured by the FID.

muestra el efecto del mapeo exponencial sobre el conjunto de datos MNIST. La distribución corresponde a la magnitud de la imagen en el espacio hiperbólico normalizada por el radio de la bola, como se ve en la ecuación (5.1). Está distribución varía según el valor c, para $c = 10^{-5}$ ($r \approx 316$) la distribución es cercana a cero, en esta zona presenta un comportamiento similar al euclídeo, por ende, no se aprecian beneficios. A diferencia de la distribución cuando $c = 10^{-2}$ ($r = 10$), que colapsa al valor del radio r. Una mala elección del valor c causa un pobre comportamiento, además de provocar problemas de inestabilidad numérica.

$$R(x) = \| \exp_0^c(x) \| / r = \tanh(\frac{\|x\|}{r}) \tag{5.1}$$

Las arquitecturas HGAN utilizan activaciones *Leaky ReLU* en sus capas neuronales, tanto en las capas euclídeas como en las hiperbólicas, sin utilizar los mapeos exponencial y logarítmico antes y después de utilizar la función. Esto difiere de la implementación original propuesta por Ganea et al. (2018), en donde se utiliza la versión de Möbius para las funciones de activaciones. Además, se utiliza las capas *Dropout* para regulación, y el algoritmo de optimización es Adam Kingma y Ba (2017).

5.2.1. Experimentos

Los experimentos consisten en entrenar las diferentes arquitecturas HGAN con diferentes configuraciones sobre el conjunto de datos MNIST, el cual es un conjunto de imágenes de un canal, con números manuscritos. El rendimiento de cada arquitectura se mide mediante las métricas *Inception Score* (IS) y *Fréchet Inception Disntace* (FID).

La implementación utilizada para la GAN original consiste en cuatro capas *feedforward* en cada red neuronal. El discriminador tiene como entrada la imagen vectorizada de dimensión 784, seguida por capas cuyo tamaño son de 102, 512, 256, y 1 unidades (en potencias de dos para facilitar la implementación en GPU). Para el generador, la capa de entrada es de 128 unidades, el cual corresponde a ruido gaussiano sin correlación, y las capas ocultas y de salida son 256, 512, 1024 y 784 unidades respectivamente. El discriminador contiene capas de *Dropout* con una probabilidad tasa de *Dropout* de 0,1. Ambas redes tienen activaciones *leaky ReLU* con factor 0.2, a excepción de las capas de salida, donde el discriminador no utiliza

Architecture HGAN	c_d	c_g	FID ↓	IS ↑
$\mathcal{D}_{eeee}\mathcal{G}_{eeee}$	-	-	67.291	8.441
$\mathcal{D}_{heee}\mathcal{G}_{ehhe}$	1e-3	1e-3	28.320	8.394
$\mathcal{D}_{ehhh}\mathcal{G}_{eehe}$	1e-5	1e-3	**18.697**	**9.291**

Tabla 5.1: Mejores resultados para arquitectura HGAN, medidos con FID e IS.

Architecture HWGAN	c_d	c_g	FID ↓	IS ↑
$\mathcal{D}_{eeee}\mathcal{G}_{eeee}$	-	-	16.130	9.284
$\mathcal{D}_{eeee}\mathcal{G}_{hhhe}$	-	0.1	12.969	**9.424**
$\mathcal{D}_{eeee}\mathcal{G}_{ehhe}$	-	0.1	**12.884**	9.291

Tabla 5.2: Mejores resultados para arquitectura HWGAN, medidos con FID e IS.

Architecture HCGAN	c_d	c_g	FID ↓	IS ↑
$\mathcal{D}_{eeee}\mathcal{G}_{eeee}$	-	-	11.588	9.910
$\mathcal{D}_{hhhh}\mathcal{G}_{hhee}$	0.01	0.01	9.171	9.899
$\mathcal{D}_{hhee}\mathcal{G}_{hhee}$	0.01	0.01	**9.157**	9.903
$\mathcal{D}_{hhhe}\mathcal{G}_{hhee}$	1e-5	1e-5	10.940	**9.919**

Tabla 5.3: Mejores resultados para arquitectura HCGAN, medidos con FID e IS.

función de activación y el generador contiene *tanh*, para así forzar que la salida vaya de -1 a 1. Para la función de costo se utiliza entropía cruzada binaria con regresión logística y para entrenar se utiliza el optimizador Adam con tasa de aprendizaje igual a 0.0001, $\beta_1 = 0{,}5$ y $\beta_2 = 0{,}999$. La WGAN-GP se construye de la misma forma que la GAN, sin embargo, utiliza la perdida de Wasserstein (3.2) con penalización de gradiente, y tasa de aprendizaje igual a 0.0001, además de los mismos valores de las betas. Finalmente, la implementación de la CGAN posee la misma estructura que ambas arquitecturas, pero la capa de entrada del discriminador posee 794 unidades y del generador 138, que corresponde a 10 unidades más debido a que se concatena la información de las etiquetas de los números manuscritos (del 0 al 9). El optimizador ocupa los mismos parámetros que la implementación de la WGAN-GP, pero utiliza la entropía cruzada con regresión logística como función de perdida.

Las redes neuronales probadas son compuestas por bloques de redes hiperbólicas, tanto en el comienzo HE, medio EHE y final EH, para el generador y el discriminador con diferentes valores de c fijos. En primera instancia se utiliza la versión hiperbólica del discriminador, manteniendo el generador euclídeo, y luego lo contrario. Las configuraciones y los valores de c que obtienen un buen desempeño medido en IS y FID, se combinan en una arquitectura HGAN donde el generador y el discriminador son hiperbólicos, para luego ser entrenadas. Este procedimiento se realizó en las arquitecturas HGAN y HCGAN, pero no en la HWGAN, debido a que el discriminador nunca mostró un rendimiento mejor que la versión euclídea. Debido a que el discriminador y generador presentan naturaleza distinta, los valores de la curvatura no siempre son los mismos, c_d es la curvatura del discriminador y c_g para el generador. Esta distinción produce mejoras importantes en la medición del FID, desde 67.291 en la GAN euclídea, a 18.697 para la HGAN con $c_d = 10^{-5}$ y $c_g = 10^{-3}$, como se muestra en la tabla 5.1. Los mejores rendimientos se muestran en la tabla 5.1, 5.2 y 5.3, para las arquitecturas HGAN, HWGAN y HCGAN, respectivamente. Además, la imagen 5.4 se ilustra los resultados de las imágenes generadas, diferenciando las mejores configuraciones de cada arquitectura con su contra parte euclídea.

Se implementa un método para visualizar la mayoría de los experimentos por cada arquitectura: HGAN, HCGAN, y HWGAN. Además, la visualización muestra cada configuración y su respectivo valor de c. Las filas indican la configuración, por ejemplo $\mathcal{D}_{hhhh}\,\mathcal{G}_{eehh}$, y las columnas indican el valor del c. Las métricas IS y FID obtenidos se ilustran en el eje x, donde, para valores grandes de IS se obtiene un mejor rendimiento (dirección →), al contrario de la métrica FID, que resultaría en un valor bajo (dirección ←). Por lo tanto, para un modelo con buen desempeño, las marcas de FID e IS se ubican en el centro, cerca la una de la otra. En el caso contrario, las marcas se ubicarán lejos del centro y de entre ellas. Adicionalmente, el

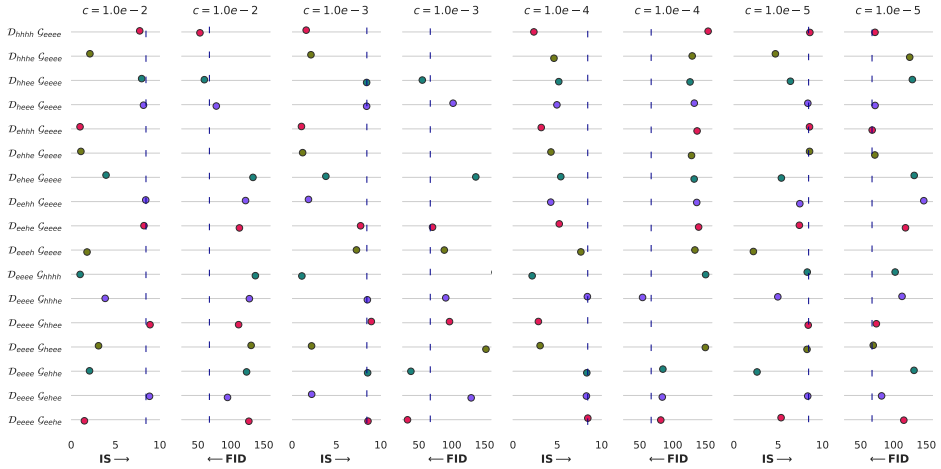


Figura 5.5: HGAN experiments visualization with only hyperbolic discriminator or hyperbolic generator measure with FID and IS. The segment line indicate the euclidean GAN performance.

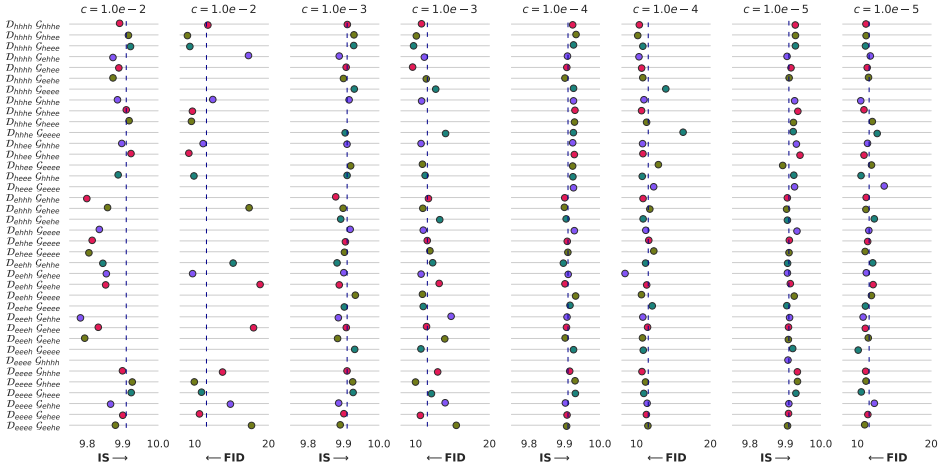


Figura 5.6: HCGAN experiments visualization with only hyperbolic discriminator or hyperbolic generator, and mixed hyperbolic configurations measure with FID and IS. The segment line indicate the euclidean CGAN performance.

rendimiento de la arquitectura euclídea se indica mediante la línea segmentada. Finalmente, si el eje x no presenta alguna marca, ya sea en el IS o el FID, significa un bajo rendimiento que esta fuera de los rangos del eje x.

Con los resultados de los experimentos, se aprecia que el uso de capas hiperbólicas mejora el desempeño considerablemente en la arquitectura GAN original. Además, también se observa mejoras en la arquitectura HWGAN y HCGAN. Por otro lado, en los gráficos 5.5, 5.7 y 5.6 se ilustra el efecto global de las capas hiperbólicas, además de la influencia del valor c. Cuando el radio de la bola es más pequeño comparado con la distribución de la norma de los datos 5.3, entonces él presenta un mayor comportamiento hiperbólico. Para estos valores del radio, se encuentran los resultados con mejor y peor desempeño, por ende, estos dependen de la ubicación de las capas hiperbólicas. Si el valor c baja en orden de magnitud, el radio crece por sobre la distribución de la norma de los datos, entonces el comportamiento de las capas es similar al euclídeo, provocando un rendimiento similar a su contraparte euclídea.

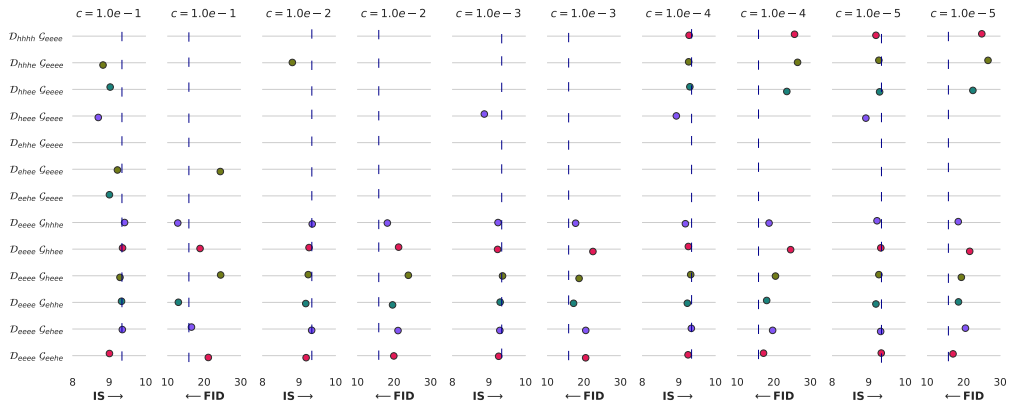


Figura 5.7: HWGAN experiments visualization with only hyperbolic discriminator or hyperbolic generator, and mixed hyperbolic configurations measure with FID and IS. The segment line indicate the euclidean WGAN performance.

5.3. StyleGAN2 con Red de Mapeo de Estilo Hiperbólico

En la arquitectura StyleGAN2, la red de mapeo de estilos f crea un espacio latente intermedio $w \in \mathcal{W}$, que representa las características de alto nivel en la imagen a generar. Estas características, al ser de alto nivel poseen mayor grado de jerarquía, la cual es posible explotar mediante la introducción de espacios hiperbólicos. La modificación consiste en reemplazar algunas capas *feedforward* ecualizadas que constituyen la red de mapeo por capas *feedforward* hiperbólicas ecualizadas en diferentes arreglos o configuraciones. Se sigue las configuraciones antes propuestas, las cuales corresponden a las configuraciones HE, EH y EHE, para diferentes valores de c. Se introduce la notación $f_{heeeeeee}$, a fin de identificar las distintas configuraciones de la red de mapeo, donde los subíndices indican las capas utilizadas con orden de lectura siguiendo la capa de entrada a la de salida. Además, se implementa una nueva capa neuronal, que consiste en una capa *feedforward* con su respectiva función de activación, pero se adhiere el mapeo exponencial al final de la capa. Esta capa se denota como m para expandir la notación anterior. y su función es restringir la norma del vector de salida a el radio de la bola de Poincaré, véase ecuación (5.1).

Se experimenta con el conjunto de datos CIFAR-10, que corresponde a un conjunto de imágenes a color de 32×32×3, donde los elementos estan constituidos por 10 clases: perros, gatos, caballos, aviones, autos, ciervos, ranas, barcos y camiones. Es decir, son imágenes pequeñas y con mucha varianza, lo que dificulta a los modelos de aprendizaje profundo. El experimento se implementa con un *batch size* de 32 imágenes, tasa de aprendizaje de 0.002, y regularización R1 con factor $\gamma = 0{,}1$, que se aplica cada 16 iteraciones. Por otro lado, no se utiliza mezcla de estilos ni regulación por largo de ruta. Se entrena con un total de 200k de iteraciones, y el resultado final se mide con la métrica FID de 50k imágenes de muestra generadas. Los resultados obtenidos se muestran en la tabla 5.4, que corresponde al promedio de 10 mediciones FID con muestras generadas con diferentes semillas. Por otro lado, las imágenes generadas de la mejor configuración se ilustran en la figura 5.8. Se observa una mejora en el desempeño de la StyleGAN2 con mapeo hiperbólico con respecto a con mapeo completamente euclídeo. Donde, la configuración con mejor desempeño corresponde a la que posee dos capas hiperbólicas al comienzo con un $c = 10^2$, esto es un radio de $r = 10$. El ruido es de dimensión de 512 y su norma promedio es de 22, es decir presenta un considerable comportamiento hiperbólico. Por otro lado, utilizar capas euclídeas con mapeo exponencial también mejoran el desempeño de la StyleGAN2, como se aprecia en la tabla 5.4.

Mapping Network	c	FID ↓
$f_{eeeeeeee}$	-	8.070
$f_{eeeeeeet}$	1e-4	7.583
$f_{ttttttt}$	1e-4	7.812
$f_{heeeeeee}$	1e-4	7.773
$f_{heeeeeee}$	1e-3	7.594
$f_{heeeeeee}$	1e-2	7.760
$f_{heeeeeee}$	1e-1	8.947
$f_{heeeeeee}$	1	10.611
$f_{hheeeeee}$	1e-3	7.933
$f_{hheeeeee}$	1e-2	**7.427**
$f_{hheeeeee}$	1e-1	9.097
$f_{hhheeeee}$	1e-3	8.203
$f_{hhhheeee}$	1e-3	8.288
$f_{eeeeeeeh}$	1e-3	7.988
$f_{eeeeeehh}$	1e-4	7.805
$f_{eeeehhhh}$	1e-3	7.957
$f_{eehhhhee}$	1e-3	9.116

Tabla 5.4: Resultados obtenido del promedio del FID de 50k de imágenes generadas, utilizando 10 semillas distintas por cada configuración probada.

(a) StyleGAN2 Euclídea **(b)** StyleGAN2 Hiperbólica

Figura 5.8: Muestra de 100 imágenes generadas a partir del modelo StyleGAN2 y la StyleGAN2 con mapeo hiperbólico que obtiene mejor FID promediado

5.4. Entrenamiento de la curvatura

El valor c de la Bola de Poincaré tiene un rango continuo de variación, desde cero a infinito. Encontrar el valor de c correcto, tal que, la HGAN presente un buen rendimiento sin que sea similar a la GAN euclídea es una tarea fundamental. Se introduce un procedimiento para realizar la búsqueda del valor c, de forma automática y en tiempo de entrenamiento. La metodología consiste en dejar c como un parámetro entrenable. Sin embargo, es necesario tomar ciertas consideraciones, ya que el valor c determina la forma y las operaciones del espacio girovectorial de Möbius, métricas y los mapas exponenciales y logarítmicos. Por tanto, el valor c debe ser positivo y distinto de cero, y además empíricamente, c no debe ser menor que $1e-7$ para asegurar estabilidad numérica. Por otro lado, para valores mayores que 1, la distribución colapsa en el borde provocando un pobre desempeño en la HGAN. Para integrar los requerimientos y restricciones del valor c, se decanta por ocupar un parámetro auxiliar entrenable δ, el cual entra a una función exponencial y así obtener el valor c real. Esta operación se muestra en la siguiente ecuación:

$$c = e^{\delta \cdot \alpha} + \epsilon. \tag{5.2}$$

Donde ϵ es una constante pequeña para no caer en problemas de estabilidad numérica (en la práctica es igual a 10^{-6}). La otra constante corresponde a α, que cumple la función de escalar la tasa de aprendizaje, ya que, en general el valor c debe tener una tasa de aprendizaje efectiva distinta a la de los otros parámetros de las capas neuronales. El parámetro δ se inicializa con la función inversa de la ecuación (5.2), para obtener el valor c inicial deseado.

Los experimentos consisten en entrenar la arquitectura HWGAN con generador hiperbolico, luego las mejores configuraciones de HGAN y HCGAN, y finalmente la StyleGAN2 con mapeo hiperbólico. Para la HWGAN, El algoritmo de entrenamiento tiene los mismos hiperparámetros que en los experimentos anteriores, pero se entrena por 800 épocas, para asegurar que el valor c se estabilice. Se implementa la ecuación (5.2), donde $\alpha = 1$, $\epsilon = 10^{-6}$ y el c inicial es igual a 1. Los resultados de los experimentos se muestran en la tabla 5.5 y 5.6, además de los gráficos de la evolución del valor c efectivo para la HWGAN. Por otro lado, se implementa en la StyleGAN2 con mapeo hiperbólico, con distintos valores de partida del c y constate α, donde los resultados se muestran en la tabla 5.7.

Architecture HWGAN	c_g final	c_g best epoch	FID ↓
$\mathcal{D}_{eeee}\mathcal{G}_{eehe}$	0.998	0.999	15.657
$\mathcal{D}_{eeee}\mathcal{G}_{ehhe}$	1.804e-3	2.156e-3	**12.433**
$\mathcal{D}_{eeee}\mathcal{G}_{ehee}$	1.467e-2	1.489e-2	12.987
$\mathcal{D}_{eeee}\mathcal{G}_{hhhe}$	8.654e-3	9.005e-3	12.503
$\mathcal{D}_{eeee}\mathcal{G}_{hhee}$	1.000	0.999	17.926
$\mathcal{D}_{eeee}\mathcal{G}_{heee}$	0.998	0.998	20.395

Tabla 5.5: Resultados de las diferentes arquitecturas HWGAN con generador hiperbólico, donde c es un parámetro entrenable y c incial igual a 1.

configuration	architecture	c_d	c_g	FID ↓
HGAN	$\mathcal{D}_{heee}\mathcal{G}_{ehhe}$	6.911e-3	3.903	54.948
HWGAN	$\mathcal{D}_{eeee}\mathcal{G}_{hhhe}$	-	9.005e-3	12.503
HWGAN	$\mathcal{D}_{eeee}\mathcal{G}_{ehhe}$	-	2.156e-3	**12.433**
HCGAN	$\mathcal{D}_{hhhe}\mathcal{G}_{hhee}$	0.999	0.999	10.422

Tabla 5.6: Mejores resultados de HGAN, HWGAN, y HCGAN con c entrenable.

configuration	c init	c final	FID ↓
$f_{heeeeeee}$	1	0.205	8.492
$f_{heeeeeee}$	0.1	8.523e-2	7.909
$f_{heeeeeee}$	0.01	4.264e-2	**7.761**
$f_{heeeeeee}$	0.001	4.120e-2	8.299
$f_{hheeeeee}$	1	3.666e-1	19.086
$f_{hheeeeee}$	0.01	6.560e-2	7.953

Tabla 5.7: StyleGAN2 con mapeo hiperbólico con c entrenable después de 200k de iteraciones de entrenamiento.

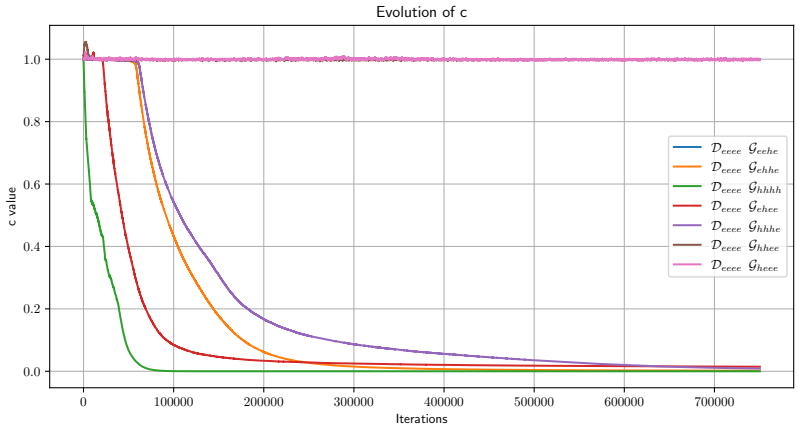


Figura 5.9: Gráfico de la evolución del valor c durante el entrenamiento del modelo.

Como se puede ver en los resultados obtenidos, la metodología para entrenar el c funciona, además resuelve los problemas de inestabilidad numérica de las metodologías anteriores. Sin embargo, es muy sensible a los hiperparámetro α y al c inicial. Por ello, es necesario encontrar una heurística que permitan guiar la elección de estos hiperparámetros.

6 | Conclusión

El trabajo expuesto muestra que las versiones hiperbólicas de las arquitecturas GAN's propuestas: HGAN, HCGAN, HWGAN, y StyleGAN2 con mapeo hiperbólico, pueden funcionar tan bien o, en algunos casos, mucho mejor que las arquitecturas euclidianas originales. De los múltiples experimentos presentados en la tesis se desprende que el rendimiento depende en gran medida de dos factores principales: la curvatura de la bola de Poincaré, a través del parámetro c, y la configuración de la arquitectura. Los mejores rendimientos se encontraron para HGAN con $c = 10^{-3}$, HCGAN $c = 10^{-2}$, HWGAN $c = 10^{-1}$ y StyleGAN2 con mapeo hiperbólico $c = 10^{-2}$, las cuales según la distribución de la norma de los datos se encuentran en una zona con características altamente hiperbólicas. Para valores pequeños de c, el crecimiento del radio provoca que las operaciones de las capas neuronales hiperbólicas se comporten como euclídeas, por lo que no hay mejora con respecto a la versión euclidiana. Para valores de c más bajos que 10^{-6} las redes no convergen debido a la inestabilidad numérica presente en las operaciones. Por otra parte, cabe destacar que es posible hacer una distinción entre los valores c usados en el discriminador con los del generador, los cuales pueden ser diferentes, que trae como consecuencia una mejora en el rendimiento de la HGAN.

Para las configuraciones, los experimentos muestran que las configuraciones EHE y HE para el generador tuvieron un mejor rendimiento. Para el discriminador, las configuraciones HE y EH funcionan mejor. Sin embargo, en la arquitectura HWGAN nunca se mostró una mejora del rendimiento con capas hiperbólicas en el discriminador, con respecto a la versión euclídea. Además, cuando el discriminador tiene una configuración EH, no se aplicó el mapa logarítmico en la salida de la red, ya que provoca inestabilidad numérica para dimensiones pequeñas.

Con respecto al entrenamiento de la curvatura o el valor c de la bola de Poincaré, los resultados son favorables, aunque todavía se necesita un mayor estudio. Si bien, la metodología propuesta mejora la estabilidad y el cumplimiento de las restricciones del valor c, aún es muy sensible a los hiperparámetros del algoritmo, donde, se obtienen buenos resultados para el entrenamiento en la arquitectura HWGAN y StyleGAN2.

Durante el desarrollo de este trabajo han surgido futuras propuestas, una de ellas es buscar la mejor configuración de la HGAN de manera automática, que se adecue al problema de generación. Esto evitaría tener que probar las distintas configuraciones para cada marco de trabajo. Por otro lado, existen otras capas hiperbólicas modernas, por ejemplo, la capa de atención hiperbólica. Futuras propuestas se fundamentarían en la arquitectura SAGAN, que es una GAN, pero con capas *Self Attention* o auto atención. Además, en investigaciones recientes se proponen capas hiperbólicas neuronales que sustituyen las capas neuronales hiperbólicas utilizadas en el desarrollo de esta tesis, estas se denominan HNN++, donde, mejoran algunos problemas de las HNN estándar, como por ejemplo la saturación al radio de la bola de Poincaré, además que introducen capas convolucionales hiperbólicas. Utilizar las capas HNN++ en lugar de las HNN estándar en la HGAN corresponde a un camino natural para seguir la misma línea de investigación.

Para finalizar, el acercamiento de la GAN hiperbólica abre un nuevo camino y familias de nuevas arquitecturas GAN. Demostrando la posibilidad de introducir espacios distintos a la euclídea, para así aprovechar las características geométricas presentes en los datos o en las representaciones internas, inclusive si el espacio utilizado no corresponde al espacio hiperbólico. Por ejemplo, el uso de redes esféricas, las cuales explotan propiedades cíclicas o de similitud entre los datos. También existe la opción de utilizar GAN's geométricas que aprendan la curvatura, en donde pueda mezclar capas euclídeas, hiperbólicas o esféricas, o

incluso de capas cuyas operaciones se definan en una variedad riemanniana. El uso de distintas geometrías en la arquitectura GAN permite mejorar la tarea de generación, ya que sus capas internas explotan las características intrínsecas de los datos o procesan datos que se encuentren en un espacio no euclídeo.

Bibliografía

Arjovsky, Martín y Bottou, Léon (2017). Towards principled methods for training generative adversarial networks. *ArXiv*, abs/1701.04862. 3.1, 3.3.2

Arjovsky, Martin; Chintala, Soumith; y Bottou, Léon (2017). Wasserstein generative adversarial networks. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70* (pp. 214–223). 3.3

Bengio, Y.; Courville, A.; y Vincent, P. (2013). Representation learning: A review and new perspectives. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(8), 1798–1828. 1

Bronstein, M. M.; Bruna, J.; LeCun, Y.; Szlam, A.; y Vandergheynst, P. (2017). Geometric deep learning: Going beyond euclidean data. *IEEE Signal Processing Magazine*, 34(4), 18–42. 1, 4

Chami, Ines; Ying, Zhitao; Ré, Christopher; y Leskovec, Jure (2019). Hyperbolic graph convolutional neural networks. In *Advances in neural information processing systems* (pp. 4868–4879). 1, 5.1

Dai, Shuyang; Gan, Zhe; Cheng, Yu; Tao, Chenyang; Carin, Lawrence; y Liu, Jingjing (2020). Apo-vae: Text generation in hyperbolic space. *arXiv preprint arXiv:2005.00054*. 1

Deng, Jia; Dong, Wei; Socher, Richard; Li, Li-Jia; Li, Kai; y Fei-Fei, Li (2009). Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition* (pp. 248–255).: Ieee. 2, 5.1

Dhingra, Bhuwan; Shallue, Christopher; Norouzi, Mohammad; Dai, Andrew; y Dahl, George (2018). Embedding text in hyperbolic spaces. In *Proceedings of the Twelfth Workshop on Graph-Based Methods for Natural Language Processing (TextGraphs-12)* (pp. 59–69). 1, 5.1

Ganea, Octavian; Becigneul, Gary; y Hofmann, Thomas (2018). Hyperbolic neural networks. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, y R. Garnett (Eds.), *Advances in Neural Information Processing Systems 31* (pp. 5345–5355). Curran Associates, Inc. 1, 4.4, 5.2

Goodfellow, Ian; Bengio, Yoshua; y Courville, Aaron (2015). *Deep Learning*. 2, 2.1.1

Goodfellow, Ian; Pouget-Abadie, Jean; Mirza, Mehdi; Xu, Bing; Warde-Farley, David; Ozair, Sherjil; Courville, Aaron; y Bengio, Yoshua (2014). Generative adversarial nets. In Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, y K. Q. Weinberger (Eds.), *Advances in Neural Information Processing Systems 27* (pp. 2672–2680). Curran Associates, Inc. 1, 3, 5.2

Gu, Albert; Sala, Frederic; Gunel, Beliz; y Ré, Christopher (2019). Learning mixed-curvature representations in product spaces. In *International Conference on Learning Representations*. 1, 4

Gulrajani, Ishaan; Ahmed, Faruk; Arjovsky, Martin; Dumoulin, Vincent; y Courville, Aaron C (2017). Improved training of wasserstein gans. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, y R. Garnett (Eds.), *Advances in Neural Information Processing Systems 30* (pp. 5767–5777). Curran Associates, Inc. 1, 3.3.3, 5.2

H., Farkas y I., Kra (1996). *Riemann Surfaces*. Springer. 1

He, Kaiming; Zhang, Xiangyu; Ren, Shaoqing; y Sun, Jian (2016). Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 770–778). 3.4.2

Heusel, Martin; Ramsauer, Hubert; Unterthiner, Thomas; Nessler, Bernhard; y Hochreiter, Sepp (2017). Gans trained by a two time-scale update rule converge to a local nash equilibrium. In *Advances in neural information processing systems* (pp. 6626–6637). 3.5.2

Huang, Xun y Belongie, Serge (2017). Arbitrary style transfer in real-time with adaptive instance normalization. In *Proceedings of the IEEE International Conference on Computer Vision* (pp. 1501–1510). 3.4.1

Karras, Tero; Aila, Timo; Laine, Samuli; y Lehtinen, Jaakko (2017). Progressive growing of gans for improved quality, stability, and variation. *arXiv preprint arXiv:1710.10196*. 3.4.1

Karras, Tero; Laine, Samuli; y Aila, Timo (2019). A style-based generator architecture for generative adversarial networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 3.4.1, 3.5.3

Karras, Tero; Laine, Samuli; Aittala, Miika; Hellsten, Janne; Lehtinen, Jaakko; y Aila, Timo (2020). Analyzing and improving the image quality of stylegan. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition* (pp. 8110–8119). 1, 3.4.2

Khrulkov, Valentin; Mirvakhabova, Leyla; Ustinova, Evgeniya; Oseledets, Ivan; y Lempitsky, Victor (2020). Hyperbolic image embeddings. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 1, 5.1

Kingma, Diederick P. y Ba, Jimmy Lei (2017). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980v9*. 2, 2.3.3, 5.2

Krioukov, Dmitri; Papadopoulos, Fragkiskos; Kitsak, Maksim; Vahdat, Amin; y Boguná, Marián (2010). Hyperbolic geometry of complex networks. *Physical Review E*, 82(3), 036106. 1, 4.2

Liu, Qi; Nickel, Maximilian; y Kiela, Douwe (2019). Hyperbolic graph neural networks. In *Advances in Neural Information Processing Systems* (pp. 8230–8241). 5.1

Mathieu, Emile; Le Lan, Charline; Maddison, Chris J; Tomioka, Ryota; y Teh, Yee Whye (2019). Continuous hierarchical representations with poincaré variational auto-encoders. In *Advances in neural information processing systems* (pp. 12565–12576). 1

Mescheder, Lars; Geiger, Andreas; y Nowozin, Sebastian (2018). Which training methods for gans do actually converge? In *International conference on machine learning* (pp. 3481–3490).: PMLR. 3.4.1

Miller, George A (1998). *WordNet: An electronic lexical database*. MIT press. 1, 5.1

Mirza, Mehdi y Osindero, Simon (2014). Conditional generative adversarial nets. *arXiv preprint arXiv:1411.1784*. 1, 3.2, 5.2

Nagano, Yoshihiro; Yamaguchi, Shoichiro; Fujita, Yasuhiro; y Koyama, Masanori (2019). A wrapped normal distribution on hyperbolic space for gradient-based learning. *arXiv preprint arXiv:1902.02992*. 1

Nazzal, Jamal; M. El-Emary, Ibrahim; y A. Najim, Salam (2008). Multilayer perceptron neural network (mlps) for analyzing the properties of jordan oil shale. *World Applied Sciences Journal*, 5. 2

Nickel, Maximillian y Kiela, Douwe (2017). Poincaré embeddings for learning hierarchicals representations. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, y R. Garnett (Eds.), *Advances in Neural Information Processing Systems 30* (pp. 6338–6347). Curran Associates, Inc. 1, 5.1

Pennington, Jeffrey; Socher, Richard; y Manning, Christopher D (2014). Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)* (pp. 1532–1543). 1

Salimans, Tim; Goodfellow, Ian; Zaremba, Wojciech; Cheung, Vicki; Radford, Alec; y Chen, Xi (2016a). Improved techniques for training gans. *Advances in neural information processing systems*, 29, 2234–2242. 3.5.1

Salimans, Tim; Goodfellow, Ian J.; Zaremba, Wojciech; Cheung, Vicki; Radford, Alec; y Chen, Xi (2016b). Improved techniques for training gans. *CoRR*, abs/1606.03498. 3.5.1

Samei, Zeynab y Jalili, Mahdi (2019). Application of hyperbolic geometry in link prediction of multiplex networks. *Scientific Reports*, 9(1). 1

Simonyan, Karen y Zisserman, Andrew (2014). Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*. 3.5.3

Tifrea, Alexandru; Bécigneul, Gary; y Ganea, Octavian-Eugen (2018). Poincaré glove: Hyperbolic word embeddings. In *International Conference on Learning Representations (ICLR 2019)*. 1

Ungar, Abraham Albert (2008). A gyrovector space approach to hyperbolic geometry. *Synthesis Lectures on Mathematics and Statistics*, 1(1), 1–194. 4.3.1, 4.3.1

Weisstein, Eric W (2002). *CRC concise encyclopedia of mathematics*. CRC press. 1, 4.1

Wilson, R. C.; Hancock, E. R.; Pekalska, E.; y Duin, R. P. W. (2014a). Spherical and hyperbolic embeddings of data. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 36(11), 2255–2269. 1

Wilson, R. C.; Hancock, E. R.; Pekalska, E.; y Duin, R. P. W. (2014b). Spherical and hyperbolic embeddings of data. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 36(11), 2255–2269. 1

Wolfe, Harold E (2012). *Introduction to non-Euclidean geometry*. Courier Corporation. 1

Zeiler, Matthew D y Fergus, Rob (2014). Visualizing and understanding convolutional networks. In *European conference on computer vision* (pp. 818–833).: Springer. 1

Zinkevich, Martin; Weimer, Markus; Smola, Alexander J; y Li, Lihong (2010). Parallelized stochastic gradient descent. In *NIPS*, volume 4 (pp.4̃).: Citeseer. 2