Universidad de Valladolid

Facultad de Ciencias

Trabajo Fin de Grado

Grado en Matemáticas

Redes Generativas Adversariales

Autor: Mª Elisabet Pelazas Rivero

Tutor/es: Eustasio del Barrio Tellado

Alberto González Sanz

Redes Generativas Adversariales (GANs)

Mª Elisabet Pelazas Rivero

Junio 2021

Índice general

Introducción

La noción de inteligencia artificial se podría considerar que surgió en 1950, momento en que el matemático Alan Turing se planteó la pregunta "¿las máquinas pueden pensar?" [75]. Además, en esta publicación propuso un test que actualmente se conoce como *test de Turing* para determinar si una máquina debe considerarse inteligente o no. Consiste en que un interrogador evalúa dos conversaciones, provenientes de una persona y de una computadora, pero sin saber cuál es cuál. Si no logra identificar qué diálogo es el que ha tenido con la computadora, entonces se considera que exhibe un comportamiento inteligente similar al de un ser humano o indistinguible de este.

Cualquiera podría adelantar que la posibilidad de que una máquina supere este test a día de hoy es realmente difícil que tenga lugar, pues todos hemos experimentado lo que es hablar con un asistente automático. Sin embargo, en otras tareas los ordenadores han logrado superar la capacidad humana. Es el caso de la *visión computacional* [70], que será el que trataremos en el presente trabajo por haber destacado las *Redes Generativas Adversariales (GANs)* en este ámbito.

Los algoritmos de *aprendizaje automático* (*machine learning*) proporcionan buenos resultados a la hora de reconocer patrones en los datos de entrada y utilizarlos para tareas de *clasificación* y *regresión*. Sin embargo, en lo que a la generación de nuevos datos se refiere, plantean numerosos problemas [44]. Lo mismo ocurre en *aprendizaje profundo* (*deep learning*), donde dichas tareas se llevan a cabo de forma, esencialmente similar, mediante una estructura característica denominada *red neuronal*. Es decir, un algoritmo es capaz de vencer en una partida de ajedrez a un ajedrecista experto, pero mantener una conversación fluida con *Alexa* o *Siri*, los asistentes de Amazon y Apple, respectivamente, es un sinsentido.

Hasta hace 10 años se podía afirmar que los métodos de aprendizaje basados en una aproximación *discriminativa* habían conseguido un nivel de éxito por encima de las capacidades humanas. No se podía decir lo mismo de problemas que tenían que abordarse con una aproximación *generativa*. No obstante, este panorama está cambiando desde 2014, cuando Ian Goodfellow, en aquel momento un estudiante de doctorado en la Universidad de Montreal, introdujo en [31] el concepto de las GANs . Son una clase de algoritmos de aprendizaje consistente en entrenar dos redes neuronales simultáneamente: un *generador*, cuya función es generar datos falsos, y un *discriminador*, que tratará de distinguir los falsos de los reales y clasificarlos convenientemente.

La palabra *redes* se refiere a la clase de algoritmos de aprendizaje que se utilizan habitualmente para representar el generador y el discriminador: redes neuronales. Dependiendo de la complejidad de la implementación de la GAN, estas pueden ser *redes neuronales prealimentadas (FFNNs)* o *redes neuronales convolucionales (CNNs)*, tal y como veremos.

En cuanto a la palabra *generativas*, esta indica su propósito: generar nuevos datos artificiales que parezcan naturales. Puede tratarse de voz, texto, imágenes, etc. Según el tipo de dato que se desee generar se deberá elegir un conjunto de entrenamiento u otro. Por ejemplo, si se trata de rostros, tal y como veremos e implementaremos finalmente, el objetivo será entrenar la GAN utilizando un conjunto de datos de multitud de distintas caras, de manera que se logre producir imágenes de caras que se asemejen todo lo posible a las reales.

Por último, el término *adversarial* hace referencia a la "competición" que tiene lugar entre el generador y el discriminador para lograr sus respectivos objetivos, que están continuamente tratando de superarse entre sí. Para intuir esto, en [31] se utiliza la siguiente metáfora: el criminal (el generador)

trata de falsificar moneda y el detective (el discriminador) pretende cazarlo. Cuanto más auténticas parezcan las monedas falsificadas, mejor estará rindiendo el detective en su tarea de detectarlas, y viceversa.

A pesar de no ser la primera técnica que permite generar datos, ha marcado la diferencia con respecto a cualquier otra gracias a la calidad de los resultados, especialmente, en la generación de imágenes [39]. Desde su invención, las GANs han atraído enormemente la atención tanto del público general como de académicos y de expertos en la industria, quienes han llegado a catalogarlas como "la mejor idea en el aprendizaje profundo en los últimos 20 años" (Yann LeCun, director de investigación en inteligencia artificial de Facebook, en [19]).

Como prueba de ello, destaquemos el ejemplo de la síntesis de rostros humanos, pues lo implementaremos en la parte final del manuscrito. Hasta la aparición de las GANs en 2014, lo mejor que se había conseguido generar mediante una máquina eran semblantes borrosos, lo cual se celebró como un gran éxito. Ya en 2017, como consecuencia de los avances de las GANs, se consiguió mejorar enormemente hasta el punto de resultar indistinguible, tal y como puede observarse en la figura 1 [8].

Figura 1: Evolución en la generación de caras. Fuente: [8].

El contexto en que nacieron, el de los *ataques adversariales*, también merece la pena mencionarlo por su estrecha relación con las GANs. Dichos ataques consisten en crear nuevas muestras que hagan fallar y cambiar las predicciones del algoritmo concreto que lleve a cabo una determinada red neuronal. Esto puede hacerse con un fin académico, de forma que se pongan en evidencia, para ser estudiados, los errores que las redes neuronales tienden a cometer; pero también, con un objetivo malicioso por parte de atacantes, lo cual podría suponer problemas en sistemas de reconocimiento facial, por ejemplo.

En el primero de los escenarios, se comprobó allá por los años 90 que el rendimiento de las redes neuronales es muy sensible a ligeras modificaciones, como consecuencia de la ingente cantidad de parámetros que intervienen, lo cual, ya de paso, nos puede concienciar de la dificultad de entrenar una GAN [67]. Por tanto, investigar en ataques adversariales es una de las mejores formas de entender cómo garantizar la precisión de las predicciones realizadas por las redes neuronales.

Los ejemplos adversariales, por tanto, nos proporcionan otra interpretación útil para entender las GANs: en vez de como una técnica cuyo objetivo es generar ejemplos realistas, como una técnica que trata de generar falsos ejemplos con los que engañar a un clasificador. Además, se ha demostrado que la utilización de GANs puede proveer de clasificadores que no sean tan sensibles a modificaciones [37], lo que brinda otro motivo para estudiarlas.

Respecto a los fundamentos matemáticos en que se sustentan el entrenamiento y las garantías de buen funcionamiento de las GANs, adelentaremos que son complejos, lo cual justifica principalmente el desarrollo del trabajo.

En general, los modelos generativos derivan de la bien conocida *estimación de máxima verosimilitud (EMV)*, puesto que para conseguir crear ejemplos realistas se debe conocer la densidad de probabilidad del conjunto de datos de entrada, y este método permite estimarla. Es el caso de los *Autoencoders Variacionales (VAEs)* [42].

Sin embargo, no ocurre lo mismo con todos los métodos generativos. El motivo es que para aplicar EMV debe existir un modelo paramétrico de densidades, lo cual probablemente, debido a la complejidad

de los datos con que se trabaja, no ocurra. Además, en caso contrario, su evaluación sería muy costosa. Por tanto, de manera alternativa, otros métodos generan ejemplos artificiales en sustitución de dicha densidad.

Tal y como trataremos de explicar en este trabajo, este es el caso de las GANs y lo que lleva a formular el problema que plantean de la siguiente manera: para cada parámetro fijo, por un lado, se trata de discriminar entre observaciones verdaderas y falsas con la mayor exactitud posible, de forma que esta máxima capacidad de discriminación sirva de medida de proximidad entre los datos. Por otro lado, se busca a la vez que dicha capacidad sea mínima con el fin de generar ejemplos artificiales indistinguibles de los reales. Es decir, se plantea este "doble juego" que constituye un *problema minimax*.

Con el objetivo de estudiar desde el punto de vista matemático las GANs, así como sus propiedades que garanticen un buen funcionamiento, nos gustaría, por tanto, tener una forma rigurosa de hacerlo. Las *divergencias y métricas estadísticas* juegan un papel fundamental en este aspecto, por lo que dedicaremos toda una sección a la exposición de algunas de las más importantes. En particular, concederemos especial importancia a la *distancia de Jensen-Shannon*, que proviene de la simetrización de la *divergencia de Kullback*, y en cuya minimización se basa, idealmente, el problema planteado por las GANs, tal y como se demuestra en [5]. Asimismo, cabe mencionar que se sigue investigando al respecto, habiéndose planteado recientemente la *métrica de Wasserstein* (definida en [76]) con el mismo objetivo [1].

Para una correcta comprensión y consecución de nuestro objetivo, por tanto, es imprescindible formular matemáticamente el problema que tratan de solventar las GANs. Veremos con más detalle que se tiene una *función objetivo* a optimizar que cuantificará la bondad en la predicción; el generador tratará de minimizarla y el discriminador, de maximizarla, de ahí la metáfora de la competición y el término "adversarial".

Asimismo, es de vital importancia la exposición de ciertos resultados fundamentales, por lo que, teniendo todo en cuenta, la organización del presente trabajo es la siguiente:

- En primer lugar, se estudian en el Capítulo 1 los conceptos previos generales que serán útiles posteriormente, como son las divergencias y métricas, y los fundamentos de aprendizaje automático desde el punto de vista de las Matemáticas.
- Al hilo de esto último, resulta fundamental explicar en el Capítulo 2 el problema que llevan a cabo las redes neuronales en su tarea de aprendizaje. Lo haremos para el caso más sencillo, las redes neuronales prealimentadas y pasaremos, después, a las redes neuronales convolucionales que son especialmente útiles en el tratamiento de imágenes.
- Tras este fundamento teórico, en el Capítulo 3 estaremos ya en condiciones de analizar las GANs, su relación con la divergencia de Jensen-Shannon y las buenas propiedades estadísticas que poseen.
- Finalmente, dedicamos el Capítulo 4 al estudio de algunas aplicaciones prácticas y aspectos computacionales de las GANs. En particular, a modo de ejemplo, nos propondremos generar dígitos numéricos y rostros humanos.

Capítulo 1

Resultados previos

Comencemos tratando cuestiones generales en Matemáticas que serán útiles a la hora de estudiar y entender los fundamentos de las GANs, así como obtener garantías teóricas sobre su buen funcionamiento.

Las *divergencias* y *métricas* en Estadística Matemática son de vital importancia en problemas clásicos de inferencia, ya que proporcionan una forma de medir la similitud entre dos distribuciones de probabilidad. Pero, además, han ganado importancia recientemente debido a que son la clave para analizar las garantías estadísticas de los métodos *aprendizaje automático* (*machine learning*) [10, 47], los cuales proporcionan reglas basadas en el análisis de una gran variedad de datos, cuya distribución resulta desconocida, con el fin de realizar predicciones precisas.

En particular, para modelos generativos (como son las GANs), se requiere medir la similitud entre la distribución de los datos generados y los datos reales de forma que se puedan optimizar los parámetros de dicho modelo. No obstante, debemos notar que una divergencia no es necesariamente una distancia, aunque es posible definir a partir de ellas otras que sí que lo cumplen, como la raíz de la *divergencia de Jensen-Shannon*, que nos resulta de especial interés en el desarrollo de los objetivos de este manuscrito. Se deriva de la simetrización de la *divergencia de Kullback-Leibler* la cual, a su vez, está íntimamente relacionada con el clásico método de *estimación de máxima verosimilitud (EMV)*.

En definitiva, en este capítulo, mediante dos extensas secciones, demostraremos cómo la teoría estadística clásica sienta las bases de los algoritmos más novedosos y útiles actualmente de aprendizaje automático, basado esencialmente, en minimizar una determinada función a partir de ciertas observaciones empíricas. Su buen rendimiento se asegura mediante un correcto ajuste de los parámetros que facilita una buena comprensión teórica, pues en muchos casos se trata de realizar prueba y error. Para ello también serán importantes estrategias de optimización como el algoritmo general de *Descenso de Gradiente*. Además, explicaremos con más detalle uno de los métodos que aprendizaje que existen por su similitud con el problema planteado en las GANs: la *regresión logística*.

Como principales referencias se empleará [74] para exponer, primeramente, los resultados relativos a divergencias y métricas, y [30, 35, 68] con el fin de detallar la teoría en que se fundamenta el aprendizaje automático.

1.1. Divergencias y métricas en Estadística Matemática

En primer lugar, nos centraremos en el estudio de algunas divergencias y métricas de especial importancia con el fin medir la similitud estadística entre ciertas distribuciones; esto es lo que se conoce como el problema estadístico de discriminación [7, 79]. Es importante aclarar que, en Estadística, dos distribuciones difieren más o menos según la dificultad que conlleve discriminar entre ellas con el mejor test [50].

Este hecho es de especial importancia en el presente trabajo porque las GANs plantean un cierto tipo de juego basado en discriminar distribuciones de probabilidad. La solución que estas redes plantean no puede entenderse a partir de métodos de estimación clásicos, como *máxima verosimilitud*. Con este

método estaríamos, asintóticamente, minimizando la *divergencia de Kullback*. Por el contrario, tal y como veremos en el Capítulo 3, las GANs minimizan, en esencia, la *distancia de Jensen-Shannon*. No obstante, todo ello está, en cierto modo, relacionado y es lo que trataremos también de explicar en esta sección.

La organización de la sección es la siguiente:

- Definición de la divergencia de Kullback-Leibler tal y como se enuncia en [74], añadiendo para facilitar su compresión, tanto su justificación a partir del *Teorema de Radon-Nykodim* A.1, como la íntima relación con el método de estimación de máxima verosimilitud. Se sentarán así las bases de la definición de la divergencia de Jensen-Shannon, motivada por el hecho de que la divergencia de Kullback no es una distancia.
- Definición de otras métricas importantes como: por un lado, las clásicas de *Hellinger* y *variación total* y, por otro, la distancia de Jensen-Shannon, que ha ganado importancia recientemente debido a su papel fundamental en el funcionamiento de las GANs y que se trata de la raíz de la divergencia de Jensen-Shannon, obtenida a partir de la simetrización de la divergencia de Kullback. Dado que nos resulta especialmente interesante en este trabajo, añadiremos la demostración de que, efectivamente, es una métrica generalizando los argumentos de [23]. Para un análisis más profundo al respecto de dichos conceptos recomendamos al lector [74].

1.1.1. Divergencia de Kullback-Leibler

En primer lugar, por tanto, se dará la definición de la divergencia de Kullback-Leibler, lo cual sentará las bases de la definición de la divergencia de Jensen-Shannon, que está motivada por el hecho de que la divergencia de Kullback no sea una distancia. Además, se estudiará la relación que guarda con la estimación de máxima verosimilitud.

La divergencia de Kullback-Leibler fue originalmente definida en 1951 [66] como una medida de la información basada en la diferencia existente entre distribuciones de probabilidad. Es, por tanto, un concepto central en Teoría de la Información y en Estadística, y juega un papel importante en el contexto del *aprendizaje automático*. Entre sus aplicaciones, cabe mencionar las que tiene en la clasificación de textos [46] o archivos multimedia [49].

En lo que sigue, trabajaremos en el espacio medible (Ω, σ) y tomaremos P y Q como dos medidas de probabilidad en dicho espacio. Para poder definir la divergencia de Kullback, puesto que esta se fundamenta en el Teorema de Radon-Nikdoym A.1, es necesario introducir ciertos conceptos cuya definición se pospone al apéndice A. Denotaremos por $P \ll Q$ a la condición de que P es *absolutamente continua* con respecto a Q, lo cual debe cumplirse para que exista la *derivada de Radon-Nikodym* de P con respecto a Q, que se denota por $\frac{dP}{dQ}$ y se deducirá en (1.9). Así, estamos ya en condiciones de enunciar la definición:

Definición 1.1. Sea (Ω, σ) un espacio medible. La **divergencia de Kullback-Leibler** se define como:

$$D_{KL}(P\|Q) = \begin{cases} \int_\Omega \frac{dP}{dQ} \log\left(\frac{dP}{dQ}\right) dQ & \text{si } P \ll Q \\ +\infty & \text{resto.} \end{cases} \tag{1.1}$$

Por tanto, es obvio que la divergencia de Kullback está bien definida en todo el espacio. En general, por simplicidad en la notación, usaremos

$$D_{KL}(P\|Q) = \int_\Omega \log\left(\frac{dP}{dQ}\right) dP. \tag{1.2}$$

Esta divergencia admite una formación alternativa, y equivalente, que mostraremos a continuación y será la que empleemos en el estudio del funcionamiento de las GANs. Es posible gracias a que siempre podemos encontrar una medida σ-finita, μ, en (Ω, σ) respecto de la cual P y Q sean absolutamente continuas. Por ejemplo, podemos elegir

$$\mu = \frac{P+Q}{2}, \tag{1.3}$$

que permite garantizar la existencia de las derivadas de Radon-Nikodym:

$$p = \frac{dP}{d\mu} \quad \text{y} \quad q = \frac{dQ}{d\mu}. \tag{1.4}$$

Entonces, puesto que $P \ll \mu$, por el Teorema de Radon-Nikodym A.1, se tiene que

$$P(A) = \int_A \frac{dP}{d\mu} d\mu = \int_A p d\mu, \quad \text{para todo A} \in \sigma. \tag{1.5}$$

De la misma forma,

$$Q(A) = \int_A \frac{dQ}{d\mu} d\mu = \int_A q d\mu, \quad \text{para todo A} \in \sigma. \tag{1.6}$$

Si, además, $P \ll Q$, entonces $q = 0$ implica $p = 0$, con lo que se reescribe (1.5):

$$P(A) = \int_A \frac{p}{q} q d\mu = \int_A \frac{p}{q} dQ, \quad \text{para todo A} \in \sigma. \tag{1.7}$$

Finalmente, utilizando las expresiones (1.4), se tiene que:

$$P(A) = \int_A \frac{dP}{dQ} dQ, \quad \text{para todo A} \in \sigma. \tag{1.8}$$

Por comparación de (1.7) y (1.8), deducimos que la derivada de Radon-Nikodym de P con respecto a Q es:

$$\frac{dP}{dQ} = \frac{p}{q}. \tag{1.9}$$

Entonces, podemos escribir la divergencia de Kullback y denotarla como se muestra a continuación:

$$D_{KL}(p\|q) = \int_\Omega p \log \frac{p}{q} d\mu. \tag{1.10}$$

En el caso particular de que P y Q tengan funciones de densidad f y g, respectivamente, con respecto a la medida de Lebesgue ([64], *Capítulo 2*), se puede probar la siguiente igualdad:

$$D_{KL}(P\|Q) = \int_{\mathbb{R}^k} f(x) \log \left(\frac{f(x)}{g(x)} \right) dx, \tag{1.11}$$

con una expresión similar si P y Q son discretas (cambiando integración por suma y densidad por función de masa de probabilidad).

Por último, demostremos algunas propiedades importantes. Puesto que a partir de ahora hablaremos de funciones convexas, seguiremos en todo el trabajo la definición de [64] (Definición 3.1).

Teorema 1.1. *Propiedades de la divergencia de Kullback-Leibler:*

1. $D_{KL}(P\|Q) \geq 0$.
2. $D_{KL}(P\|Q) = 0 \iff P = Q$.
3. $D_{KL}(P\|Q)$ *es convexa en* P.
4. *No depende de la elección de la medida* μ.
5. *No es una distancia.*

Demostración.

1. El caso infinito es trivial. Usemos la expresión de la divergencia (1.1). Consideremos la función $\varphi : \mathbb{R} \to \mathbb{R}$ definida por $\varphi(x) = x \log(x)$. Es una función estrictamente convexa, pues $\varphi''(x) = \frac{1}{x}$. Además, puesto que $\frac{dP}{dQ} \geq 0$ por ser una densidad y, en caso de ser ser cero, lo es en un conjunto de medida nula, se puede tomar $\varphi(\frac{dP}{dQ})$ y aplicar la desigualdad de Jensen ([64], Teorema 3.3):

$$D_{KL}(P\|Q) = \int_\Omega \varphi \left(\frac{dP}{dQ} \right) dQ \geq \varphi \left(\int_\Omega \frac{dP}{dQ} dQ \right) = \varphi(1) = 0. \tag{1.12}$$

2. Para lograr la igualdad en (1.12), como la función φ es estrictamente convexa, debe ocurrir que $\frac{dP}{dQ} = 1$. De acuerdo con (1.9) esto ocurre si, y solo si, $p \overset{c.s.}{=} q$, lo cual es equivalente a que $P = Q$.

3. Sean $P_1 \neq P_2$ dos probabilidades diferentes. La combinación lineal $\alpha P_1 + (1-\alpha)P_2$, $\alpha \in [0,1]$ sigue siendo una probabilidad. Veamos que:

$$D_{KL}(\alpha P_1 + (1-\alpha)P_2 \| Q) \leq \alpha D_{KL}(P_1\|Q) + (1-\alpha)D_{KL}(P_2\|Q).$$

Trabajemos con la expresión de la divergencia de Kullback (1.1). Sea φ definida como en la demostración de la anterior propiedad 1. Se tiene que φ es convexa, luego: $\varphi(tx+(1-t)y) \leq t\varphi(x)+(1-t)\varphi(y)$, $t \in [0,1]$. Puesto que $D_{KL}(P\|Q) = \int_\Omega \varphi(\frac{dP}{dQ})dQ$, podemos utilizar esto junto con la convexidad de φ y escribir:

$$\begin{aligned} D_{KL}(\alpha P_1 + (1-\alpha)P_2\|Q) &= \int_\Omega \varphi\left(\tfrac{d}{dQ}\left(\alpha P_1 + (1-\alpha)P_2\right)\right) dQ \\ &\leq \int_\Omega \left[\alpha\varphi\left(\tfrac{dP_1}{dQ}\right) + (1-\alpha)\varphi\left(\tfrac{dP_2}{dQ}\right)\right] dQ \\ &= \alpha D_{KL}(P_1\|Q) + (1-\alpha)D_{KL}(P_2\|Q), \end{aligned}$$

donde la última igualdad es consecuencia de la linealidad de la derivada y de la integral.

4. Consideremos $P \ll \lambda$ y $Q \ll \lambda$. A partir de la expresión (1.10), veamos que

$$\int_\Omega p \log \frac{p}{q} d\mu = \int_\Omega p \log \frac{p}{q} d\lambda.$$

La estrategia consiste en escribir ambas integrales con respecto a la medida $\lambda + \mu$. Puesto que $\mu \ll \lambda + \mu$, el Teorema de Radon-Nikodym A.1 implica la existencia de una función no negativa $d\mu/d(\lambda+\mu)$ tal que

$$\begin{aligned} &\int_\Omega p \log \frac{p}{q} d\mu \\ &= \int_\Omega p \log \frac{p}{q} \frac{d\mu}{d(\lambda+\mu)} d(\lambda+\mu) \\ &= \int_\Omega \frac{dP}{d\mu} \log \frac{p}{q} \frac{d\mu}{d(\lambda+\mu)} d(\lambda+\mu) \\ &= \int_\Omega \frac{dP}{d(\lambda+\mu)} \log \frac{p}{q} d(\lambda+\mu). \end{aligned}$$

La última igualdad se tiene $\lambda+\mu$-casi siempre y es consecuencia de la regla de la cadena.

5. Veamos un contraejemplo. Consideremos el espacio discreto $\Omega = \{0,1\}$ en el que definimos las probabilidades P, Q, R tales que sus funciones de masa de probabilidad p, q, r cumplen que:
$p(0) = 1/2$, $p(1) = 1/2$; $q(0) = 1/7$, $q(1) = 6/7$; $r(0) = 1/4$, $r(1) = 3/4$; con lo cual es obvio que P, Q, R son probabilidades.
Entonces, utilizando la expresión de la divergencia de Kullback (1.11) en el caso discreto, tenemos que:

$$\begin{aligned} D_{KL}(p\|q) &= \frac{1}{2}\log\frac{7}{2} + \frac{1}{2}\log\frac{7}{12} \approx 0.3569, \\ D_{KL}(p\|r) &= \frac{1}{2}\log 2 + \frac{1}{2}\log\frac{2}{3} \approx 0.1438, \\ D_{KL}(r\|q) &= \frac{1}{4}\log\frac{7}{4} + \frac{3}{4}\log\frac{7}{8} \approx 0.0398, \end{aligned}$$

con lo cual es evidente que

$$D_{KL}(p\|q) > D_{KL}(p\|r) + D_{KL}(r\|q),$$

incumpliendo la desigualdad triangular. Además, si calculamos

$$D_{KL}(q\|p) = \frac{1}{7}\log\frac{2}{7} + \frac{6}{7}\log\frac{12}{7} \approx 0.2830,$$

observamos que tampoco hay simetría. □

La divergencia de Kullback es, por tanto, una medida de la discrepancia entre P y Q, nula si $P = Q$ y positiva en cualquier otro caso. Veamos ahora que, asintóticamente, el clásico método de máxima verosimilitud busca minimizar esta discrepancia. De hecho, esta es la razón fundamental detrás del buen funcionamiento del estimador máximo verosímil.

1.1.1.1. Relación con el método de máxima verosimilitud

Se ha comenzado exponiendo la Definición 1.1 de la divergencia de Kullback. No obstante, cabe destacar que el método más clásico de estimación, máxima verosimilitud (ver, por ejemplo, [78], *Capítulo 9*), también contribuye a sentar las bases de esta definición. Expongámoslo a continuación de forma que, a la vez, encontremos la relación entre la divergencia de Kullback y la estimación de máxima verosimilitud.

Sean $X_1, \ldots, X_n$ vectores aleatorios independientes e igualmente distribuidas con ley desconocida P, donde P es un elemento de la familia paramétrica $\{P_\theta : \theta \in \Theta\}$. Es decir, $P = P_{\theta_0}$ para algún $\theta_0 \in \Theta$, donde $\Theta \subset \mathbb{R}^k$. El objetivo es encontrar un estimador de θ_0 mediante el método de máxima verosimilitud.

Comencemos suponiendo que la función de densidad de P es f. Por su parte, la de P_θ será $f(\cdot|\theta)$ y centrémonos en una parte de la muestra. La función de log-verosimilitud es, entonces:

$$K_n(\theta) = \frac{-1}{n}\sum_{i=1}^{n}\log f(x_i|\theta). \tag{1.13}$$

En este caso, el estimador máximo verosímil es:

$$EMV = \underset{\theta\in\Theta}{\operatorname{argmin}}\, K_n(\theta). \tag{1.14}$$

Sabemos por la Ley Fuerte de los Grandes Números que

$$K_n(\theta) \xrightarrow{c.s.} -\mathbb{E}_P \log f(x|\theta). \tag{1.15}$$

Sumamos el siguiente término a nuestra función de log-verosimilitud (1.13):

$$K_n(\theta) + \frac{-1}{n}\sum_{i=1}^{n}\log f(x_i). \tag{1.16}$$

Aplicando ahora la Ley Fuerte de los Grandes Números a (1.16) se tendrá el término que dará lugar a la definición de la divergencia de Kullback-Leibler.

$$K_n(\theta) + \frac{-1}{n}\sum_{i=1}^{n}\log f(x_i) \xrightarrow{c.s.} -\mathbb{E}_P \log f(x|\theta) + \mathbb{E}_P \log f(x). \tag{1.17}$$

Nótese que el segundo término de (1.17) se corresponde con la divergencia de Kullback en el caso particular de $\mathbb{R}^k$ que hemos visto en (1.11), puesto que:

$$D_{KL}(P\|P_\theta) = \mathbb{E}_P \log\left(\frac{f(x)}{f(x|\theta)}\right) = \int_{\mathbb{R}^k} f(x)\log\left(\frac{f(x)}{f(x|\theta)}\right)dx, \tag{1.18}$$

con una expresión similar si P y Q son discretas (cambiando integración por suma y densidad por función de masa de probabilidad).

Con esta discusión informal se puede entender que **el método de máxima verosimilitud tiende a minimizar la divergencia de Kullback** respecto a un modelo dado. Esta es la razón por la cual el método de máxima verosimilitud tiende a aproximar de manera precisa la función de densidad en ciertos

modelos paramétricos.

Sin embargo, la estimación clásica de máxima verosimilitud no es útil para estimar los parámetros en problemas tan complejos como los que tienen lugar en las redes neuronales. Además, **la divergencia de Kullback plantea el problema de no ser una distancia**. Estas dos razones motivan la exposición de métricas probabilísticas en la siguiente sección.

1.1.2. Métricas

El estudio de las discrepancias entre variables aleatorias requiere, en ciertas ocasiones, la utilización de distancias entre las mismas. Recordemos que una *distancia* o *métrica* es una aplicación $d : X \times X \longrightarrow \mathbb{R}$, donde X es un espacio topológico y tal que $d(x, y) \geq 0$ (la igualdad se da si, y solo si, $x = y$), es simétrica y cumple la desigualdad triangular.

Como hemos visto anteriormente, una divergencia estadística no es necesariamente una métrica. En particular, la divergencia de Kullback no satisface la desigualdad triangular, sin embargo su simetrización da lugar a la divergencia de Jensen-Shannon, cuya raíz sí que lo es, lo cual resulta de gran utilidad a la hora de resolver el problema de discriminación estadística que mencionábamos al inicio. Motivados por esta necesidad de comparar distribuciones, dedicaremos esta sección a la exposición de algunas métricas probabilísticas.

Más concretamente, describiremos la antes mencionada divergencia de Jensen-Shannon, la *distancia de Hellinger* y la de *variación total*. Seguidamente veremos que estas pueden ser controladas por la divergencia de Kullback a través de la famosa *desigualdad de Pinsker*. Haremos una mayor incidencia en la divergencia de Jensen-Shannon ya que para los objetivos del manuscrito (estudiar los beneficios y propiedades de las GANs) esta es la que posee un mayor interés.

1.1.2.1. Distancia de Hellinger y distancia de variación total

Recordemos que siempre que se tienen dos probabilidades P y Q en el espacio medible (Ω, σ), existe una medida σ-finita μ que domina a las dos como, por ejemplo (1.3), y tal que podemos definir al igual que hicimos en (1.4):

$$p = \frac{dP}{d\mu} \quad \text{y} \quad q = \frac{dQ}{d\mu}.$$

Teniendo esto en cuenta, es posible escribir las definiciones de la distancia de Hellinger y de variación total. Comencemos por la primera:

Definición 1.2. La ***distancia de Hellinger*** entre P y Q se define como:

$$d_H(P, Q) = \left(\frac{1}{2} \int_\Omega (\sqrt{p} - \sqrt{q})^2 d\mu \right)^{1/2}. \tag{1.19}$$

En la bibliografía (por ejemplo, [74]) se puede encontrar definida también sin el factor $1/2$, en cuyo caso se modifican trivialmente las propiedades que se exponen a continuación:

Teorema 1.2. *Propiedades de la distancia de Hellinger:*

1. $d_H(P, Q)$ *es una distancia.*
2. *La distancia de Hellinger no depende de la elección de la medida* μ.
3. $0 \leq d_H(P, Q) \leq 1$.
4. $d_H^2(P, Q) = 1 - \int_\Omega \sqrt{pq}\, d\mu$.

Demostración.

1. Dado que estamos tomando la distancia $\mathcal{L}^2$ entre $\sqrt{p}$ y $\sqrt{q}$ e identificando las probabilidades con su densidad respecto de μ, es evidente que es una métrica.

2. Consideremos $P \ll \lambda$ y $Q \ll \lambda$. Veamos que

$$\int_\Omega \left| \left(\frac{dP}{d\lambda}\right)^{1/2} - \left(\frac{dQ}{d\lambda}\right)^{1/2} \right|^2 d\lambda = \int_\Omega \left| \left(\frac{dP}{d\mu}\right)^{1/2} - \left(\frac{dQ}{d\mu}\right)^{1/2} \right|^2 d\mu.$$

La estrategia consiste en escribir ambas integrales con respecto a la medida $\lambda + \mu$. Puesto que $\mu \ll \lambda + \mu$, el Teorema de Radon-Nikodym A.1 implica la existencia de una función no negativa $d\mu/d(\lambda + \mu)$ tal que

$$\begin{aligned}
&\int_\Omega \left| \left(\frac{dP}{d\mu}\right)^{1/2} - \left(\frac{dQ}{d\mu}\right)^{1/2} \right|^2 d\mu \\
&= \int_\Omega \left| \left(\frac{dP}{d\mu}\right)^{1/2} - \left(\frac{dQ}{d\mu}\right)^{1/2} \right|^2 \frac{d\mu}{d(\lambda+\mu)} d(\lambda+\mu) \\
&= \int_\Omega \left| \left(\frac{dP}{d\mu}\frac{d\mu}{d(\lambda+\mu)}\right)^{1/2} - \left(\frac{dQ}{d\mu}\frac{d\mu}{d(\lambda+\mu)}\right)^{1/2} \right|^2 d(\lambda+\mu) \\
&= \int_\Omega \left| \left(\frac{dP}{d(\lambda+\mu)}\right)^{1/2} - \left(\frac{dQ}{d(\lambda+\mu)}\right)^{1/2} \right|^2 d(\lambda+\mu).
\end{aligned}$$

La última igualdad se tiene $\lambda + \mu$-casi siempre y es consecuencia de la regla de la cadena.

3. La primera desigualdad es trivial. Veamos la segunda:

$$d_H^2(P,Q) = \frac{1}{2}\int_\Omega (\sqrt{p} - \sqrt{q})^2 d\mu = \frac{1}{2}\int_\Omega (p + q - 2\sqrt{pq}) d\mu = 1 - \int_\Omega \sqrt{pq}\, d\mu \leq 1.$$

Es obvio que $d_H(P,Q) = 0$ si, y solo si, $P = Q$ y $d_H(P,Q) = 1$ si, y solo si, existe $A \in \sigma$ tal que $P(A) = 1$ y $Q(A) = 0$.

4. Se ha deducido en la prueba anterior. □

A continuación, pasemos a definir la distancia de variación total:

Definición 1.3. La ***distancia de variación total*** entre P y Q se define como:

$$d_{TV}(P,Q) = \frac{1}{2}\int_\Omega |p - q| d\mu. \tag{1.20}$$

Algunas propiedades de la distancia de variación total son:

Teorema 1.3. *Propiedades de la distancia de variación total:*

1. $d_{TV}(P,Q)$ *es una distancia.*
2. $d_{TV}(P,Q)$ *no depende de la elección de la medida* μ*. De hecho,*

$$d_{TV} = \sup_{A \in \sigma} |P(A) - Q(A)|. \tag{1.21}$$

3. $0 \leq d_{TV}(P,Q) \leq 1$.

Demostración.

1. Dado que estamos tomando la distancia $\mathcal{L}^1$ entre p y q e identificando las probabilidades con su densidad (a partir de μ), es evidente que es una métrica.

2. Puesto que $p = \frac{dP}{d\mu}$ y $q = \frac{dQ}{d\mu}$ son funciones medibles, se tiene que $B_0 = \{p \geq q\} \in \Omega$. Entonces,

$$P(A) - Q(A) = \int_A (p-q)d\mu = \int_{A\cap B_0} (p-q)d\mu + \int_{A\cap B_0^c} (p-q)d\mu.$$

Buscamos acotar esta diferencia por arriba y por abajo:

$$P(A) - Q(A) \leq \int_{A\cap B_0} (p-q)d\mu \leq \int_{B_0} (p-q)d\mu.$$

$$P(A) - Q(A) \geq \int_{A\cap B_0^c} (p-q)d\mu \geq \int_{B_0^c} (p-q)d\mu.$$

Pero, además:

$$\int_{B_0^c} (p-q)d\mu = -\int_{B_0} (p-q)d\mu,$$

puesto que

$$0 = \int_\Omega |p-q|d\mu = \int_{B_0} (p-q)d\mu + \int_{B_0^c} (p-q)d\mu.$$

Así, sea cual sea el conjunto A, llegamos a que:

$$|P(A) - Q(A)| \leq \int_{B_0} (p-q)d\mu = P(B_0) - Q(B_0),$$

con lo que:

$$\sup_{A\in\sigma} |P(A) - Q(A)| = P(B_0) - Q(B_0), \ \ \forall A \in \sigma.$$

Veamos, para acabar, que este superior coincide con la distancia de variación total. Usando lo que hemos deducido, tenemos que:

$$\int_\Omega |p-q|d\mu = 2\int_{B_0} (p-q)d\mu.$$

Luego,

$$P(B_0) - Q(B_0) = \frac{1}{2}\int_\Omega |p-q|d\mu.$$

3. Se deduce de (1.21), $d_{TV} = \sup_{A\in\sigma} |P(A) - Q(A)|$. □

De hecho, $d_{TV} = 0$ si, y solo si, $P = Q$ y $d_{TV} = 1$ si, y solo si, existe $A \in \sigma$ tal que $P(A) = 1$ y $Q(A) = 0$.

1.1.2.2. Distancia de Jensen-Shannon

La distancia de Jensen-Shannon ha ganado importancia últimamente por su importancia en la explicación teórica de las GANs. Se obtiene a partir de la raíz de la divergencia de Jensen-Shannon que, a su vez, proviene de la simetrización de la divergencia de Kullback, tal y como veremos a continuación.

Definición 1.4. La **divergencia de Jensen-Shannon** entre P y Q se define como

$$D_{JS}(P,Q) = \frac{1}{2}D_{KL}\left(P\middle\|\frac{P+Q}{2}\right) + \frac{1}{2}D_{KL}\left(Q\middle\|\frac{P+Q}{2}\right). \tag{1.22}$$

Esta forma de simetrizar la divergencia de Kullback es especialmente importante porque, como hemos adelantado, su raíz es una distancia. Antes de pasar a demostrarlo veamos algunas propiedades, recordando primeramente que siempre que se tienen dos probabilidades P y Q en el espacio medible (Ω, σ), existe una medida $\sigma-$finita μ que domina a las dos como, por ejemplo (1.3), y tal que podemos definir al igual que hicimos en (1.4):

$$p = \frac{dP}{d\mu} \quad \text{y} \quad q = \frac{dQ}{d\mu}.$$

Teorema 1.4. *Propiedades de la divergencia de Jensen-Shannon:*

1. $0 \leq D_{JS}(P, Q) \leq \log 2$.
2. $\delta = \sqrt{D_{JS}(P, Q)}$ *es una métrica y se conoce como* ***distancia de Jensen-Shannon****.*

Demostración.

1. La primera desigualdad es trivial a partir de la definición, puesto que ya sabemos que la divergencia de Kullback es mayor o igual que 0. Veamos la segunda, que se deduce inmediatamente a partir de lo siguiente:
$$D_{KL}\left(p, \frac{p+q}{2}\right) = \int_\Omega p \log \frac{2p}{p+q} d\mu \leq \int_\Omega p \log \frac{2p}{p} d\mu = \log 2.$$

2. Remarquemos, de nuevo, que esta propiedad es la razón por la que introducimos esta nueva nueva divergencia. Probemos que, efectivamente, la raíz de $D_{JS}(P, Q)$ es una métrica, para lo cual adaptaremos la demostración dada en [23]. Observemos que, en nuestro caso, es válida para cualquier tipo de probabilidad y no está limitada para soporte finito, como ocurre en [23].

   En lo que sigue, $\mathbb{R}^+$ incluye al cero. Sea $\mathcal{P}$ el conjunto de las distribuciones de probabilidad en Ω. Consideraremos la función $D_{JS} : \mathcal{P} \times \mathcal{P} \to \mathbb{R}^+$ definida por

   $$D_{JS}(p, q) = \int_\Omega \left(p \log \frac{2p}{p+q} + q \log \frac{2q}{p+q} \right) d\mu, \tag{1.23}$$

   que representa la divergencia de Jensen-Shannon teniendo en cuenta que la divergencia de Kullback se puede expresar como en (1.10). Esta definición no depende de la elección de la medida μ tal y como se demostró en el Teorema 1.1 y es, por tanto, correcta.

   Ya sabemos que $D_{KL}(P\|Q) = 0$ si, y solo si, $P = Q$ y estrictamente positiva en otro caso. Además, $D_{JS}(P, Q)$ es simétrica en P y Q y, por tanto, también lo es $\sqrt{D_{JS}(P, Q)}$. Entonces, solo falta probar la desigualdad triangular, para lo cual se introducen los siguientes resultados previos:

   Sea la función $L(p, q) : \mathbb{R}^+ \times \mathbb{R}^+ \to \mathbb{R}^+$ definida por

   $$L(p, q) = p \log \frac{2p}{p+q} + q \log \frac{2q}{p+q}. \tag{1.24}$$

   Esta función se puede tomar como el integrando de (1.23). Veamos que $L(p, q) \geq 0$ y que la igualdad se da solo para $p = q$, para lo cual, conviene escribirla como

   $$L(p, q) = \log \frac{2}{1 + \frac{q}{p}} + \frac{q}{p} \log \frac{2}{1 + \frac{1}{q/p}}, \tag{1.25}$$

   y tomar la función

   $$h(x) = \log \frac{2}{1+x} + x \log \frac{2}{1 + \frac{1}{x}}, \quad x = \frac{q}{p}. \tag{1.26}$$

   Está claro que $h(1) = 0$, es decir, se anula cuando $p = q$. Ademas, derivando obtenemos que

   $$h'(x) = \log \frac{2x}{1+x} = \log(2) - \log \frac{1}{1 + \frac{1}{x}}. \tag{1.27}$$

Por tanto, deducimos que $h'(x) > 0 \iff x > 1$ y $h'(x) < 0 \iff x < 1$, y la igualdad se da para $x = 1$. Con lo cual, la función presenta un mínimo en este punto cuya imagen es $h(1) = 0$ y, por tanto, se deduce lo que queríamos probar.

Nos interesa ahora ver que $\sqrt{L(p,q)}$ cumple al desigualdad triangular. Puesto que vamos a necesitar algunas propiedades de la derivada parcial de $L(p,q)$, introducimos la función $g : \mathbb{R}^+ \setminus \{1\} \to \mathbb{R}^+$ definida por

$$g(x) = \frac{\log \frac{2}{x+1}}{\sqrt{L(x,1)}}. \tag{1.28}$$

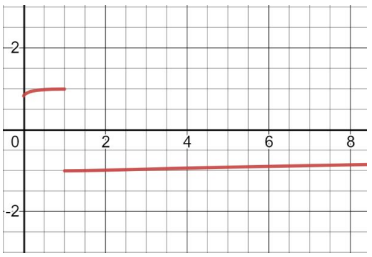

Figura 1.1: Función g definida en (1.28).

Lema 1.1. *Sea g definida como en (1.28). Entonces*

a) $\lim_{x \to \mp 1} g(x) = \pm 1$, *es decir, g presenta un salto finito de $+1$ a -1 en $x = 1$.*

b) $\frac{dg}{dx} > 0$, $\forall x \in \mathbb{R}^+ \setminus \{1\}$.

Como consecuencia de este lema, $|g(x)| \leq 1$ y la igualdad se da en $x = 1$. Además, es fácil ver que $|g|$ es continua, pero no g.

Demostración.

Nótese en primer lugar que g cambia de signo en $x = 1$. Una aplicación directa del teorema de L'Hôpital (tras derivar dos veces) nos da $\lim_{x \to 1} g^2(x) = 1$. Derivando, se obtiene que $\frac{dg}{dx}$ es positivo si, y solo si, $f < 0$ donde f viene dada por

$$f(x) = \log \tfrac{2}{1+x} + \log \tfrac{2x}{1+x}.$$

De nuevo, evaluando y derivando, obtenemos que $f(1) = f'(1) = 0$ y

$$f''(x) = \tfrac{-1}{x^2(1+x)} \left(\log \tfrac{2}{1+x} + x^2 \log \tfrac{2x}{1+x} \right).$$

Usando la desigualdad $\log a \geq 1 - \frac{1}{a}$, deducimos que $f'' < 0$, luego f es cóncava. Esto, combinado con lo anterior, nos lleva a que $f < 0$ para $x \neq 1$. □

Lema 1.2. *Sean $p, q, r \in \mathbb{R}^+$. Entonces $\sqrt{L(p,q)} \leq \sqrt{L(p,r)} + \sqrt{L(r,q)}$.*

Demostración.

El caso en que p, q, r son cero es trivial. Supongamos ahora que $p \leq q$ y denotemos $h(r) = \sqrt{L(p,r)} + \sqrt{L(r,q)}$. Veamos que $h(r)$ tiene dos mínimos, uno en $r = p$ y otro en $r = q$, y un solo máximo entre p y q.

Derivando,

$$\frac{\partial h(r)}{\partial r} = \frac{\log \frac{2r}{p+r}}{2\sqrt{L(p,r)}} + \frac{\log \frac{2r}{q+r}}{2\sqrt{L(q,r)}}.$$

Tomando g como en el Lema 1.1 y $x = \frac{p}{r}$ y $\beta x = \frac{q}{r}$ ($\beta > 1$), deducimos que

$$2\sqrt{r}\frac{\partial h(r)}{\partial r} = g(x) + g(\beta x).$$

El hecho de que $|g(x)| \leq 1$ y la igualdad se de solo en $x = 1$, junto con el hecho de que g da un salto finito de $+1$ a -1 en $x = 1$ (por lo demostrado en el Lema 1.1), implican que la derivada $\frac{\partial h(r)}{\partial r}$ presente un cambio de signo en $r = p$ porque, en ese caso, $x = 1$ y $|g(x)| > |g(\beta x)|$. Lo mismo sucede si $r = q$. Estos extremos son mínimos porque r es el recíproco de x.

Además, $\frac{dg(x)}{dx} \geq 0$. Por tanto, entre $x = \frac{1}{\beta}$ y $x = 1$, se tiene que $g(x) + g(\beta x)$ es monótona creciente y, en consecuencia, a lo sumo, presenta un cambio de signo. □

Estamos ya en condiciones de terminar de probar que la raíz de la divergencia de Jensen-Shannon es una métrica: aplicando la desigualdad de Minkowski a la raíz cuadrada de expresión (1.23), por el Lema 1.2, se tiene que se satisface la desigualdad triangular y podemos concluir. □

1.1.2.3. Desigualdades

Notemos que hemos acotado la distancia de Jensen-Shannon, la de Hellinger y la variación total, lo cual es importante, pues nos da una idea de cómo miden estas distancias. Además, a partir de la definición, es fácil observar que la d_{TV} tiene una interpretación más simple, seguida de la d_H. Aquí exploraremos algunas relaciones entre métricas, lo que nos permite valorar mejor el conocimiento, por ejemplo, de que la divergencia de Kullback entre P y Q esté por debajo de un determinado umbral.

Lema 1.3. *Sean P y Q dos medidas de probabilidad en el espacio medible (Ω, σ). Entonces, se cumple que:*

$$d_H(P,Q) \leq \sqrt{\frac{D_{KL}(P\|Q)}{2}}. \tag{1.29}$$

Demostración.

El caso $D_{KL}(P\|Q) = \infty$ es trivial. Supongamos que $D_{KL}(P\|Q) < \infty$ y, por tanto, según (1.1), $P \ll Q$. Puesto que $-\log(x+1) \geq -x$ si $x > -1$, tenemos

$$\begin{aligned} D_{KL}(P\|Q) &= \int_\Omega p\left(\log\frac{p}{q}\right) = 2\int_\Omega p\left(\log\sqrt{\frac{p}{q}}\right) \\ &= -2\int_\Omega p\log\left(\left[\sqrt{\frac{q}{p}} - 1\right] + 1\right) \\ &\geq -2\int_\Omega p\left[\sqrt{\frac{q}{p}} - 1\right] \\ &= -2\left(\int_\Omega \sqrt{pq} - 1\right) = 2d_H^2(P,Q). \end{aligned}$$

□

Lema 1.4. ***(Desigualdad de Pinsker)*** *Sean P y Q dos medidas de probabilidad en el espacio medible (Ω, σ). Entonces, se cumple que:*

$$d_{TV}(P,Q) \leq \sqrt{\frac{D_{KL}(P\|Q)}{2}}. \tag{1.30}$$

Demostración.

Consideremos la función

$$\varphi(x) = x\log x - x + 1, \;\; x \geq 0,$$

considerando $0\log 0 = 0$. Por tanto, notemos que $\varphi(0) = 1$, $\varphi(1) = 0$, $\varphi'(1) = 0$, $\varphi''(x) = 1/x \geq 0$, y $\varphi(x) \geq 0, \quad \forall x \geq 0$. Además,

$$\left(\frac{4}{3} + \frac{2}{3}x\right)\varphi(x) \geq (x-1)^2, \quad x \geq 0. \tag{1.31}$$

Esta desigualdad está clara para $x = 0$. Si $x > 0$, la función

$$g(x) = (x-1)^2 - \left(\frac{4}{3} + \frac{2}{3}x\right)\varphi(x)$$

satisface

$$g(1) = 0, \quad g'(1) = 0, \quad g''(x) = -\frac{4\varphi(x)}{3x} \leq 0.$$

Por tanto, para ξ cumpliendo que $|\xi - 1| < |x - 1|$, tenemos

$$g(x) = g(1) + g'(1)(x-1) + \frac{g''(\xi)}{2}(x-1)^2 = -\frac{4\varphi(\xi)}{6\xi}(x-1)^2 \leq 0,$$

lo cual demuestra (1.31). Utilizando esta desigualdad obtenemos que, si $P \ll Q$:

$$\begin{aligned} d_{TV}(P,Q) &= \frac{1}{2}\int_\Omega |p-q|d\mu = \frac{1}{2}\int_\Omega \left|\frac{p}{q} - 1\right| q d\mu \\ &\leq \frac{1}{2}\int_\Omega q\sqrt{\left(\frac{4}{3} + \frac{2p}{3q}\right)\varphi\left(\frac{p}{q}\right)}d\mu \\ &\leq \frac{1}{2}\sqrt{\int_\Omega \left(\frac{4q}{3} + \frac{2p}{3}\right)d\mu}\sqrt{\int_\Omega q\varphi\left(\frac{p}{q}\right)d\mu} \\ &= \sqrt{\frac{1}{2}\int_\Omega p\log\frac{p}{q}d\mu} = \sqrt{\frac{D_{KL}(P\|Q)}{2}}. \end{aligned}$$

De esta forma, obtenemos la desigualdad buscada. □

Lema 1.5. *Sean P y Q dos medidas de probabilidad en el espacio medible (Ω, σ). Entonces, se cumple que:*

$$d_{TV}(P,Q) \leq \sqrt{2}\delta, \tag{1.32}$$

donde δ es la distancia de Jensen-Shannon.

Demostración.

Por la desigualdad de Pinsker (1.30) se cumple que:

$$\frac{D_{KL}\left(P\middle\|\frac{P+Q}{2}\right)}{2} \geq d_{TV}^2\left(P, \frac{P+Q}{2}\right) = \frac{d_{TV}^2(P,Q)}{4},$$

y

$$\frac{D_{KL}\left(Q\middle\|\frac{P+Q}{2}\right)}{2} \geq d_{TV}^2\left(Q, \frac{P+Q}{2}\right) = \frac{d_{TV}^2(P,Q)}{4}.$$

En ambos casos, la última igualdad se deduce a partir de de la expresión (1.21) para la distancia de variación total:

$$\begin{aligned} d_{TV}\left(P, \frac{P+Q}{2}\right) &= \sup_{A\in\sigma}\left|P(A) - \frac{P(A)+Q(A)}{2}\right| \\ &= \frac{1}{2}\sup_{A\in\sigma}|P(A)-Q(A)| \\ &= \frac{1}{2}d_{TV}(P,Q). \end{aligned}$$

A partir de las expresiones anteriores, por la definición de la divergencia de Jensen-Shannon, tenemos que

$$D_{JS}(P,Q) \geq \frac{d_{TV}^2}{2}(P,Q), \tag{1.33}$$

y recordando que $\delta = \sqrt{D_{JS}(P,Q)}$, concluimos que

$$d_{TV}(P,Q) \leq \sqrt{2}\delta.$$

□

1.2. Aprendizaje supervisado

El *aprendizaje automático* (*machine learning*) se enmarca en el contexto de la teoría del aprendizaje estadístico y tiene como objetivo extraer patrones de un conjunto de datos de entrada sin procesar con el fin de encontrar un modelo que realice predicciones con precisión. La mayoría de los problemas se pueden clasificar en dos categorías: *aprendizaje supervisado* y *aprendizaje no supervisado*.

En el problema de aprendizaje supervisado se utiliza un conjunto de entrenamiento en el que los datos se organizan en parejas: cada *atributo* lleva asociada su correspondiente *etiqueta*. El objetivo es encontrar una regla que en futuras observaciones permita predecir la etiqueta observando únicamente el atributo. Dependiendo de si la variable respuesta es discreta o continua, se distingue además entre problemas de *clasificación* y de *regresión*, respectivamente. Entre las numerosas técnicas que existen para dar solución a los problemas de aprendizaje supervisado cabe destacar el *Análisis Discriminante (LDA)*, *Vecinos Más Próximos (KNN)*, *Máquinas de Vectores de Soporte (SVM)*, *árboles de decisión*, *bosques aleatorios* o la *regresión logística*.

En cambio, en los problemas de aprendizaje no supervisado no hay etiquetas preasignadas. Algunos métodos para resolver la gran variedad de problemas que existe de este tipo son el *Análisis Clúster*, el *Análisis de Componentes Principales (PCA)* o el *Análisis de Correspondencias (AC)*. Para mayor información se puede consultar [35].

No obstante, la línea que distingue ambos problemas de aprendizaje es, a menudo, difusa. De hecho, existen algunas técnicas de aprendizaje automático que se pueden utilizar para llevar a cabo ambas tareas. En nuestro caso, puesto que tenemos por objetivo estudiar las Redes Generativas Adversariales, nos centraremos en explicar los conceptos básicos del aprendizaje supervisado, ya que las GANs funcionan con conjuntos de datos con etiquetas ya asignadas (en particular, como veremos posteriormente: "imagen real" o "imagen generada").

La organización de la sección es la siguiente:

- Se exponen los fundamentos del *aprendizaje supervisado*, explicándolo desde el punto de vista estadístico y numérico, pues permite entender los algoritmos de aprendizaje automático que llevan a cabo las redes neuronales, en particular, las que componen las GANs. Se aborda la descripción de conceptos básicos como la *función de pérdida* que cuantifica la bondad de la predicción, para lo cual son imprescindibles los *clasificadores*, especialmente los lineales si se componen con una función de pérdida convexa. La razón de esto reside en que la convexidad facilita la resolución de la *minimización del riesgo empírico (ERM)*, cuya similitud con el método de máxima verosimilitud también mencionaremos.

- ERM se trata de un problema de optimización y, como tal, se puede abordar mediante el algoritmo de *Descenso de Gradiente*. De esta forma veremos que se pueden ajustar correctamente los parámetros del modelo, pero haciendo frente a una serie de inconvenientes habituales como el *sobreajuste* e *infrajuste*, similares al tradicional problema estadístico de "sesgo-varianza" y para cuya resolución se ha ideado la división del conjunto de datos en *conjuntos de entrenamiento y test*, así como técnicas de regularización como la *interrupción anticipada*. Además, en la práctica, se utilizan ciertos subconjuntos, *minilotes*, que se extraen aleatoriamente del conjunto de entrenamiento con el fin de reducir los exigentes requerimientos de almacenamiento computacionales.
- Se abordará también una descripción de los tipos de problemas de aprendizaje supervisado: *clasificación* y *regresión*, mencionando algunos de los algoritmos más importantes que se utilizan para resolverlos. En particular, se detallará la *regresión logística* para tareas de clasificación binaria, por su analogía con el problema planteado en las GANs que podremos ver en el Capítulo 3.

1.2.1. Marco estadístico

En primer lugar, formulemos matemáticamente el problema, lo cual resulta fundamental para entenderlo y generalizarlo después al caso de las GANs. Como referencia emplearemos [68] (*Capítulos 2, 3 y 9*).

Supongamos que los datos de entrada son $\mathbf{x} = (x_1, \dots, x_p)^\top \in \mathcal{X}$, donde $\mathcal{X} = \mathbb{R}^p$, $p \geq 1$, es el *dominio*, es decir, el conjunto al que pertenecen los datos que queremos estudiar (también se conocen como *atributos*). Sea $\mathcal{Y}$ un espacio medible. Este es el conjunto de posibles *etiquetas* o *clases* a las que pueden pertenecer los datos de entrada.

El objetivo es predecir, a partir de $\mathbf{x} \in \mathcal{X}$, la etiqueta correspondiente $y \in \mathcal{Y}$. Para ello, los datos de salida a partir de los de entrada vienen dados por una cierta *función predictora* $h : \mathcal{X} \to \mathcal{Y}$, con $h \in \mathcal{H}$, el *espacio de clasificadores o predictores* (también conocido como *clase de hipótesis*). El éxito de la predicción, es decir, lo que esta se diferencia del valor real, se medie mediante la *función de pérdida* $\ell : (\mathcal{X} \times \mathcal{Y}) \times \mathcal{H} \to \mathbb{R}^+$.

Sea D una distribución desconocida y perteneciente al conjunto de todas las posibles distribuciones de probabilidad de $\mathcal{X} \times \mathcal{Y}$, es decir, $D \in \mathcal{P}(\mathcal{X} \times \mathcal{Y})$. Con el fin de encontrar el predictor que optimice la pérdida, se define el *riesgo* (asociado a la regla de clasificación) como sigue, y se busca que sea lo más bajo posible:

$$\hat{\ell}_D(h) = \mathbb{E}_{(\mathbf{x},y)\sim D}[\ell((\mathbf{x}, y), h)]. \tag{1.34}$$

Puesto que (1.34) depende de D, que lo desconocemos, se realiza la siguiente aproximación: ***Minimización del Riesgo Empírico (ERM)***, que describimos a continuación.

El algoritmo de aprendizaje recibe como datos de entrada un *conjunto de entrenamiento* con n ejemplos, $S = \{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \dots, (\mathbf{x}_n, y_n)\}$, donde $(\mathbf{x}_i, y_i) \sim D$. Por tanto, se estima (1.34) mediante el *riesgo empírico*:

$$\hat{\ell}_S(h) = \frac{1}{n} \sum_{i=1}^{n} \ell((\mathbf{x}_i, y_i), h). \tag{1.35}$$

Entonces, ERM consiste en buscar $\hat{h} \in \mathcal{H}$ tal que

$$\hat{h} = \underset{h\in\mathcal{H}}{\operatorname{argmin}}\, \hat{\ell}_S(h). \tag{1.36}$$

1.2.2. Optimización: Descenso de Gradiente

Tal y como se acaba de exponer, el objetivo de cualquier algoritmo de aprendizaje es resolver (1.36). Se trata de un problema de optimización que se suele solventar mediante los métodos iterativos de descenso de gradiente que se mostrarán a continuación, empleando como referencia la información expuesta al respecto en [30] (*Capítulo 8*) y [68] (*Capítulo 14*).

En primer lugar, expongamos el algoritmo de *Descenso de Gradiente* en su versión general, tratando de explicarlo y justificarlo con el fin de obtener finalmente una particularización a nuestro caso concreto.

Algoritmo 1: Algoritmo de Descenso de Gradiente: el objetivo es lograr la convergencia hacia el valor mínimo de cualquier función diferenciable y convexa. Consiste en mejorar la solución en cada iteración avanzando en la dirección contraria al gradiente de dicha función en el punto en que nos encontremos. El ratio de aprendizaje η controla cuánto se avanza. La salida también podría ser el último vector $\mathbf{x}^{(T)}$, o el que tenga mejor rendimiento, $\operatorname{argmin}_{t\in[T]} f\left(\mathbf{x}^{(t)}\right)$, pero tomar el vector promedio suele resultar lo más útil.

Resultado: $\overline{\mathbf{x}} = \frac{1}{T}\sum_{t=1}^{T} \mathbf{x}^{(t)}$;

$f : \mathbb{R}^d \to \mathbb{R}$ diferenciable y convexa;
Se inicializa $\mathbf{x}^{(1)}$, por ejemplo a 0;
Se inicializa el ratio de aprendizaje η, preferiblemente en el intervalo $(0, 1)$;
Número de iteraciones $T > 0$;
para $t = 0, \ldots, T$ **hacer**
  $\mathbf{x}^{(t+1)} = \mathbf{x}^{(t)} - \eta \nabla_{\mathbf{x}} f\left(\mathbf{x}^{(t)}\right)$;
fin

El algoritmo de Descenso de Gradiente sirve, por tanto, para minimizar una función y consiste en mejorar la solución de forma iterativa avanzando en dirección contraria al gradiente en el punto en que nos encontremos, y que esperamos que esté cerca del mínimo absoluto. La justificación de esta dirección de avance se obtiene a partir del desarrollo de Taylor:

$$f(\mathbf{x} + h\mathbf{u}) \approx f(\mathbf{x}) + h\mathbf{u}\nabla f(\mathbf{x}), \quad \|\mathbf{u}\| = 1, \tag{1.37}$$

de donde se deduce que la dirección de máximo decrecimiento es:

$$\mathbf{u} = \frac{-\nabla f(\mathbf{x})}{\|\nabla f(\mathbf{x})\|}. \tag{1.38}$$

Respecto a la convergencia del algoritmo, existen resultados teóricos (ver [9]) que la garantizan si la función objetivo es **convexa** y satisface algunas condiciones de regularidad. En ese caso, el ratio de aprendizaje η se puede elegir bien. Sin embargo, si la función no es convexa, la situación es mucho más complicada. Por ejemplo, no se sabe cómo escoger η y, a veces, hay que probar por ensayo y error. Además, evaluar el gradiente puede ser enormemente costoso a nivel computacional, por lo que se plantea la alternativa de recurrir a evaluar promedios sobre un subconjunto de muestras aleatorias independientes e igualmente distribuidas extraídas del conjunto de entrenamiento conocido como ***minilote*** (o *minibatch*). Otra razón que motiva esta práctica es que es posible tener numerosas muestras en el conjunto de entrenamiento cuyas contribuciones al gradiente de la función de pérdida sean muy similares, de forma utilizando un minilote nos ahorraremos computarlas de forma repetida.

Los algoritmos que no utilizan el conjunto de entrenamiento en su totalidad, sino uno o varios minilotes se suelen identificar con el término *estocásticos*; en particular, detallaremos el *Algoritmo de Descenso de Gradiente Estocástico (SGD)* con el fin de resolver ERM (1.36), que es el caso que nos ocupa.

Notemos que un aspecto clave para la utilización de minilotes es que la función de pérdida se descompone en una suma sobre las muestras de entrenamiento, tal y como se hace al trabajar con el riesgo empírico (1.35). También debemos tener en cuenta que el clasificador elegido $h \in \mathcal{H}$ depende de ciertos parámetros $\boldsymbol{\theta}$ característicos (como detallaremos posteriormente en la sección 1.2.4) que se pueden configurar a partir de los datos de entrada. En consecuencia, SGD tratará de encontrar aquellos que minimicen el riesgo empírico (1.35).

Respecto a esto, cabe mencionar además que ahora en la práctica cada vez serán menores los cambios en dichos parámetros, pudiendo ser ajustados con mayor precisión. La razón es que se reduce gradualmente

el ratio de aprendizaje debido a que el estimador de SGD introduce ruido al extraer aleatoriamente los minilotes, que ni siquiera desvanece cuando se alcanza el mínimo.

Algoritmo 2: Algoritmo de Descenso de Gradiente Estocástico: actualización progresiva de los parámetros de la función de pérdida con el fin de minimizarla. Se trata de una particularización del algoritmo de Descenso de Gradiente que utiliza en cada iteración un minilote y mejora la velocidad de convergencia.

Inicialización de los parámetros de la función de pérdida $\boldsymbol{\theta}$;
Inicialización del ratio de aprendizaje, preferiblemente $\eta_0 \in (0, 1)$;
Reducción gradual η_t;
Número de iteraciones $T > 0$;
para $t = 0, \ldots, T$ **hacer**
  Extacción de un minilote $\{\mathbf{x}_1, \ldots, \mathbf{x}_m\}$;
  Etiquetas correspondientes al minilote $\{y_1, \ldots, y_m\}$;
  para $i = 0, \ldots, m$ **hacer**
    Estimación del gradiente: $\hat{\boldsymbol{g}} \leftarrow \frac{1}{m} \nabla_{\boldsymbol{\theta}} \sum_i \ell((\mathbf{x}_i, y_i), h_{\boldsymbol{\theta}})$;
    Actualización parámetros: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \eta_t \hat{\boldsymbol{g}}$;
  fin
fin

Existen técnicas que tratan de mejorar la convergencia del algoritmo SGD como son: *SGD con momento*, *SGD con momento de Nesterov* o algoritmos con ratios de aprendizaje adaptables como *AdaGrad*, *RMSProp* o *Adam*. Sin embargo, en la actualidad no hay consenso sobre qué algoritmo de optimización elegir y debe seleccionarse mediante prueba y error. Para obtener una explicación más detallada al respecto se puede consultar [30] (*Capítulo 8*). En la práctica el que más se utiliza, sobre todo cuando el tamaño del conjunto de datos es grande, es *Adam*, por lo que indicaremos como referencia el papel original [41], pues una descripción detallada se escapa de los objetivos del trabajo. Basta conocer que se trata de una mejora del algoritmo SGD que emplearemos en el capítulo final de implementación para optimizar los resultados.

Observemos que al resolver el problema de la optimización de la función objetivo han aparecido parámetros configurables, pero que no dependen del conjunto de datos de entrada, motivo por el cual se les conoce como *hiperparámetros*. El más importante es el ratio de aprendizaje η, que controla cuánto cambian en cada iteración los parámetros $\boldsymbol{\theta}$ a la hora de buscar el mínimo del riesgo empírico. En otras palabras, controla la velocidad a la que el algoritmo es capaz de resolver ERM, por lo que es de vital importancia. Como sabemos, en ocasiones, especialmente si la función no es convexa y no tiene ciertas condiciones de regularidad, se debe elegir mediante prueba y error, por ejemplo mediante las curvas de aprendizaje que expondremos en la sección 1.2.3. No obstante, se recomienda que se encuentre en el intervalo $(0, 1)$ [3]. El número de iteraciones T es también un hiperparámetro y se puede seleccionar de manera óptima también mediante las técnicas descritas posteriormente.

Otro hiperparámetro es el tamaño de los minilotes, que se elige teniendo en cuenta los siguientes criterios prácticos que nos serán de utilidad y deberemos recordar a la hora de llevar a cabo en el presente trabajo la implementación final de una GAN:

- Los procesadores multinúcleo de los ordenadores actuales no se aprovechan completamente con lotes demasiados pequeños, por lo que suele establecerse un mínimo en su tamaño.
- Si todas muestras del lote se van a procesar paralelamente (como suele ser habitual), entonces la cantidad de memoria necesaria es proporcional al tamaño del lote. Para muchas configuraciones de hardware este es el factor limitante.
- Algunos tipos de hardware logran mejores resultados en el tiempo de cómputo con arrays de dimensiones específicas. En particular, cuando se utilizan GPUs (Unidad de Procesamiento Gráfico) es común que los tamaños de los lotes que sean potencias de 2 ofrezcan mejor rendimiento.

En definitiva, es evidente que la optimización es, generalmente, una tarea compleja. Por este motivo, estos algoritmos que se acaban de exponer presentan algunos problemas a la hora de ser implementados

en la práctica. Por ejemplo, que las funciones de pérdida no sean diferenciables, en cuyo caso, se emplean funciones sustitutas. Incluso cuando estas funciones son convexas, aparecen algunos problemas como gradientes que desvanecen o que explotan, o que son inexactos. Puesto que existen varias posibilidades de elección de la función de pérdida dependiendo del problema al que nos estemos enfrentando y la técnica elegida para resolverlo, haremos alusión a estos inconvenientes y cómo solventarlos más adelante, concretamente en la sección dedicada a las técnicas de aprendizaje 1.2.4.

1.2.3. Sobreajuste e infrajuste

El desafío central en aprendizaje automático es que el algoritmo realice predicciones correctamente cuando se empleen nuevos datos de entrada distintos a las que hayan sido empleadas para entrenarlo, lo cual se conoce como *generalización*.

Acabamos de ver que para llevar a cabo el entrenamiento de un algoritmo de aprendizaje automático se tiene acceso a un conjunto de entrenamiento $S = \{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots, (\mathbf{x}_n, y_n)\}$, $(\mathbf{x}_i, y_i) \sim D$, D es desconocida y $D \in \mathcal{P}(\mathcal{X} \times \mathcal{Y})$; el riesgo empírico $\hat{\ell}_S$ (1.35) representa lo que se conoce como *error de entrenamiento* y tratamos de minimizarlo mediante ERM.

Sin embargo, lo que nos interesa es determinar el *error de generalización*, también conocido como *error de test*, que nos interesa igualmente minimizarlo. Se define como el valor esperado del error cuando se utilizan nuevos datos de entrada, calculándose esta esperanza a partir de todas las posibles entradas cuya distribución sea la que se espera encontrar en la práctica. Es decir, el error de generalización es lo que habíamos definido como riesgo real $\hat{\ell}_D(\hat{h})$ (1.34). Puesto que D es desconocida, normalmente se estima esta cantidad midiendo el rendimiento del algoritmo en un *conjunto de test* seleccionado a partir de ejemplos distintos de los que componen el de entrenamiento.

Entonces, los factores que determinan la capacidad de aprendizaje de un algoritmo son la minimización tanto del error de entrenamiento como de la diferencia entre este y el error de test. Ambos problemas se corresponden con los desafíos centrales del aprendizaje automático: *sobreajuste* (u *overfitting*) e *infrajuste* (o *underfitting*). El sobreajuste ocurre cuando dicha diferencia es demasiado grande $\hat{\ell}_D(\hat{h}) \gg \hat{\ell}_S(\hat{h})$, es decir, el modelo comienza a memorizar las predicciones para el conjunto de entrenamiento y a no ser capaz de realizarlas correctamente para otros datos de entrada nuevos. La razón de esta alta variabilidad suele residir en la complejidad del modelo (demasiadas variables). Por su parte, el infrajuste tiene lugar cuando el modelo no es capaz de obtener un error de entrenamiento suficientemente bajo debido a que es más simple de lo necesario (pocas variables). Por tanto, podemos pensar que estos dos problemas representan el tradicional ***conflicto sesgo-varianza*** en Estadística. Veámoslo formalmente con el fin de entenderlo y tratar de controlarlo en lo posible posteriormente.

Sea $z_i = (\mathbf{x}_i, y_i)$, $z_i \sim D$. El objetivo es resolver ERM (2.4), es decir, buscamos $\hat{h}$ tal que $\hat{\ell}_S(\hat{h}) \leq \hat{\ell}_S(h)$, para todo $h \in \mathcal{H}$, donde $\hat{\ell}_S(h)$ es el riesgo empírico (1.35), que representa el error de entrenamiento. Nos interesa asimismo controlar el error de generalización, que se corresponde con el riesgo real $\hat{\ell}_D(\hat{h})$ (1.34).

Por un lado, tenemos una regla óptima que proviene de la regla de Bayes (ver [35], *Capítulo 2*):

$$h_B \quad \text{tal que} \quad \hat{\ell}_D(\hat{h}) \leq \hat{\ell}_D(h), \quad \forall h \in \mathcal{H}_\infty, \tag{1.39}$$

donde $\mathcal{H}_\infty$ denota la clase de todos los clasificadores posibles.

Por otro lado, existe una regla óptima en $\mathcal{H}$

$$\bar{h} \quad \text{tal que} \quad \hat{\ell}_D(\bar{h}) \leq \hat{\ell}_D(h), \quad \forall h \in \mathcal{H}. \tag{1.40}$$

Además, en cualquier procedimiento de aprendizaje estadístico, al analizar la regla obtenida, se obtiene lo que se conoce como *exceso de riesgo de la regla*:

$$\mathcal{E}_D(h) = \hat{\ell}_D(h) - \hat{\ell}_D(h_B). \tag{1.41}$$

En particular, lo que nos interesa es controlar el exceso de riesgo de ERM:

$$\mathcal{E}_D(h) = \hat{\ell}_D(\hat{h}) - \hat{\ell}_D(h_B). \tag{1.42}$$

Veamos, entonces, cómo acotarlo. Teniendo en cuenta la definición de $\hat{h}$ (1.36) y las dos reglas óptimas (1.39) y (1.40), se tiene que

$$\begin{aligned} \mathcal{E}_D(h) &= \hat{\ell}_D(\hat{h}) - \hat{\ell}_S(\hat{h}) + \hat{\ell}_S(\hat{h}) - \hat{\ell}_D(\bar{h}) + \hat{\ell}_D(\bar{h}) - \hat{\ell}_D(h_B). \\ &\leq 2 \sup_{h \in \mathcal{H}} |\hat{\ell}_S(h) - \hat{\ell}_D(h)| + \hat{\ell}_D(\bar{h}) - \hat{\ell}_D(h_B). \end{aligned} \tag{1.43}$$

El término

$$2 \sup_{h \in \mathcal{H}} |\hat{\ell}_S(h) - \hat{\ell}_D(h)| \tag{1.44}$$

es el *error de estimación*; por su parte,

$$\hat{\ell}_D(\bar{h}) - \hat{\ell}_D(h_B) = \min_{h \in \mathcal{H}} \hat{\ell}_D(h) - \min_{h \in \mathcal{H}_\infty} \hat{\ell}_D(h) \tag{1.45}$$

es el *error de aproximación*.

Notemos que, al aumentar el tamaño del modelo, es decir, cuanto mayor sea $\mathcal{H}$, mayor será el error de estimación, mientras que sucederá lo contrario con el error de aproximación, de ahí el término "conflicto" acuñado para hacer referencia a este problema. En otras palabras, al aumentar $\mathcal{H}$, estamos reduciendo solamente el error de aproximación, de forma que habrá un mejor rendimiento en el error de entrenamiento; pero esto no implica que se reduzca el exceso de riesgo $\mathcal{E}_D$, que es lo que realmente nos interesa para lograr un buen aprendizaje, minimizando así el error de generalización.

En base a lo que acabamos de exponer para la explicación del problema, pasemos a analizar cómo resolverlo. Para ello existen lo que se conocen como *técnicas de regularización*, que son especialmente útiles a la hora de ajustar los hiperparámetros, como el ratio de aprendizaje o el tamaño de los minilotes. Una de las más importantes y en la que nos centraremos consiste en estudiar la relación entre el sobreajuste y el infrajuste analizando las *curvas de aprendizaje* que se muestran a continuación.

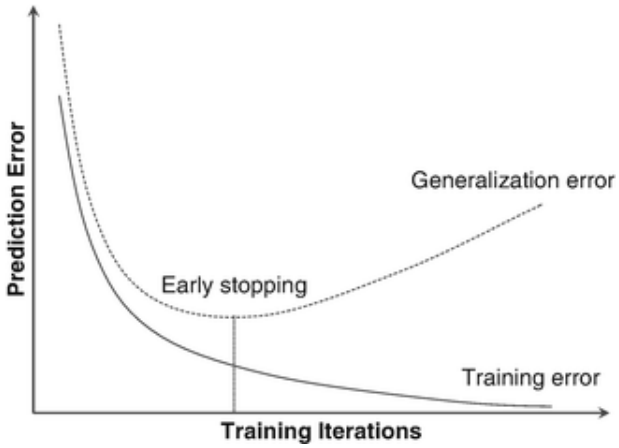

Figura 1.2: Curvas de aprendizaje. Fuente: [58].

Es evidente a partir de la figura 1.2 que el error de entrenamiento y el error de test se comportan de manera diferente. Al comienzo de las iteraciones ambos son altos: esta es la zona de infrajuste. Según se va avanzando el error de entrenamiento disminuye, pero la diferencia entre ambos aumenta a un ritmo que acaba superando el ritmo de decrecimiento del error de entrenamiento, momento en el que se entra en la zona de sobreajuste. La situación de rendimiento óptimo se obtiene llevando a cabo la técnica de *interrupción anticipada* (o *early stopping*), consistente en parar el algoritmo en el momento en que el error de test es mínimo, tal y como se indica en la gráfica.

Se puede considerar la interrupción anticipada como un parámetro más que se debe ajustar, ya que se trata de realizar varias pruebas en las que se irán guardando los valores de los parámetros del modelo para elegir aquellos con los que se obtengan los mejores resultados. La idea general es que si vemos que el error de generalización está empezando a decrecer, entonces algunas soluciones razonables son: aumentar el conjunto de datos o disminuir la complejidad de la clase de hipótesis $\mathcal{H}$. Por otra parte, si vemos que el error de test se estanca en torno a 1/2, no tendremos evidencia de que hayamos elegido correctamente $\mathcal{H}$. En definitiva, aumentar el conjunto de datos es una solución óptima pero, en términos de costo computacional, lo recomendable es probar primeramente a reducir la complejidad de la clase de hipótesis.

Veremos en el próximo capítulo que esta técnica resulta de especial utilidad en el entrenamiento de las redes neuronales, ya que en ese caso buscar el óptimo es aún más complicado por la gran cantidad de parámetros que interviene. Mediante la interrupción anticipada, por tanto, basta que se lleven a cabo unas pocas iteraciones del Descenso de Gradiente hasta el punto en que se observe un aumento del error de generalización.

Por último, cabe mencionar que existe la posibilidad de obtener una tercera división de nuestro conjunto de datos de entrada, independiente del resto y conocida como *conjunto de validación*. Se tratará entonces de entrenar diferentes algoritmos (o el mismo algoritmo con diferentes parámetros) en el conjunto de entrenamiento y aplicar ERM en el conjunto de validación. Las curvas de aprendizaje representando, en este caso, el error de entrenamiento y el error de validación, nos indicarán qué modelo es el adecuado. Una vez elegido el modelo con sus hiperparámetros adecuados, se comprueba su rendimiento en el conjunto test, de manera que se obtendrá una estimación óptima del error de generalización. Además, el proceso de validación se puede llevar a cabo particionando el conjunto tal y como hacen las técnicas de *validación cruzada* o *Leave-One-Out* (para mayor información consultar [68], *Capítulo 12*).

1.2.4. Tipos de algoritmos de aprendizaje supervisado

El problema de aprendizaje supervisado se resuelve mediante distintos tipos de algoritmos que mencionaremos en esta sección, centrándonos principalmente en la *regresión logística*. Para ello, veremos qué posibilidades de elección hay tanto de la función de pérdida como de los clasificadores, pues determinan el método a emplear como solución. Analizaremos, además, algunos de ellos según confieran buenos resultados o no en base a las condiciones anteriormente expuestas relativas a optimización, sobreajuste, etc.

En primer lugar, para poder elegir convenientemente el modelo, podemos distinguir dos tipos de problemas de aprendizaje supervisado:

- *Clasificación*: consiste en encontrar una función que logre categorizar el conjunto de datos en clases en función de los diferentes parámetros. La variable de salida debe ser un valor discreto, por ejemplo "masculino" o "femenino"; "verdadero" o "falso", etc., lo cual determina el conjunto $\mathcal{Y}$. El objetivo es encontrar el límite de decisión que permita dividir el conjunto de datos en clases. Existen dos tipos de clasificación: *binaria* y *multiclase*. Como ejemplos de aplicaciones podemos mencionar la identificación de spam [53], el reconocimiento por voz [51] o la detección de células cancerígenas [57].
- *Regresión*: consiste en encontrar las correlaciones entre variables dependientes e independientes. La variable de salida debe ser una variable continua o un valor real, por ejemplo el precio, el salario o la edad, lo cual determina el conjunto $\mathcal{Y}$. El objetivo es encontrar la mejor recta de ajuste que permita realizar predicciones con precisión. Existen dos tipos de regresión: *lineal* y *no lineal*. Entre sus aplicaciones nos encontramos, por ejemplo, la predicción meteorológica [56] o la predicción de los precios de la vivienda [84].

Existen varios algoritmos que sirven para solventar ambas tareas como *SVM*, los *árboles de decisión*, *bosques aleatorios*, etc., así como las *redes neuronales*. Sin embargo, también existen otros que son específicos como por ejemplo *mínimos cuadrados* para la regresión lineal; o *LDA* y *regresión logística*, ambos métodos lineales de clasificación. Para mayor información al respecto se puede consultar [35].

Resulta de especial interés en este trabajo el problema de clasificación binaria abordado mediante regresión logística ya que, como veremos, en las GANs será necesario discriminar si una imagen es "generada" o "falsa" y para tal fin se utiliza una función objetivo muy similar a la función de pérdida que emplea dicho método.

Pasemos ahora a enumerar varias posibilidades de elección del clasificador y de la función de pérdida que permitirán decidir el método empleado como solución.

En cuanto a cómo elegir la clase de hipótesis $\mathcal{H}$, notemos la importancia en la expresión que plantea el problema de ERM (1.36) de hacerlo adecuadamente para lograr un buen rendimiento del algoritmo evitando el *sobreajuste* o el *infrajuste*. De hecho, ya hemos visto que esta elección conlleva la existencia de ciertos parámetros que hemos englobado en $\boldsymbol{\theta}$ y que determinan el problema de optimización. Veamos, por tanto, un par de ejemplos importantes:

- *Vecino más cercano*: ocasiona problemas relativos a una alta varianza.

$$h_S(\mathbf{x}) = y_i \quad \text{tal que} \quad i = \underset{j}{\operatorname{argmin}}\, d(\mathbf{x}, \mathbf{x}_j). \tag{1.46}$$

- *Clasificador lineal*: es el que presenta mejores propiedades y es de los más utilizados. Además, demostraremos en la Proposición 1.1 la existencia de útiles variantes que resultan de composiciones con funciones no lineales, pero sí convexas, para introducir no linealidad cuando sea necesario.

$$h_{\mathbf{w},b}(\mathbf{x}) = \langle \mathbf{w}, \mathbf{x} \rangle + b \quad \mathbf{w} \in \mathbb{R}^p, \quad b \in \mathbb{R}. \tag{1.47}$$

Como hemos visto, la elección del clasificador determina los parámetros de la función de pérdida y, por tanto, cómo optimizarla. Centrémonos ahora en cómo elegir de manera adecuada la función de pérdida, lo cual se hace en base al tipo de problema al que nos enfrentemos que, a su vez, determina el conjunto de etiquetas $\mathcal{Y}$.

Se buscará en todo momento, si es posible, relacionarlas con los clasificadores lineales de manera que se comprenda la importancia y utilidad de estos. En ese caso, esto implica que los parámetros que las determinan serán $\mathbf{w}$ y b, los cuales veremos más adelante que se corresponden con los pesos y los sesgos en las redes neuronales. Veamos algunos ejemplos a continuación en los cuales, por simplicidad en la notación y en la exposición, supondremos $b = 0$ por el momento:

- Clasificación binaria: tiene lugar cuando solamente hay dos posibles clases en las que incluir los datos de salida y se toma $\mathcal{Y} = \{0, 1\}$ ó $\mathcal{Y} = \{-1, 1\}$. En cada caso, se pueden utilizar las siguientes funciones de pérdida, respectivamente:

  - *Pérdida 0-1*:

$$\ell((\mathbf{x}, y), h) = \mathbb{1}_{[y \neq h(\mathbf{x})]}, \quad y \in \{0, 1\}. \tag{1.48}$$

$$\ell((\mathbf{x}, y), h_{\mathbf{w}}) = \mathbb{1}_{[y \neq \operatorname{sign}(\langle \mathbf{w}, \mathbf{x} \rangle)]} = \mathbb{1}_{[y\langle \mathbf{w}, \mathbf{x} \rangle \leq 0]}, \quad y \in \{-1, 1\}. \tag{1.49}$$

  - *Hinge*:

$$\ell(\mathbf{x}, y), h_{\mathbf{w}}) = \max\{0, 1 - y\langle \mathbf{w}, \mathbf{x} \rangle\}, \quad \mathbf{w} \in \mathbb{R}^p, \quad y \in \{-1, 1\}. \tag{1.50}$$

  - *Entropía cruzada binaria (binary cross-entropy) o pérdida logística*: útil en tareas de regresión logística, que se utiliza para clasificación binaria.

$$\ell\left((\mathbf{x}, y), h_{\mathbf{w}}\right) = \log(1 + \exp(-y\langle \mathbf{w}, \mathbf{x} \rangle)), \quad \mathbf{w} \in \mathbb{R}^p, \quad y \in \{-1, 1\}. \tag{1.51}$$

  La pérdida 0-1 surge de manera natural, y se emplea también en problemas de clasificación multiclase. Sin embargo, lleva a problemas ERM no tratables. Para solucionarlos, se proponen convexificaciones como son hinge o la pérdida logística, tal y como veremos posteriormente en esta misma sección (figura 1.3).

- Clasificación multiclase: tiene lugar cuando hay más de dos clases posibles. Se elige $\mathcal{Y} = \{0, 1, \ldots, k\}$ y como funciones de pérdida se pueden tomar, entre otras:
  - *Divergencia de Kullback*: supongamos que la distribución de probabilidad real de los datos es P y que el modelo la aproxima por P_θ. Lo que difieren ambas distribuciones es lo que mide la divergencia de Kullback, tal y como vimos en la sección 1.1.1.

$$D_{KL}(P\|P_\theta) = \sum_x P(x) \log \frac{P(x)}{P_\theta(x)}. \tag{1.52}$$

  - *Entropía cruzada (cross-entropy)*: se define de manera similar a la anterior.

$$H(P\|P_\theta) = -\sum_x P(x) \log P_\theta(x). \tag{1.53}$$

- Regresión: en este caso, se trata de encontrar una relación funcional entre los componentes de $\mathcal{X}$ e $\mathcal{Y}$. Se toma $\mathcal{Y} = \mathbb{R}$. Algunos ejemplos de funciones de pérdida que se pueden escoger son:
  - *Pérdida cuadrática*: útil cuando se emplea el método de mínimos cuadrados.

$$\ell((\mathbf{x}, y), h_{\mathbf{w}}) = (h_{\mathbf{w}}(\mathbf{x}) - y)^2 = (\langle \mathbf{w}, \mathbf{x} \rangle - y)^2. \tag{1.54}$$

  - *Pérdida en valor absoluto*:

$$\ell((\mathbf{x}, y), h) = |h(\mathbf{x}) - y|. \tag{1.55}$$

La deducción de las expresiones (1.49), (1.50), (1.51) y (1.54) con el fin de buscar la relación con los clasificadores lineales, o sus variantes provenientes de composiciones con funciones convexas, se puede consultar en [68] (págs. 167, 126-127 y 163-164). En particular, la de (1.51) se hará en la subsección 1.2.4.2.

Tal y como habíamos adelantado, con lo que se acaba de exponer y como consecuencia de la utilización de clasificadores lineales, se observa que las funciones de pérdida están parametrizadas por un vector $\mathbf{w} \in \mathbb{R}^p$ y por una constante b (que habíamos supuesto nula), con lo que se redefine la minimización del riesgo empírico como:

$$\hat{h} = \underset{\mathbf{w} \in \mathcal{H}}{\operatorname{argmin}}\, \hat{\ell}_S(\mathbf{w}). \tag{1.56}$$

Esta reinterpretación nos será útil para establecer condiciones sobre las funciones de pérdida y para describir el funcionamiento de las redes neuronales posteriormente, donde los pesos asociados a cada capa jugarán el papel de $\mathbf{w}$.

Cabe destacar, ya que se expuso anteriormente la estimación de máxima verosimilitud, que esta juega un papel importante en este marco teórico, pues es similar a la minimización del riesgo empírico en el sentido de que muchas funciones de pérdida se pueden deducir a partir de dicho estimador. Por ejemplo, se puede demostrar que es así para el error cuadrático medio (el riesgo empírico asociado a la función de pérdida cuadrática (1.54)) (consultar en [30], *Capítulo 5*, págs. 131-132) y la entropía cruzada (1.53) (consultar en [68], pág. 345).

1.2.4.1. Problemas de aprendizaje convexos

Ahora que ya conocemos las posibilidades de elección tanto de la clase de hipótesis como de la función de pérdida, pasemos a analizar bajo qué condiciones se puede resolver ERM (1.56). Nos centraremos en explicar, en líneas generales, los problemas de aprendizaje relacionados con la convexidad que pueden consultarse en mayor profundidad en [68] (*Capítulo 12*). Esto nos resulta de interés porque sabemos que la optimización de una función convexa es factible y alcanza el extremo absoluto, lo cual garantiza el buen funcionamiento de los Algoritmos 1 y 2.

En primer lugar, se dice que un problema de aprendizaje es convexo si el conjunto $\mathcal{H}$ es convexo y la función de pérdida $\ell(\mathbf{w}, z)$ es convexa en $\mathbf{w}$ para todo $z = (\mathbf{x}, y) \in \mathcal{X} \times \mathcal{Y}$. En ese caso, el problema de ERM (1.56) se puede considerar un problema convexo de optimización que, por tanto, puede resolverse eficientemente al ser posible garantizar que un extremo local es realmente un extremo global y, por tanto, la solución óptima del problema.

No obstante, no todos los problema convexos pueden ser aprendidos por un algoritmo (ver contra-ejemplo en [68], págs. 164-165). Son necesarias más restricciones: además de convexas y diferenciables, las funciones de pérdida deben ser acotadas, de Lipschitz o suaves.

Si no lo cumplen, como ocurre de manera habitual, se pueden implementar, en su lugar, las funciones convexas denominadas *funciones de pérdida sustitutas*. Por ejemplo, la función de pérdida 0-1 (1.48) no es convexa. Pero, si $\mathcal{Y} = \{-1, 1\}$, se puede considerar la función hinge (1.50) o la pérdida logística (1.51) que son convexas, como sustitutas, tal y como podemos ver en la figura 1.3.

Cabe destacar, además, que la función de pérdida cuadrática (1.54) también es convexa. Justificaremos todo ello de forma que demostraremos lo que ya se anunció previamente acerca de la importancia y utilidad de los clasificadores lineales, a veces compuestos con determinadas funciones convexas en la siguiente Proposición 1.1.

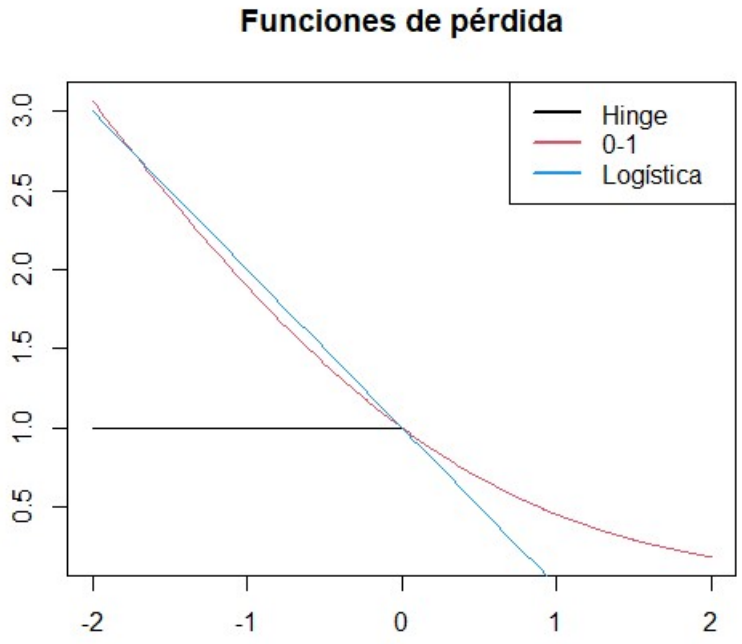

Figura 1.3: Función de pérdida 0-1. Hinge y logística como sustitutas.

Proposición 1.1. *Sea $f : \mathbb{R}^p \to \mathbb{R}$ tal que $f(\mathbf{w}) = g(\langle \mathbf{w}, \mathbf{x} \rangle + y)$, para cierto $\mathbf{x} \in \mathbb{R}^p$, $y \in \mathbb{R}$, y $g : \mathbb{R} \to \mathbb{R}$. Entonces, la convexidad de g implica la convexidad de f.*

Demostración.
Sean $\mathbf{w}_1, \mathbf{w}_2 \in \mathbb{R}^p$ y $\alpha \in [0, 1]$. Tenemos que:

$$\begin{aligned} f\left(\alpha \mathbf{w}_1 + (1-\alpha)\mathbf{w}_2\right) &= g\left(\langle \alpha \mathbf{w}_1 + (1-\alpha)\mathbf{w}_2, \mathbf{x} \rangle + y\right) \\ &= g\left(\alpha \langle \mathbf{w}_1, \mathbf{x} \rangle + (1-\alpha) \langle \mathbf{w}_2, \mathbf{x} \rangle + y\right) \\ &= g\left(\alpha \left(\langle \mathbf{w}_1, \mathbf{x} \rangle + y\right) + (1-\alpha) \left(\langle \mathbf{w}_2, \mathbf{x} \rangle + y\right)\right) \\ &\leq \alpha g\left(\langle \mathbf{w}_1, \mathbf{x} \rangle + y\right) + (1-\alpha) g\left(\langle \mathbf{w}_2, \mathbf{x} \rangle + y\right), \end{aligned}$$

donde de la última desigualdad se deduce la convexidad de g. □
Es decir, hemos demostrado que la composición de una función convexa con una función lineal es convexa.

Por su parte, la entropía cruzada (1.53) también es convexa puesto que se puede escribir como:

$$H(P\|P_\theta) = H(P) + D_{KL}(P\|P_\theta).$$

Dado que $H(P)$ es fijo, la convexidad se deduce de la convexidad de la divergencia de Kullback, que ya se demostró anteriormente en el Teorema 1.1.

1.2.4.2. Regresión logística

Centrémonos finalmente en la regresión logística como método lineal de clasificación, en particular, binaria, pues ya hemos mencionado que por analogía nos será de utilidad a la hora de comprender el funcionamiento a nivel estadístico de las GANs. Este particular tipo de red neuronal veremos que trata de discriminar entre imágenes "generadas" o "reales" mediante una función objetivo (3.8) que recuerda a la de la regresión logística, relación que detallaremos en el correspondiente Capítulo 3. Como referencia principal emplearemos [35] (*Capítulo 4*).

Supongamos que nuestro predictor $h \in \mathcal{H}$ toma valores discretos en $\mathcal{Y}$ y que hay K clases etiquetadas mediante $1, 2, \ldots, K$. El ajuste lineal para el indicador de la variable respuesta $k-$ésima es $\hat{f}_k(\mathbf{x}) = \hat{w}_{k0} + \hat{\mathbf{w}}_k^T\mathbf{x} = \hat{w}_{k0} + \langle\hat{\mathbf{w}}_k, \mathbf{x}\rangle$. El límite de decisión entre la clase k y la clase l es el conjunto de puntos tales que $\hat{f}_k(\mathbf{x}) = \hat{f}_l(\mathbf{x})$, es decir, el conjunto: $\left\{\mathbf{x} : (\hat{w}_{k0} - \hat{w}_{l0}) + (\hat{\mathbf{w}}_k - \hat{\mathbf{w}}_l)^T\mathbf{x} = 0\right\}$, un conjunto afín de hiperplanos. Puesto que esto mismo ocurre para cualquier par de clases, el conjunto de datos de entrada se divide en regiones de clasificación constantes con hiperplanos como límites de decisión.

Existen dos formas de determinarlas: mediante modelos que emplean *funciones discriminantes* $\delta_k(\mathbf{x})$ para cada clase y clasifican cada $\mathbf{x}$ según la clase cuya función de como resultado un mayor valor, como es el caso de LDA; y modelos que emplean las *probabilidades a posteriori* $\mathrm{P}(h = K|X = \mathbf{x})$, como es el caso de la regresión logística. Si, o bien $\delta_k(\mathbf{x})$, o bien $P(h = K|X = \mathbf{x})$ son lineales en $\mathbf{x}$, entonces el límite de decisión será también lineal. Por ejemplo, si hay solamente dos clases, $K = 2$, un popular modelo para los probabilidades a posteriori es

$$\begin{aligned}\mathrm{P}(h = 1 \mid X = \mathbf{x}) &= \frac{\exp\left(w_0 + \mathbf{w}^T\mathbf{x}\right)}{1 + \exp\left(w_0 + \mathbf{w}^T\mathbf{x}\right)},\\ \mathrm{P}(h = 2 \mid X = \mathbf{x}) &= \frac{1}{1 + \exp\left(w_0 + \mathbf{w}^T\mathbf{x}\right)}.\end{aligned} \tag{1.57}$$

Llevando a cabo la *transformación logit* [17] llegamos a que

$$\log\frac{\mathrm{P}(h = 1 \mid X = \mathbf{x})}{\mathrm{P}(h = 2 \mid X = \mathbf{x})} = w_0 + \mathbf{w}^T\mathbf{x}. \tag{1.58}$$

Entonces, el límite de decisión en este caso de clasificación binaria es el hiperplano $\left\{\mathbf{x} \mid w_0 + \mathbf{w}^T\mathbf{x} = 0\right\}$.

En este contexto surge el modelo de regresión logística con el objetivo de modelar las probabilidades a posteriori de las K clases por medio de funciones lineales en $\mathbf{x}$ asegurando, al mismo tiempo, que suman 1 y que permanecen en el intervalo $[0, 1]$. El modelo tiene la forma:

$$\begin{aligned}\mathrm{P}(h = k \mid X = \mathbf{x}) &= \frac{\exp\left(w_{k0} + \mathbf{w}_k^T\mathbf{x}\right)}{1 + \sum_{l=1}^{K-1}\exp\left(w_{l0} + \mathbf{w}_l^T\mathbf{x}\right)}, \quad k = 1, \ldots, K-1.\\ \mathrm{P}(h = K \mid X = \mathbf{x}) &= \frac{1}{1 + \sum_{l=1}^{K-1}\exp\left(w_{l0} + \mathbf{w}_l^T\mathbf{x}\right)}.\end{aligned} \tag{1.59}$$

En este caso el conjunto completo de parámetros es $\boldsymbol{\theta} = \left\{w_{10}, \mathbf{w}_1^T, \ldots, w_{(K-1)0}, \mathbf{w}_{K-1}^T\right\}$. Para enfatizar esta dependencia denotaremos las probabilidades $\mathrm{P}(h = k \mid X = \mathbf{x}) = p_k(\mathbf{x}; \boldsymbol{\theta})$.

Los modelos de regresión logística se ajustan normalmente mediante el *método de máxima verosimilitud* utilizando la probabilidad condicionada $\mathrm{P}(h \mid X)$. Entonces, la log-verosimilitud para n observaciones es

$$\ell(\boldsymbol{\theta})) = \sum_{i=1}^{n}\log p_{g_i}\left(\mathbf{x}_i; \boldsymbol{\theta}\right)), \tag{1.60}$$

donde $p_k(\mathbf{x}_i;\boldsymbol{\theta}) = \mathrm{P}(h = k \mid X = \mathbf{x}_i;\boldsymbol{\theta})$.

En el caso de que haya solamente dos clases, $K = 2$, el algoritmo se simplifica notablemente. De hecho, es conveniente codificar las dos clases h_i por medio de la variable respuesta y_i, donde $y_i = 1$ cuando $h_i = 1$, e $y_i = 0$ cuando $h_i = 2$. Sean, además, $p_1(\mathbf{x};\boldsymbol{\theta}) = p(\mathbf{x};\boldsymbol{\theta})$, y $p_2(\mathbf{x};\boldsymbol{\theta}) = 1 - p(\mathbf{x};\boldsymbol{\theta})$. Por tanto, la log-verosimilitud se puede escribir como

$$\begin{aligned}\ell(\mathbf{w}) &= \sum_{i=1}^{n} \left\{ y_i \log p\left(\mathbf{x}_i;\mathbf{w}\right) + (1-y_i) \log\left(1 - p\left(\mathbf{x}_i;\mathbf{w}\right)\right)\right\} \\ &= \sum_{i=1}^{n} \left\{ y_i \mathbf{w}^T \mathbf{x}_i - \log\left(1 + e^{\mathbf{w}^T \mathbf{x}_i}\right)\right\},\end{aligned} \tag{1.61}$$

donde $\mathbf{w} = \{w_{10}, \mathbf{w}_1\}$ (asumiendo que el vector de entrada $\mathbf{x}_i$ incluye el término constante 1) y $p\left(\mathbf{x}_i;\mathbf{w}\right) = \hat{y}_i$ es la etiqueta que se predice para la entrada $\mathbf{x}_i$.

Lo que nos resultará especialmente interesante en el desarrollo posterior del Capítulo 3 es que para ajustar el modelo correctamente se debe maximizar la log-verosimilitud (1.61), lo cual es equivalente a minimizar la función de pérdida logística (1.51) cuando $y \in \{-1, 1\}$ y para todo n. Esto se intuye si nos fijamos en (1.57) y tenemos en cuenta cómo hemos codificado las clases cuando hay solamente dos. Nos gustaría que $\mathrm{P}(h = 1 \mid X = \mathbf{x}_i)$ fuera grande si $y_i = 1$, y que $1 - \mathrm{P}(h = 1 \mid X = \mathbf{x}_i) = \mathrm{P}(h = 2 \mid X = \mathbf{x}_i)$ fuera grande si $y_i = -1$. Entonces, cualquier función de pérdida crecerá con $\frac{1}{1+\exp\{y(w_0 + \mathbf{w}^T\mathbf{x})\}}$, o equivalentemente con $1 + \exp\{-y\left(w_0 + \mathbf{w}^T\mathbf{x}\right)\}$. Por tanto, el modelo debe penalizarlo y, en particular, la pérdida logística (1.51) penaliza el logaritmo de este término (puesto que el logaritmo es una función creciente), es decir, habrá que minimizarla para ajustar el modelo.

Hemos justificado así la expresión (1.51) y, en particular, notemos que se corresponde con (1.61) cuando $n = 1$ e $y \in \{-1, 1\}$. Su convexidad ya se demostró mediante la Proposición 1.1.

Finalmente, destaquemos que métodos lineales de clasificación cuyos límites de decisión son hiperplanos se generalizan en el caso en que las clases no sean separables mediante las técnicas de Máquinas de Vectores de Soporte (SVM) [35] (*Capítulo 12*).

Capítulo 2

Redes neuronales

Históricamente se han buscado reglas lineales con las que resolver problemas de aprendizaje automático, pero en general son complicadas y muestran tendencia al sobreajuste. El motivo habitual es que los conjuntos de datos no están bien separados linealmente, al menos en el espacio original de los atributos. Por tanto, es conveniente buscar una transformación que mejore la separación lineal. En este contexto se sitúa la aparición del *aprendizaje profundo*, capaz de deducir la correcta representación a partir de los datos de entrada. Esta tarea es facilitada por el hecho de añadir más *capas* al modelo de aprendizaje, confiriendo la *profundidad* que da nombre a este procedimiento, cuyo origen se sitúa en 1940.

El nombre de *red neuronal* viene por analogía con la estructura de conexiones entre las células neuronales, pero más allá de esta interpretación, constituyen un campo especialmente relacionado con la Matemática Aplicada (Álgebra Lineal, Probabilidad, Teoría de la Información y Optimización Numérica). Además, su popularidad y utilidad han crecido en gran medida gracias, principalmente, a la mejora en la potencia de los ordenadores actuales, el aumento de los conjuntos de datos y la investigación en técnicas para entrenar redes aún más profundas. En el futuro, por tanto, se esperan nuevos desafíos y oportunidades para seguir mejorando.

El objetivo de este capítulo es introducir el concepto de red neuronal, el elemento básico del aprendizaje profundo, de forma que se sentarán definitivamente las bases para el posterior estudio de las Redes Generativas Adversariales que dan título al trabajo. La organización es la siguiente:

- Descripción de las *redes neuronales profundas pre-alimentadas*, también conocidas como *perceptrón multicapa (MLP)* o *feedforward neural network (FFNN)*: se trata de la arquitectura más simple. Se estudia su estructura por capas, que están totalmente interconectadas (*capas densas*) y se introduce el concepto de *función de activación* como herramienta para tratar problemas no lineales.
  Además, se relaciona su funcionamiento con el problema expuesto al estudiar el aprendizaje supervisado, lo cual lleva, en este caso de mayor complejidad en el que se pierde la convexidad, a plantear el algoritmo de *back-propagation* para computar eficientemente los gradientes. Se trata de un método iterativo general basado en aplicar la regla de la cadena de forma ordenada que particularizaremos al caso de las redes neuronales, concienciando de su tendencia al sobreajuste y de la existencia de técnicas de regularización y optimización concretas.

- Existen otros tipos de redes en los cuales no todas las capas son densas, lo cual aumenta la eficiencia a nivel computacional, que se emplean para tareas específicas. Motivados por el hecho de que nuestro interés es implementar una GAN que genere imágenes falsas, nos centraremos en estudio de las *redes neuronales convolucionales (CNNs)*, que son de especial utilidad en tareas de visión computacional. Detallaremos el funcionamiento de las características *capas convolucionales* mediante variantes de la operación matemática de *convolución* y, en líneas generales, cómo se lleva a cabo en estas la computación de los gradientes mediante el ya descrito algoritmo de back-propagation.

Como referencias principales se emplean [68] y [30], seleccionando la información necesaria y tratando de exponerla convenientemente para poder utilizarla después a la hora de estudiar las GANs. Aparte, con este mismo objetivo, añadiremos explicaciones relevantes que pueden encontrarse en [35, 52, 69, 72].

2.1. Redes neuronales prealimentadas (FFNNs)

El aprendizaje profundo se trata de aprendizaje automático con redes neuronales profundas, las cuales están teniendo una gran y exitosa aplicación en la actualidad. En esencia, una *red neuronal* es una clase de funciones que se obtienen componiendo transformaciones no lineales sobre combinaciones lineales de atributos en la capa anterior.

El objetivo de esta sección, utilizando principalmente como referencia [30] (*Capítulo 6*), es explicar cómo funcionan, en particular, las *redes neuronales prealimentadas*. De este tipo son todas aquellas redes que se basan en realizar predicciones a partir de los atributos, sin ningún tipo de conexión retroalimentada (o "hacia atrás") entre capas, como sí que ocurre, en cambio, en las *redes neuronales recurrentes* [30] (*Capítulo 10*), que dejaremos fuera de nuestro estudio.

El método de aprendizaje que llevan a cabo se puede entender bien a partir del grafo que se muestra en la figura 2.1. Además, daremos una descripción formal a nivel matemático con el fin de exponer el proceso de aprendizaje supervisado explicado en el capítulo anterior, concretando al caso en que se empleen redes neuronales para llevarlo a cabo. La principal diferencia es que tendremos que usar el algoritmo de *back-propagation*, que puede considerarse una particularización del Descenso de Gradiente basada en aplicar ordenadamente la regla de la cadena. El resto del desarrollo es, en esencia, muy similar y trataremos de mostrarlo.

2.1.1. Arquitectura

En primer lugar analizaremos su arquitectura, es decir, su estructura a nivel de capas que permitirá comprender correctamente la formulación matemática que expondremos a continuación.

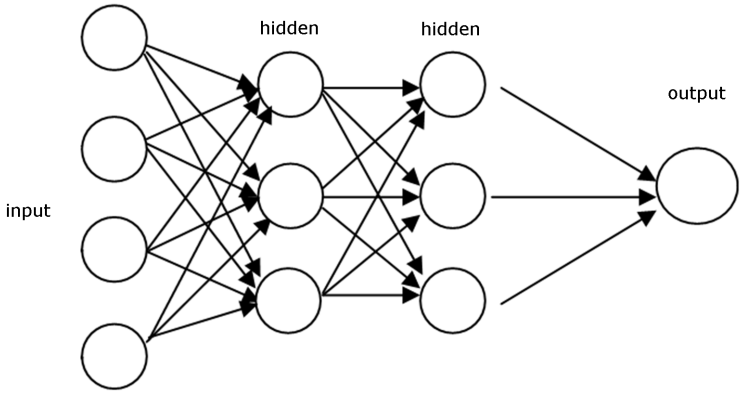

Figura 2.1: Red neuronal prealimentada. Fuente [63].

Tal y como podemos observar en la figura 2.1, están constituidas por L capas, a su vez formadas por A *nodos*. El número de nodos por capa se denomina *anchura*, A. Por su parte, si $L = 1$ se dice que se trata de una *red neuronal no profunda*, mientras que si $L > 1$ se denomina *red neuronal profunda*. En este último caso, existen capas intermedias que se denominan *capas ocultas*. La profundidad es importante para lograr buenas predicciones ya que, si tuviéramos una conexión directa entre los datos de entrada y los de salida, la contribución de cada entrada al valor de salida sería independiente de la del resto de entradas. Sin embargo, en la realidad, las variables de entrada tienden a ser altamente interdependientes y afectan a la salida de forma combinada y compleja. Las capas ocultas nos ayudan a capturar estas interacciones, pues permiten que la red neuronal se capaz de aprender sobre varios niveles de abstracción de los datos.

Cada capa está completamente conectada a la siguiente, es decir, sus nodos están conectados entre sí; por este motivo por se denominan *capas densas*. En esta conexión, para determinar la cantidad de información de cada nodo que se transmite a los nodos de la siguiente, se multiplica por unas cantidades seleccionadas denominadas *pesos*, $\mathbf{w}$. Es decir, el flujo de información va de la primera capa o *capa de*

entrada (los atributos, $\mathbf{x}$) a la última, *capa de salida* (la predicción, y), pasando por todas las capas ocultas. En las redes neuronales prealimentadas (*feedforward neural network*), como su propio nombre indica, el flujo de información va siempre en esta dirección, es decir, avanza hacia delante.

Además de los pesos, existe una un parámetro adicional modificable denominado *sesgo* o *bias*, b, que se utiliza para ajustar la salida, la cual se se calcula multiplicando los datos de entrada por los pesos correspondientes y sumando este término. Esto es lo que ocurre en un modelo simple lineal, pero la mayoría aplica en las capas ocultas una función no lineal llamada *función de activación*, σ, que permite a la red neuronal aprender relaciones no lineales entre los datos de entrada y los de salida, aumentando su capacidad y flexibilidad. Además, puesto que son diferenciables en general, permiten que el entrenamiento de la red neuronal se pueda realizar exitosamente mediante el algoritmo de back-propagation que se presentará posteriormente.

Una vez entendida la estructura de la red neuronal, pasemos a formularla matemáticamente. Denotemos a los j nodos de la capa i por $o_{i,j}$, , con $j = 1, ..., A^{(i)}$, donde $A^{(i)}$ es el número de nodos que hay en la capa i, $i = 1, ..., L-1$. Estos valores se engloban en el vector columna $\boldsymbol{o}_i = (o_{i1}, o_{i2}, \dots, o_{ij})^\top \in \mathbb{R}^j$. Para $i = 0$ tenemos los datos de entrada, $\boldsymbol{o}_0 = \mathbf{x} \in \mathbb{R}^p$ (entonces, $A^{(0)} = p$) y para $i = L$, el de salida (la predicción de la etiqueta), $o_L = \hat{y} \in \mathbb{R}$ (en este caso, $A^{(L)} = 1$). Cabe mencionar que la salida no tiene por qué estar compuesta por un solo nodo. No obstante, sin pérdida de generalidad, trataremos este caso por simplicidad.

Cada nodo j de la capa i está conectado al nodo k de la capa siguiente $i+1$ (con $k = 1, \dots, A^{(i+1)}$) mediante un valor conocido como peso, que podemos denotar como $w_{j,k}^{i,i+1}$ y, por simplicidad, $w_{j,k}$. Es decir, podemos englobar en el vector fila $\mathbf{w}_k = (w_{1k}, w_{2k}, \dots, w_{jk}) \in \mathbb{R}^k$ los pesos que conectan el nodo k con todos los nodos de la capa anterior. Ahora es fácil construir la matriz de pesos $\mathbf{W}^{(i,i+1)} \in \mathcal{M}_{k\times j}(\mathbb{R})$ tomando como filas dichos vectores:

$$\mathbf{W}^{(i,i+1)} = \begin{pmatrix} w_{11} & w_{21} & \cdots & w_{j1} \\ w_{12} & w_{22} & \cdots & w_{j2} \\ \vdots & \vdots & \ddots & \vdots \\ w_{1k} & w_{2k} & \cdots & w_{jk} \end{pmatrix}. \tag{2.1}$$

En cuanto al sesgo, existe un valor $b_k \in \mathbb{R}^k$ para todo k en cada paso de la capa i a la capa $i+1$; todos ellos se engloban en el vector columna $\mathbf{b}^{(i,i+1)} = (b_1, \dots, b_k)^\top$. Respecto a la función de activación, se elige normalmente la misma en todas las capas ocultas y, puede ser, que en la $L-1$ se emplee otra diferente en función del tipo de predicción que haga el problema al que nos estamos enfrentando. No obstante, supondremos, sin pérdida de generalidad y para simplificar la notación, que se utiliza la misma, σ.

Teniendo esto en cuenta, la operación matemática que una red neuronal prealimentada lleva a cabo para calcular los nodos de cada capa a partir de los anteriores es:

$$o_{i+1,k} = \sigma(\mathbf{w}_k \cdot \boldsymbol{o}_i + b_k), \quad i = 0, \dots, L-1. \tag{2.2}$$

Y, en notación vectorial, el vector que contiene los nodos de la capa siguiente es:

$$\boldsymbol{o}_{i+1} = \sigma(\mathbf{W}^{(i,i+1)}\boldsymbol{o}_i + \mathbf{b}^{(i,i+1)}) \quad i = 0, \dots, L-1. \tag{2.3}$$

(Se trata de aplicar la función de activación σ componente a componente al vector resultante)

Entonces, para ver más clara la analogía con lo expuesto en la sección 1.2, de acuerdo con (2.3), podemos considerar que a partir de unos datos de entrada $\mathbf{x} \in \mathcal{X} = \mathbb{R}^p$ se obtiene $\hat{y}$, que trata de predecir la etiqueta real que le corresponde $y \in \mathcal{Y}$. Para ello, se utilizan predictores de la siguiente forma:

$$h(\mathbf{x}, \mathbf{W}^{(i,i+1)}, \mathbf{b}^{(i,i+1)}) \in \mathcal{H}_{NN}(L, A, \sigma), \quad i = 0, \dots, L-1,$$

donde $\mathcal{H}_{NN}(L, A, \sigma)$ denota la clase de predictores de la red neuronal, es decir, las funciones de la red neuronal; es, por tanto, su propia arquitectura la que determina la clase de hipótesis. Esta depende de la profundidad y de la anchura, ya que cada h lo hace de los pesos y los sesgos existentes en cada paso

de una capa a la siguiente. Por la misma razón, depende de la función de activación que se elija, con lo cual, los hiperparámetros en el caso de modelos de aprendizaje implementados mediante redes neuronales son L, A y σ, y los parámetros que deben actualizarse para minimizar el riesgo empírico $\boldsymbol{\theta}$, son $\mathbf{W}$ y $\mathbf{b}$.

El problema de minimización del riesgo empírico enunciado en (1.56), cuando consideramos un conjunto de entrenamiento $S = \{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots, (\mathbf{x}_n, y_n)\}$, donde $(\mathbf{x}_i, y_i) \sim D$, $D \in \mathcal{P}(\mathcal{X} \times \mathcal{Y})$, como hicimos en ese momento, se escribe ahora así:

$$\hat{h} = \underset{\mathbf{w},\mathbf{b}\in\mathcal{H}_{NN}}{\operatorname{argmin}} \frac{1}{n} \sum_{j=0}^{n} \ell\left((\mathbf{x}_j, y_j), h(\mathbf{x}_j, \mathbf{W}^{(i,i+1)}, \mathbf{b}^{(i,i+1)})\right), \quad i = 0, \ldots, L-1. \tag{2.4}$$

Como sabemos, se debe elegir la función de pérdida ℓ convenientemente. No obstante, las redes neuronales a causa de los pesos y las funciones de activación asignadas a cada capa estropean la convexidad de (2.4) en la mayoría de los casos, haciéndolo difícil de resolver. Por tanto, en aprendizaje profundo no se resuelve ERM como habíamos visto. En ese caso, la idea sería aplicar SGD (Algoritmo 2) para (2.4) y se pararía sin llegar al final, porque no se conoce el mínimo.

En definitiva, en aprendizaje profundo no se deja iterar SGD hasta la convergencia por varias razones: computacionalmente es muy costoso, no hay garantías de alcanzar el mínimo global y probablemente habría sobreajuste. Alternativamente, se busca reducir suficientemente el valor de la función de pérdida para obtener un error de generalización aceptable. Para llevarlo a cabo, los gradientes que hay que calcular son muy complejos y con el fin de computarlos se emplea el algoritmo de *propagación inversa* o *back-propagation*.

En aprendizaje profundo, también cabe señalar que la función de pérdida se denomina de forma más general *función objetivo*, ya que en algunos problemas se busca maximizarlas, como veremos que ocurre en las GANs para el caso de los discriminadores. En la literatura se puede encontrar que estos términos se diferencian (junto con el de *función de coste*), pero en este trabajo los consideraremos sinónimos, tal y como se hace en [30] (*Capítulo 4, sección 4.3*).

Finalmente, expongamos con algo más de detalle los tipos funciones de activación σ que hay, pues como se ha explicado, juegan un papel fundamental en el aprendizaje de la red neuronal permitiéndole hacer frente a problemas no lineales. Las más utilizadas son, tal y como se muestra en [72]:

- *Función sigmoide o logística*: la principal razón por la que se utiliza es porque su imagen está entre 0 y 1. Por tanto, es especialmente útil para predecir probabilidades. Es diferenciable y monótona, pero su derivada no es monótona. Se emplea en tareas de regresión logística binaria, nótese su similitud con las expresiones (1.57).
$$\sigma(x) = \frac{1}{1 + e^{-x}}. \tag{2.5}$$

- *Función softmax*: es una generalización de la anterior a múltiples dimensiones, es decir, se emplea en tareas de clasificación multiclase mediante regresión logística. Por este motivo, se aprecia la similitud con las expresiones (1.59).
$$\begin{aligned} &\sigma : \mathbb{R}^K \to [0,1]^K \\ &\sigma(\mathbf{x})_j = \frac{e^{x_j}}{\sum_{k=1}^{K} e^{x_k}} \quad \text{para } j = 1, \ldots, K. \end{aligned} \tag{2.6}$$

- *Función tangente hiperbólica* o *tanh*: es similar a la función sigmoide pero funciona mejor. Su imagen va de -1 a 1, es diferenciable y monótona; su derivada no es monótona. La ventaja que presenta es que las entradas negativas tienen imágenes negativas y la nulas, muy cercanas al cero. Se utiliza principalmente en clasificación binaria.
$$\sigma(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}. \tag{2.7}$$

- *Unidad Lineal Rectificada* o *ReLu*: su imagen va de 0 a infinito. Tanto la función como su derivada son monótonas. El problema que plantea es que todas las entradas negativas tienen imagen nula,

lo que hace que estas entradas no se representen adecuadamente.

$$\sigma(x) = \begin{cases} 0 & \text{si} \quad x < 0 \\ x & \text{si} \quad x \geq 0. \end{cases} \tag{2.8}$$

Existen algunas alternativas para solucionar ese problema y son las que se muestran a continuación:

- *Leaky ReLu* (si $\alpha = 0.01$) o *ReLu paramétrica* (si $\alpha \neq 0.01$):

$$\sigma(\alpha, x) = \begin{cases} \alpha x, & \text{si } x < 0 \\ x, & \text{si } x \geq 0. \end{cases} \tag{2.9}$$

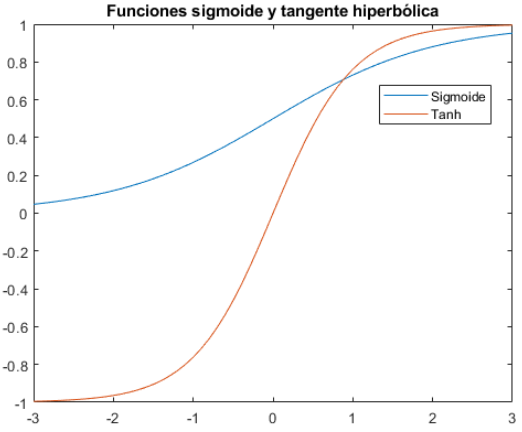

Figura 2.2: Función sigmoide y tangente hiperbólica.

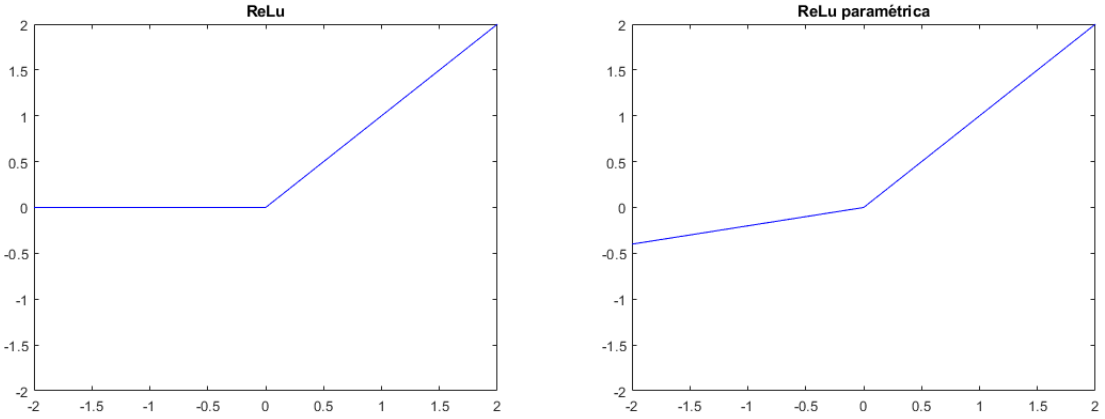

Figura 2.3: Función ReLu y ReLu paramétrica.

2.1.2. Back-propagation

El objetivo del algoritmo de propagación inversa es encontrar un conjunto de pesos y sesgos óptimo, es decir, que minimice en lo posible la función de pérdida. Fue propuesto originalmente por Rumelhart en 1986 [65]. Cabe mencionar que, aunque principalmente se emplee para la función de pérdida de las redes neuronales prealimentadas, este algoritmo sirve para computar gradientes de cualquier función, ya que se fundamenta básicamente en la regla de la cadena. A continuación lo veremos formalmente y, para ello, nos basaremos en la referencia [30] (*Capítulo 6*, págs. 200-217).

Para describirlo y entenderlo correctamente es útil representar las redes neuronales como *grafos computacionales*, un forma de representar una función matemática en el lenguaje de la teoría de grafos [61]. Cada nodo del grafo está conectado al siguiente mediante ejes que representan una determinada función. De esta forma, cada nodo se calcula a partir de los que estén conectados a él mediante dicha función. Veamos el siguiente sencillo ejemplo en la figura 2.4: se trata de usar la misma función $f : \mathbb{R} \to \mathbb{R}$ de forma que $x = f(w), y = f(x), z = f(y)$.

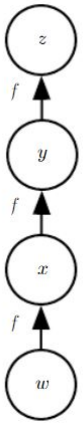

Figura 2.4: Ejemplo de grafo computacional. Fuente: [30] (*Capítulo 6*, pág. 205).

Para computar $\frac{\partial z}{\partial w}$ se aplica la regla de la cadena y se llega a que:

$$\begin{aligned}&\frac{\partial z}{\partial w}\\ =&\frac{\partial z}{\partial y}\frac{\partial y}{\partial x}\frac{\partial x}{\partial w}\\ =&f'(y)f'(x)f'(w)\\ =&f'(f(f(w)))f'(f(w))f'(w).\end{aligned}$$

La última igualdad nos da una implementación válida del algoritmo de back-propagation. No obstante, la segunda igualdad también sirve si la memoria que ocupan los valores de esas expresiones es baja y, de hecho, es preferible porque el tiempo de cómputo es menor.

Nótese que lo que se hace es computar las derivadas en orden inverso al grafo. Si hubiera más nodos "padres", habría que sumar los resultados obtenidos al aplicar la función correspondiente y esto se reflejaría trivialmente en el cálculo de la derivada, teniendo en cuenta que la derivada de una suma es la suma de las derivadas. Veremos este caso a continuación, pues vamos a estudiar un grafo más general.

Consideremos un grafo computacional que describe cómo computar un escalar $u^{(n)}$ (podría ser la función de pérdida de un determinado conjunto de entrenamiento). Queremos obtener el gradiente de este escalar, con respecto a los n_i nodos $u^{(1)}$ a $u^{(n_i)}$ (que podrían ser los parámetros del modelo). En otras palabras, queremos computar $\frac{\partial u^{(n)}}{\partial u^{(i)}}$ para todo $i \in \{1, 2, ..., n_i\}$.

Cada nodo $u^{(i)}$ está asociado con una operación $f^{(i)}$ (puede suponerse la función de activación) y se computa evaluando la función en $\mathbb{A}^{(i)}$, el conjunto de todos los nodos "padres"de $u^{(i)}$ (es decir, de los que proviene y están por debajo en el grafo), que se denota por $Pa\left(u^{(i)}\right)$:

$$u^{(i)} = f\left(\mathbb{A}^{(i)}\right). \tag{2.10}$$

Representemos la propagación hacia delante mediante un grafo $\mathcal{G}$ y, hacia atrás, mediante otro grafo $\mathcal{B}$, con un nodo por cada nodo de $\mathcal{G}$. La computación en $\mathcal{B}$ va en orden contrario al de la computación en $\mathcal{G}$, y cada nodo de $\mathcal{B}$ computa la derivada $\frac{\partial u^{(n)}}{\partial u^{(i)}}$ asociada con el nodo $u^{(i)} \in \mathcal{G}$. Esto se hace aplicando la regla de la cadena con respecto al escalar de salida $u^{(n)}$ tal que así:

$$\frac{\partial u^{(n)}}{\partial u^{(j)}} = \sum_{i:j\in Pa\left(u^{(i)}\right)} \frac{\partial u^{(n)}}{\partial u^{(i)}} \frac{\partial u^{(i)}}{\partial u^{(j)}}. \tag{2.11}$$

$\mathcal{B}$ contiene exactamente un eje por cada eje del nodo $u^{(j)}$ al nodo $u^{(i)}$ de $\mathcal{G}$. El eje de $u^{(j)}$ a $u^{(i)}$ está asociado con la computación de $\frac{\partial u^{(i)}}{\partial u^{(j)}}$. Además, en cada nodo se lleva a cabo un producto escalar, entre el gradiente ya computado con respecto a los nodos $u^{(i)}$ que son hijos de $u^{(j)}$ y el vector que contiene las derivadas parciales $\frac{\partial u^{(i)}}{\partial u^{(j)}}$ para los mismos nodos hijos $u^{(i)}$.

Tras esta previa introducción para entender que el algoritmo de back-propagation se fundamenta en aplicar de manera ordenada la regla de la cadena, pasemos ahora a exponer cómo se aplica en el caso de una red neuronal prealimentada, pudiendo considerarse una particularización del algoritmo de Descenso de Gradiente.

En primer lugar, la red lleva a cabo lo que se conoce como *propagación "hacia delante"*: tal y como se ha explicado anteriormente, los datos de entrada $\mathbf{x}$ se propagan a través de las capas ocultas hasta producir finalmente $\hat{y}$. Una vez se llega a este punto, se calcula la función de pérdida ℓ según los parámetros correspondientes, en nuestro caso, pesos y sesgos de cada capa.

Por otro lado, en la *propagación "hacia atrás"* (o back-propagation) la información fluye en sentido contrario con el objetivo de computar el gradiente. Suponiendo que la red es un grafo, se procede tal y como se pudo ver en los ejemplos iniciales.

Veamos ambos algoritmos. Se empleará, por simplicidad, solo un ejemplo de entrenamiento. En la práctica, como ya sabemos del capítulo anterior, se suele emplear un minilote en cada iteración. No obstante, sin pérdida de generalidad a la hora de comprender los algoritmos y por simplicidad en la notación y en la exposición, lo omitiremos en lo que sigue.

Algoritmo 3: Propagación "hacia delante": se avanza hacia delante a través de todas las capas de la red neuronal tal y como se indica en (2.3). El objetivo es predecir la etiqueta real $y \in \mathcal{Y}$ correspondiente a los datos de entrada $\mathbf{x} \in \mathcal{X} = \mathbb{R}^p$. La predicción obtenida es $\hat{y}$; una vez computada, se puede calcular la función de pérdida $\ell((\mathbf{x}, y), \hat{y})$.

Resultado: $\hat{\boldsymbol{y}} = \boldsymbol{o}^{(L)}$;
$\ell = \ell((\mathbf{x}, y), \hat{y})$;
L la profundidad de la red neuronal prealimentada.;
$\mathbf{W}^{(i,i+1)}, \quad i \in \{0, \dots, L-1\}$ las matrices de pesos;
$\mathbf{b}^{(i,i+1)}, \quad i \in \{0, \dots, L-1\}$ los sesgos;
$\mathbf{x}$ los datos de entrada;
y la etiqueta real;
para $k = 1, \dots, L$ **hacer**
  $\boldsymbol{a}^{(k)} = \boldsymbol{W}^{(k-1,k)}\boldsymbol{o}^{(k-1)} + \boldsymbol{b}^{(k-1,k)}$;
  $\boldsymbol{o}^{(k)} = \sigma\left(\boldsymbol{a}^{(k)}\right)$;
fin

Antes de exponer el algoritmo de propagación "hacia atrás", es necesario hacer un inciso de notación, ya que mediante $\odot$ representaremos el producto de Hadamard. Se trata de una operación binaria que toma dos matrices de las mismas dimensiones y produce otra matriz de la misma dimensión, donde cada elemento i, j es el producto de los elementos i, j de las dos matrices originales. Es decir, $(A \odot B)_{ij} = (A)_{ij}(B)_{ij}$. Utilizaremos esto para computar los gradientes tal y como se muestra a continuación.

Algoritmo 4: Propagación "hacia atrás" o back-propagation: se computan los gradientes comenzando por la capa de salida y avanzando hacia atrás por las capas ocultas hasta la de entrada. Los gradientes respecto a los pesos y sesgos se pueden interpretar como una indicación de cómo la salida de cada capa debe variar para reducir el error. La actualización de estos parámetros se hará posteriormente mediante el algoritmo de Descenso de Gradiente Estocástico (SGD).

Tras la computación hacia delante, se calcula el gradiente en la capa de salida:

$$\boldsymbol{g} \leftarrow \nabla_{\hat{y}} \ell = \nabla_{\hat{y}} \ell((\mathbf{x}, y), \hat{y})$$

para $k = L, L-1, \ldots, 1$ **hacer**
  Se convierte este gradiente en otro previo a que actúe la función de activación;
  $\boldsymbol{g} \leftarrow \nabla_{\boldsymbol{a}^{(k)}} \ell = \boldsymbol{g} \odot \sigma' \left(\boldsymbol{a}^{(k)}\right)$;
  $\boldsymbol{o}^{(k)} = \sigma \left(\boldsymbol{a}^{(k)}\right)$;
  Se computan los gradientes respecto de los pesos y los sesgos;
  $\nabla_{\boldsymbol{b}^{(k-1,k)}} \ell = \boldsymbol{g}$;
  $\nabla_{\boldsymbol{W}^{(k-1,k)}} \ell = \boldsymbol{g}\boldsymbol{o}^{(k-1)\top}$;
  Se propagan los gradientes con respecto a la siguiente capa inferior:;
  $\boldsymbol{g} \leftarrow \nabla_{\boldsymbol{o}^{(k-1)}} \ell = \boldsymbol{W}^{(k-1,k)\top} \boldsymbol{g}$;
fin

Una vez se tienen los gradientes computados a partir del algoritmo de back-propagation, estos deben utilizarse para actualizar los pesos y sesgos en cada capa de forma que se minimice en lo posible la función de pérdida. Esto se lleva a cabo mediante el algoritmo SGD (Algoritmo 2), teniendo en cuenta que en este caso el conjunto total de parámetros $\boldsymbol{\theta}$ está compuesto por las matrices de pesos $\mathbf{W}$ y los sesgos $\mathbf{b}$. En este caso, por tanto, son los parámetros que hay que buscar de manera que se optimice el error, en analogía con lo expuesto en el capítulo anterior.

Precisamente por lo expuesto en el capítulo anterior, sabemos que la optimización es una tarea complicada y aún más en el caso de las redes neuronales, donde interviene una cantidad ingente de parámetros y de hiperparámetros que provoca la habitual tendencia al sobreajuste. Puesto que actualmente no existen resultados teóricos que garanticen el buen funcionamiento de un determinado algoritmo de optimización y este debe elegirse mediante prueba y error, se han propuesto ciertas mejoras del algoritmo SGD ya mencionadas en su momento, así como otras técnicas de optimización y regularización, aparte de la interrupción anticipada, también expuesta con anterioridad. Como ejemplo mencionaremos las siguientes, pues nos serán de utilidad a la hora de llevar a cabo la implementación práctica de una GAN como parte final del trabajo:

- Añadir un *parámetro de penalización de la norma* que dependa de los pesos y los sesgos a la función de pérdida: $\ell = \ell + \alpha\Omega$, donde $\alpha \in [0, \infty)$ es un parámetro que debe ajustarse convenientemente para seleccionar el grado de regularización. Se puede tomar $\Omega(\mathbf{w}) = \frac{1}{2}\|\mathbf{w}\|_2^2$ o $\Omega(\mathbf{w}) = \|\mathbf{w}\|_1$.
- *Abandono* o (*dropout*) [71]: la idea clave es omitir aleatoriamente capas de la red neuronal durante el proceso de entrenamiento, lo que evita que acaben memorizando las predicciones.
- *Normalización* de los datos de entrada [20], de manera que se tengan en el rango $[0, 1]$ o $[-1, 1]$, según convenga. Esto es muy común en tareas de visión computacional, porque los píxeles varían en un amplio rango. Como resultado, se evitan los gradientes que explotan y se mejora la velocidad de convergencia.
- En la línea de la idea anterior, se introdujo la *normalización en lotes* [36], consistente en normalizar los minilotes en cada iteración.

Más información sobre estrategias de regularización y optimización en el caso de redes neuronales se puede consultar en [30] (*Capítulo 7* y *Capítulo 8*, respectivamente), así como [3].

Finalmente, aclaremos que la descripción del algoritmo de back-propagation que hemos dado es más simple que las que las implementaciones prácticas llevan a cabo. Por ejemplo, no hemos descrito cómo controlar el consumo de memoria ni tampoco cómo lograr que detecte cuándo un gradiente no está bien definido. Estos y otros inconvenientes que se deben de tener en cuenta en la práctica se escapan de los objetivos de este trabajo, pero se pueden consultar en [2]. Nos resulta suficiente ser conscientes de ellos y centrarnos en su comprensión teórica. A la hora de utilizarlo en la correspondiente parte de implementación, utilizaremos programas que ya lo incorporan.

2.2. Redes neuronales convolucionales (CNNs)

Muchas arquitecturas de redes neuronales han sido desarrolladas para llevar a cabo tareas específicas. Por ejemplo, las *redes neuronales convolucionales (CNN)*, introducidas por primera vez por Yann LeCun en 1989 [18], están especializadas en visión computacional. Por su parte, las redes prealimentadas que acabamos de describir se pueden generalizar, como ya comentamos, a redes neuronales recurrentes (RNN), basadas en el trabajo de Rumelhart en 1986 [65], cuya principal aplicación es la de procesar datos secuenciales con una relación temporal, como es el reconocimiento de voz [32], el procesamiento de lenguaje natural [73] o la composición de música [22], entre muchos otros. No obstante, nos centraremos solamente en las redes neuronales convolucionales puesto que son las que ofrecen mejores resultados en la implementación práctica de las GANs a la hora de generar y discriminar imágenes, tal y como veremos finalmente.

De manera análoga al estudio de las redes neuronales prealimentadas, expondremos en primer lugar la arquitectura de las CNNs y la formularemos matemáticamente. El término "convolucional" proviene de la *operación convolución*. Recordaremos cómo se define dicha operación y veremos cómo la aplican estas redes en particular, ya que no se corresponde exactamente con la definición formal. Finalmente, veremos en líneas generales cómo se lleva a cabo el algoritmo de back-propagation en estas capas convolucionales. Emplearemos como referencia [30] (*Capítulo 9*).

2.2.1. Arquitectura

Hasta ahora hemos descrito las redes neuronales como cadenas de capas densas, es decir, totalmente conectadas, y hemos tenido en consideración la profundidad de la red y la anchura de cada capa. Sin embargo, las redes convolucionales establecen menos conexiones, es decir, sus capas no son densas sino que están conectadas entre sí solamente en pequeñas unidades. Esto tiene como consecuencia que se reduce notablemente el número de parámetros y, por tanto, el costo computacional.

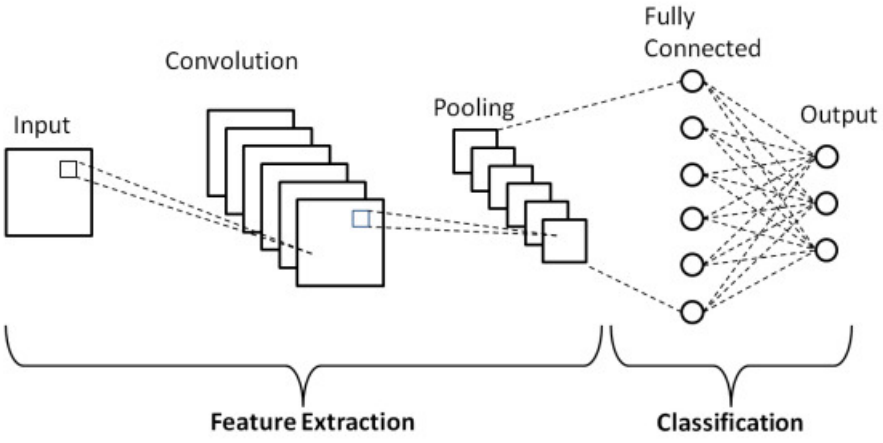

Figura 2.5: Red neuronal convolucional. Fuente [59].

En definitiva, se caracterizan por la *escasez de interacciones* en comparación con las redes tradicionales, lo cual se logra eligiendo núcleos de menores dimensiones que la entrada. Por ejemplo, al procesar

una imagen, como dato de entrada puede que tenga miles o millones de píxeles, pero se pueden detectar pequeños (aunque significantes) atributos que ocupan solamente decenas o cientos de píxeles. Por tanto, se reducen notablemente tanto el número de operaciones necesarias para computar la salida como el requerimiento de memoria.

Por otra parte, otra mejora que presentan estas nuevas redes es que *comparten parámetros*, es decir, utilizan el mismo parámetro para más de una función en un modelo. En cambio, en una red tradicional, cada elemento de la matriz de pesos se utiliza solamente una vez. En las redes convolucionales, cada elemento del núcleo se emplea en cada posición de la entrada (excepto quizá algunos píxeles de los bordes). En consecuencia, se ajusta en lugar de un conjunto de parámetros para cada paso de una capa a otra, un solo conjunto para toda la red. Esto no afecta al tiempo de cómputo de la propagación "hacia delante", pero sí reduce enormemente el almacenamiento requerido.

Una CNN típica está compuesta por tres tipos de capas, tal y como podemos ver en la figura 2.5: capas convolucionales, capas de reducción (o *pooling*) y capas densas. Los datos de entrada serán, en el caso de imágenes, los píxeles de esta, de los que pasarán a la capa convolucional aquellos que se encuentren la región que esté conectada. En este momento se suele aplicar también la función de activación, normalmente ReLu (2.8). A continuación, se encuentran las capas de reducción que llevan a cabo, como su propio nombre indica, la reducción de las dimensiones espaciales de la entrada. Tras estas operaciones se extraen los principales atributos y, finalmente, en las capas densas, siguiendo las operaciones ya explicadas en la sección anterior, se realizan las predicciones y las consecuentes clasificaciones.

En definitiva, las CNNs son capaces de transformar las entradas, generalmente imágenes, capa a capa utilizando convoluciones y técnicas de reducción para llevar a cabo predicciones que serán útiles en tareas de clasificación o de regresión.

2.2.2. Operación convolución

Estudiemos ahora con detalle las operaciones que se llevan a cabo en las características capas de convolución, lo cual también nos permitirá entender mejor el funcionamiento de las capas de reducción.

Supongamos que x y w son funciones medibles en $\mathbb{R}^p$. Su producto de convolución en un punto $\mathbf{t} \in \mathbb{R}^p$, denotado por $s(\mathbf{t}) = (x \star w)(\mathbf{t})$, es:

$$s(\mathbf{t}) = \int x(\mathbf{a}) w(\mathbf{t} - \mathbf{a}) d\mathbf{a}. \tag{2.12}$$

En el contexto de las CNNs, x serían los datos de entrada, w el *núcleo* (o *kernel*), también conocido como *filtro*, y $s(\mathbf{t})$ la salida.

No obstante, esto no es realista. En la mayoría de las aplicaciones el conjunto de datos de entrada es un tensor (también conocido como array multidimensional en computación) y el núcleo, otro repleto de parámetros que deben ajustarse convenientemente, lo cual es el objetivo de las redes convolucionales. Por ejemplo, si llevamos a cabo una convolución en 2 dimensiones, tal y como será nuestro caso usando imágenes como datos de entrada, la salida se puede escribir como:

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

donde hemos utilizado la conmutatividad de la convolución y S denota la salida, K el núcleo e I la entrada. El papel del núcleo se puede apreciar claramente en la figura 2.6: se trata habitualmente de una matriz cuadrada $m \times m$ que, desplazándose por toda la matriz de entrada, extrae elementos de dicho tamaño y los multiplica por un valor determinado. La magnitud de este desplazamiento se controla, como veremos en lo que sigue, mediante un parámetro conocido como *paso* (o *stride*).

Nótese que la propiedad conmutativa que hemos aplicado en (2.13) proviene de *invertir* el núcleo con respecto a los datos de entrada, en el sentido de que si m aumenta, los índices de la entrada también,

pero los del núcleo, no. En lugar de realizar esto, muchas implementaciones llevan a cabo la operación convolución conocida como *correlación cruzada*:

$$S(i,j) = (K * I)(i,j) = \sum_m \sum_n I(i+m, j+n) K(m,n). \tag{2.14}$$

Distinguiremos (2.13) de (2.14) haciendo referencia a si el núcleo se ha invertido o no.

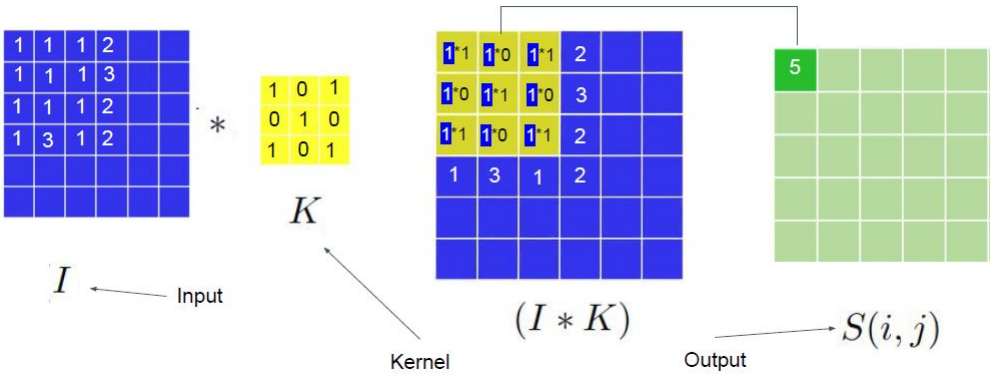

Figura 2.6: Operación convolución en 2D (2.13). Fuente [12].

En la práctica, las funciones empleadas por estas redes difieren ligeramente de la operación convolución tal y como se define formalmente. En primer lugar, cuando nos referimos a dicha operación en este contexto, hacemos referencia a la aplicación de varias convoluciones de forma paralela.

Además, los datos de entrada no son solamente un array de valores sino que, por ejemplo, una imagen a color alberga en cada píxel información sobre la intensidad de rojo, verde y azul. Esto se conoce como *canales* y el número de canales, como *profundidad* (una imagen a color tendrá 3 canales y en blanco y negro, 1). Por tanto, cuando se trabaja con imágenes, se suelen representar en tensores tridimensionales: una dimensión para los canales y las otras dos, espaciales. De hecho, puesto que en la práctica lo más eficiente es trabajar con minilotes, las implementaciones trabajan con tensores en cuatro dimensiones, tal y como veremos cuando llevemos a cabo nuestras particulares simulaciones. En esta cuarta dimensión, por simplicidad en la exposición del desarrollo teórico, la omitiremos, pero notemos que sirve para guardar las muestras del correspondiente minilote.

Supongamos que tenemos un tensor de cuatro dimensiones como núcleo, $\mathbf{K}$, cuyos elementos son $K_{i,j,k,l}$ que conecta una determinada unidad de salida en el canal j con una determinada unidad de entrada en el canal i de salida, con una diferencia de k filas y l columnas menos entre la salida y la entrada (la operación convolución así descrita reduce las dimensiones). Supongamos también que los datos de entrada son $\mathbf{V}$ con elementos $V_{i,j,k}$ referidos al valor de la unidad de entrada en el canal i en la fila j y la columna k. Análogamente, la salida la denotaremos por $\mathbf{Z}$, que será producida mediante la convolución de $\mathbf{K}$ y $\mathbf{V}$, sin invertir el núcleo:

$$Z_{i,j,k} = \sum_{l,m,n} V_{l,j+m-1,k+n-1} K_{i,l,m,n}, \tag{2.15}$$

donde los índices del sumatorio hacen referencia a aquellos valores de los tensores para los que la suma es posible.

Si queremos omitir ciertas posiciones del núcleo para reducir el costo computacional (en detrimento de la obtención de mejores atributos), es posible reducir la salida de forma que el núcleo se desplace s píxeles antes de realizar cada convolución.

$$Z_{i,j,k} = c(\mathbf{K}, \mathbf{V}, s)_{i,j,k} = \sum_{l,m,n} \left[V_{l,(j-1)\times s+m,(k-1)\times s+n} K_{i,l,m,n} \right], \tag{2.16}$$

donde c denota la operación convolución correspondiente. Además, denominaremos a s el *paso* (o *stride*), siendo también posible definir uno para cada dirección.

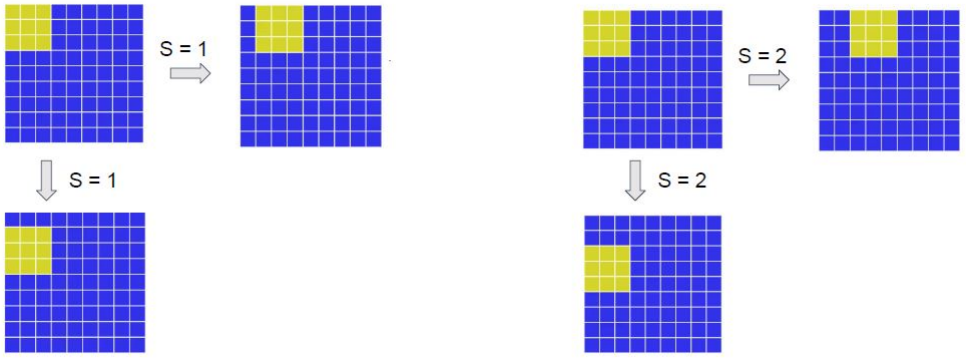

Figura 2.7: Paso o *stride*. Fuente [12].

Otra operación que llevan a cabo las redes neuronales convolucionales es el *relleno* (o *padding*), que se denota por p: se trata de rellenar los bordes del tensor de datos de entrada $\mathbf{V}$ habitualmente con ceros, con el fin de controlar la dimensión del núcleo y de la salida de forma independiente. Si no se lleva a cabo esta técnica, el núcleo solo es capaz de desplazarse ciertas posiciones en la imagen, en concreto, aquellas posiciones en las que esté por completo contenido en el interior de dicha imagen.

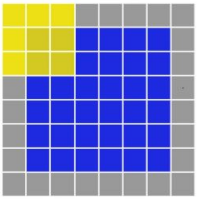

Figura 2.8: Relleno o *padding*, $p = 1$. Fuente [12].

Si tratamos con imágenes en dos dimensiones $W_{in} \times H_{in}$ (anchura y altura), entonces teniendo en cuenta las dimensiones del núcleo $k \times k$, el paso s y el relleno p, las dimensiones de la salida serán:

$$\begin{aligned} W_{out} &= \frac{W_{in} - k + 2p}{s} + 1 \\ H_{out} &= \frac{H_{in} - k + 2p}{s} + 1. \end{aligned} \tag{2.17}$$

Estamos ya en condiciones, por tanto, de comprender cómo funcionan las capas convolucionales. No obstante, también se ha explicado que son características las capas de reducción, que podemos entender ahora que sabemos lo que son los núcleos y los pasos. El tipo más común de reducción es el *max-pooling*, consistente en retener el valor máximo en cada desplazamiento en los datos de entrada de un núcleo de determinadas dimensiones, tal y como se observa en la figura 2.9.

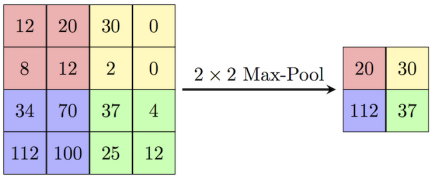

Figura 2.9: *Max-pool*, con núcleo 2×2 y $s = 2$. Fuente [26].

Tal y como acabamos de ver, la operación convolución que llevan a cabo las redes neuronales convolucionales se puede representar como un producto de matrices dispersas si transformamos el tensor de entrada en un vector mediante *aplanamiento* (o *flattening*), es decir, eliminando todas las dimensiones excepto una, como puede verse en la figura 2.10. Esta reinterpretación nos permite definir otra importante operación: la *convolución traspuesta*, consistente en multiplicar por la matriz traspuesta de la convolución. Como resultado de esta operación, al contrario que al realizar la convolución, se aumenta la dimensión de la salida. Esto veremos que será de especial importancia en las GANs a la hora de generar imágenes.

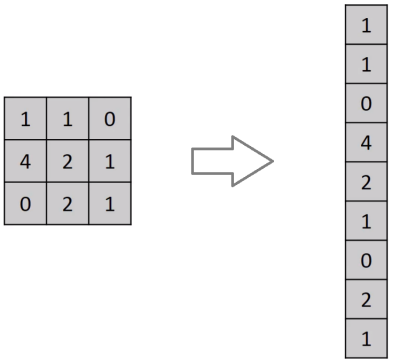

Figura 2.10: Aplanamiento o *flattening*. Fuente [12].

2.2.3. Back-propagation en CNNs

En al sección 2.1.2 se dio la descripción del algoritmo de back-propagation en general y, además, en particular para las capas densas. En el caso de las redes convolucionales la deducción de las ecuaciones se complica debido a que, además de las operaciones matriciales que se llevan a cabo en las capas densas, se introducen variantes de la operación convolución de expresión más complicada (2.16) en las capas caracterísiticas de estas redes. Un desarrollo detallado al respecto se puede consultar en [29]; con el fin de tener una idea general de cómo funciona dicho estudio, veámoslo para el caso de dos dimensiones y para una sola muestra de entrada, tal y como se hace en [30] (*Capítulo 9*).

Supongamos que queremos entrenar una red neuronal convolucional de núcleo $\mathbf{K}$ aplicado a una imagen $\mathbf{V}$ con paso s. Entonces, en la propagación "hacia delante" en las capas convolucionales se aplica la ecuación (2.16) para determinar la salida $\mathbf{Z}$ y se computa la función de pérdida $\ell(\mathbf{V}, \mathbf{K})$. A continuación, al comenzar la propagación "hacia atrás", se calcula el tensor $\mathbf{G}$ cuyas componentes son

$$G_{i,j,k} = \frac{\partial}{\partial Z_{i,j,k}} J(\mathbf{V}, \mathrm{K}). \tag{2.18}$$

Como sabemos, para entrenar la red debemos computar las derivadas con respecto a los núcleos, para lo cual se utiliza la función

$$g(\mathbf{G}, \mathbf{V}, s)_{i,j,k,l} = \frac{\partial}{\partial K_{i,j,k,l}} J(\mathbf{V}, \mathbf{K}) = \sum_{m,n} G_{i,m,n} V_{j,(m-1)\times s+k,(n-1)\times s+l}, \tag{2.19}$$

y, además, para propagar el gradiente será necesario computar el gradiente con respecto a $\mathbf{V}$, para lo cual se utiliza la función

$$\begin{aligned} h(\mathbf{K}, \mathbf{G}, s)_{i,j,k} &= \frac{\partial}{\partial V_{i,j,k}} J(\mathbf{V}, \mathbf{K}) \\ &= \sum_{l,m} \sum_{n,p} \sum_{q} K_{q,i,m,p} G_{q,l,n}, \end{aligned} \tag{2.20}$$

con l, m tales que $(l-1) \times s + m = j$ y n, p tales que $(n-1) \times s + p = k$.

Notemos que el procedimiento es análogo al descrito en los Algoritmos 3 y 4, con la única diferencia de la mayor complejidad que las convoluciones introducen en las expresiones correspondientes.

Capítulo 3

Redes Generativas Adversariales (GANs)

Los procedimientos de aprendizaje automático, tal y como se remarcó anteriormente, son de gran importancia en la actualidad y se realiza una importante labor investigadora sobre ellos. Como es ya sabido, las redes neuronales son un grupo de reglas de aprendizaje de los más utilizados, pues han conseguido un éxito enorme en tareas de clasificación (reconocimiento de dígitos manuscritos, detección de spam...). Sin embargo, en los años 90 se comenzó a ver su fragilidad frente a ataques consistentes en ligeras modificaciones que derivaban en completos cambios en las predicciones. En el contexto de estudiar y diseñar estos ataques adversariales, surgen las Redes Generativas Adversariales (GANs), propuestas originalmente por Ian Goodfellow en 2014 [31].

Las GANs son una clase de algoritmos generativos. Su fin es generar observaciones falsas cuya distribución sea lo más aproximada posible a una distribución objetivo $P^\star$ de la que solo disponemos de una muestra verdadera.

Los datos involucrados en el proceso son, en general, muy complejos y no es posible determinar con precisión $P^\star$ mediante un modelo paramétrico clásico, como es la estimación de máxima verosimilitud. Además, la dimensión de las muestras es muy grande, por lo que también hay que descartar técnicas de estimación no paramétrica, como vecinos más próximos.

El proceso se lleva a cabo mediante un cierto tipo de juego en el que participan una familia de generadores y una familia de discriminadores: los primeros tratan de crear muestras que sigan en la medida de lo posible la distribución $P^\star$, mientras que los segundos tienen como objetivo distinguir dichas observaciones generadas de las reales provenientes de $P^\star$. Ambos son implementados mediante redes neuronales que son entrenadas para optimizar una función objetivo tratando de conseguir que las muestras generadas sean lo más indistinguibles posibles de las originales.

Para comprender esto mejor, de manera más intuitiva, podemos pensar en el símil que se emplea en el papel original [31]: el generador se puede considerar análogo a un equipo de falsificadores que tratan de falsificar moneda y utilizarla sin ser detectados. Por su parte, el discriminador se puede interpretar como la policía, tratando de detectar la moneda falsa. La competición en este juego lleva a ambos equipos a mejorar sus métodos hasta que las falsificaciones son indistinguibles de los artículos genuinos.

Este capítulo se organiza tal y como se muestra a continuación, de forma que se aprovechan los conceptos anteriormente expuestos pudiendo, por fin, ver su relación con el objetivo del presente trabajo:

- Formulación matemática del problema: se seguirá el planteamiento expuesto en [31] y [5], buscando en esta sección una exposición lo más clara posible con el fin de ver la analogía con lo expuesto formalmente sobre el aprendizaje automático y las redes neuronales en el Capítulo 2. Por tanto, se explica el problema a resolver por las GANs y cómo implementarlo en las redes neuronales que conforman el generador y el discriminador, respectivamente.

- Relación con la divergencia de Jensen-Shannon: remarcamos en esta sección la importancia de resolver con la métrica adecuada el problema de discriminación estadístico, lo que nos había llevado a desarrollar la sección 1.1. Veremos, siguiendo los argumentos de [5], que las GANs, en el caso particular de poder trabajar con todos los discriminadores posibles, resuelven el problema expuesto anteriormente minimizando la divergencia de Jensen-Shannon. Para demostrarlo, se requiere el hecho de que su raíz sea una distancia. Además, finalmente, se reflexiona sobre la similitud con un problema de estimación clásico de máxima verosimilitud.
- Buenas propiedades estadísticas de las GANs: el caso particular que acabamos de mencionar de no restringir la familia de discriminadores es una mera idealización, pues en la práctica no sucede así. No obstante, demostraremos, tal y como se hace en [5], que las GANs presentan una serie de propiedades que garantizan su buen funcionamiento aún en el caso real en términos del error de Jensen-Shannon.

3.1. Formulación matemática del problema

Tras la descripción introductoria que acabamos de hacer de las GANs, nos encontramos ya en condiciones de describirlas formalmente teniendo en cuenta que los capítulos anteriores nos sirven de fundamento.

El propósito de las GANs, como ya sabemos, es generar ejemplos artificiales lo más parecidos posibles a los reales que se utilizan como entrada. Este concepto de similitud, impreciso así descrito, es el que se trataremos ahora de desarrollar y formular matemáticamente, basándonos en el papel original [31] y en [5].

Supondremos que estamos trabajando en $E \subseteq \mathbb{R}^d$, con d suficientemente grande y, además, que todas las medidas de probabilidad consideradas en esta sección son absolutamente continuas con respecto a una cierta medida μ. Consideraremos que nuestro modelo de datos disponibles (por ejemplo, imágenes) lo constituye una muestra de vectores aleatorios independientes e igualmente distribuidos $X_1, \dots, X_n$ con ley desconocida $P^\star$ en E, cuya función de densidad (respecto a μ) denotaremos por $p^\star$ y ya hemos demostrado que existe en estas condiciones anteriormente.

Se asume que es posible generar variables aleatorias $Z \in \mathbb{R}^{d'}$, $d' \ll d$, conocidas como *variables ruido*; y se considera una familia paramétrica de *generadores* $\mathcal{G} = \{G_\theta\}_{\theta\in\Theta}, \Theta \subset \mathbb{R}^p$, con p suficientemente grande y $G_\theta : \mathbb{R}^{d'} \to E$, que se aplican a dichas variables aleatorias Z.

El objetivo es usar $G_\theta(Z)$, con una elección adecuada de θ, como observación "falsa", es decir, con distribución parecida a la original $P^\star$. Identificamos para ello las leyes de $G_\theta(Z)$ con la densidad (respecto a μ), que denotamos por p_θ. De esta forma, la familia de densidades asociada a los generadores es $\mathcal{P} = \{p_\theta\}_{\theta\in\Theta}$, es decir, tenemos que $G_\theta(Z) \overset{\mathscr{L}}{=} p_\theta \mathrm{d}\mu$, donde para cada θ, p_θ es un candidato a representar la densidad $p^\star$. Es importante destacar que $p^\star$ no tiene por qué pertenecer a $\mathcal{P}$. De hecho, en general $p^\star \notin \mathcal{P}$ por la complejidad de los datos con los que trabajan las GANs.

En definitiva, se busca estimar θ tal que las observaciones procedentes de p_θ sean difíciles de distinguir de las de $p^\star$. Para conseguirlo se parte de $X_1, \dots, X_n$ y de una muestra "falsa" representada por $G_\theta(Z_1), \dots, G_\theta(Z_n)$, con $Z_1, \dots, Z_n$ *i.i.d.* $\overset{\mathscr{L}}{=} Z$; y se trata de distinguirlas, lo que conlleva entrenar un *discriminador* $D \in \mathcal{D}$, para cierta clase $\mathcal{D}$, con $D : E \to [0, 1]$. $D(\mathbf{x})$ se puede interpretar como la probabilidad de que la observación $\mathbf{x}$ provenga de la distribución original $P^\star$: cuanto más alto sea el valor, mayor es dicha probabilidad.

Con el objetivo que se acaba de describir, podríamos minimizar el riesgo empírico asociado a la *pérdida logística*. Veámoslo, teniendo presente lo visto en la Sección 1.2.4.2.

Codificando las dos clases por medio de la variable respuesta $y_i \in \mathcal{Y} = \{-1, 1\}$, sabemos que se tiene la pérdida logística (1.51) para $n = 1$. Es decir, en nuestro caso, con n observaciones, tenemos que resolver

$$\min_{w_0,\mathbf{w}} \sum_{i=1}^{n} \log\left(1 + e^{-y_i f(\mathbf{x}_i; w_0, \mathbf{w})}\right), \quad f(x_i; w_0, \mathbf{w}) = w_0 + \mathbf{w}^\top \mathbf{x}_i, \quad y_i \in \mathcal{Y}. \tag{3.1}$$

Una interpretación equivalente es, tal y como hicimos en (1.57), utilizando la transformación logit [17]:

$$\begin{aligned} \mathrm{P}(y=1 \mid X=\mathbf{x}_i) &= \frac{1}{1+e^{-f(x_i;w_0,\mathbf{w})}}, \\ \mathrm{P}(y=-1 \mid X=\mathbf{x}_i) &= \frac{e^{-f(x_i;w_0,\mathbf{w})}}{1+e^{-f(x_i;w_0,\mathbf{w})}} = \frac{1}{1+e^{f(x_i;w_0,\mathbf{w})}}. \end{aligned} \tag{3.2}$$

Las probabilidades las podemos denotar $\mathrm{P}(y=y_i \mid X=\mathbf{x}_i) = p(\mathbf{x}_i, y_i; w_0, \mathbf{w})$. Por tanto, $f(\mathbf{x}_i, w_0, \mathbf{w}) \geq 0$ se clasifica como $y_i = +1$ y esto ocurre si, y solo si, $p(\mathbf{x}_i, y_i; w_0, \mathbf{w}) \geq \frac{1}{2}$; y al contrario con el caso $f(\mathbf{x}_i, w_0, \mathbf{w}) < 0$.

El problema (3.1) ya sabemos que es equivalente a minimizar

$$-\sum_{i=1}^{N} \log p(\mathbf{x}_i, y_i; w_0, \mathbf{w}),$$

es decir, a maximizar la log-verosimilitud.

En concreto, en el caso que nos ocupa, $X_1, \ldots, X_n$ frente a $G_\theta(Z_1), \ldots, G_\theta(Z_n)$, se trata de maximizar

$$\sum_{i=1}^{n} \log p(X_i, y_i; w_0, \mathbf{w}) + \sum_{i=1}^{n} \log\left(1 - p\left(G_\theta(Z_i), y_i; w_0, \mathbf{w}\right)\right), \tag{3.3}$$

o, equivalentemente,

$$\prod_{i=1}^{n} p(X_i, y_i; w_0, \mathbf{w}) \times \prod_{i=1}^{n} \left(1 - p\left(G_\theta(Z_i), y_i; w_0, \mathbf{w}\right)\right). \tag{3.4}$$

Notemos que podemos tomar otra función $D(\mathbf{x}_i) \in [0,1]$, que no tiene por qué ser necesariamente $p(\mathbf{x}_i, y_i; w_0, \mathbf{w})$. Es decir, un discriminador D de cierta clase $\mathcal{D}$ y maximizar

$$\prod_{i=1}^{n} D(X_i) \times \prod_{i=1}^{n} \left(1 - D \circ G_\theta\left(Z_i\right)\right).$$

Entonces,

$$\max_{D \in \mathcal{D}} \prod_{i=1}^{n} D\left(X_i\right) \times \prod_{i=1}^{n} \left(1 - D \circ G_\theta\left(Z_i\right)\right) \tag{3.5}$$

es la verosimilitud maximizada; es una medida de la capacidad de la clase $\mathcal{D}$ (el modelo $\mathcal{D}$) de discriminar observaciones verdaderas de observaciones "falsas". La idea de las GANs es, como dijimos, buscar el valor de θ que hace más difícil discriminar entre las observaciones auténticas y las generadas en el sentido anterior, lo que motiva considerar el *problema minimax*

$$\inf_{\theta \in \Theta} \sup_{D \in \mathcal{D}} \left[\prod_{i=1}^{n} D\left(X_i\right) \times \prod_{i=1}^{n} \left(1 - D \circ G_\theta\left(Z_i\right)\right) \right], \tag{3.6}$$

o, equivalentemente, encontrar $\hat{\theta} \in \Theta$ tal que

$$\sup_{D \in \mathcal{D}} \hat{\ell}(\hat{\theta}, D) \leq \sup_{D \in \mathcal{D}} \hat{\ell}(\theta, D) \quad \forall \theta \in \Theta, \tag{3.7}$$

donde se define función objetivo a optimizar como

$$\hat{\ell}(\theta, D) = \frac{1}{n} \sum_{i=1}^{n} \log D\left(X_i\right) + \frac{1}{n} \sum_{i=1}^{n} \log\left(1 - D \circ G_\theta\left(Z_i\right)\right). \tag{3.8}$$

Se trata de la versión empírica de:

$$\ell(\theta, D) = \int \log(D) p^\star d\mu + \int \log(1-D) p_\theta d\mu. \tag{3.9}$$

Estamos ya en condiciones de apreciar la clara y previamente anunciada analogía existente entre la función objetivo a optimizar en el caso de las GANs (3.8) y en el caso de la regresión logística (1.61).

Continuando con nuestro desarrollo, a partir de (3.9), se puede reescribir el problema minimax como

$$\min_{\theta} \max_{D} \ell(\theta, D) = \mathbb{E}_{X\sim P^\star}[\log D(X)] + \mathbb{E}_{Z\sim P_\theta}[\log(1 - D \circ (G_\theta(Z))]. \tag{3.10}$$

Interpretemos estas expresiones. El juego con el que comenzábamos describiendo las GANs viene representado tanto por (3.6) como por (3.10): se trata de que la distribución de $G_\theta(Z_i)$ (es decir, p_θ) sea lo más indistinguible posible de la distribución de X_i (es decir, $P^\star$). Sabemos que la capacidad de distinción la tienen los discriminadores. Por tanto, D se determina de tal forma que sea máximo en X_i y mínimo en $G_\theta(Z_i)$.

En otras palabras, D se entrena para maximizar la probabilidad de asignar las etiquetas correctas tanto a las muestras del conjunto de entrenamiento como a las generadas; por su parte, G se entrena para minimizar $\log(1 - D \circ (G_\theta(Z))$, es decir, para minimizar la capacidad de discriminación.

Equivalentemente, en (3.7) se muestra que la situación más favorable será aquella en que se anule $\sup_{D\in\mathcal{D}} \hat{\ell}(\theta, D)$ o, lo que es lo mismo, cuanto más grande sea esta cantidad, más distintas serán las muestras.

En resumen, G_θ debe ser tal que minimice $\sup_{D\in\mathcal{D}} \hat{\ell}(\theta, D)$ con el objetivo de encontrar $\hat{\theta}$ tal que $G_{\hat{\theta}}(Z_1), \ldots, G_{\hat{\theta}}(Z_n)$ tengan una distribución lo más aproximada posible a la desconocida $P^\star$. Es decir, que las muestras generadas sean lo más indistinguibles posibles de las originales. Por su parte, D debe ser tal que maximice $\sup_{D\in\mathcal{D}} \hat{\ell}(\theta, D)$ y, por tanto, maximice la probabilidad de asignar la etiqueta correcta tanto a las muestras del conjunto de entrenamiento como a las que han sido generadas.

Notemos que la familia de generadores $\mathcal{G}$ es paramétrica, por lo que en la práctica son funciones diferenciables con parámetro θ que se trata de estimar mediante una red neuronal. Asimismo, la familia de discriminadores $\mathcal{D}$ también se tendrá que considerar paramétrica en la práctica $\mathcal{D} = \{D_\alpha\}_{\alpha\in\Lambda}$, $\Lambda \subset \mathbb{R}^q$, con q suficientemente grande, e implementarse mediante otra red neuronal.

Ambas redes neuronales se entrenarán simultáneamente, teniendo en cuenta cuál es su objetivo a la hora de computar el algoritmo de Descenso de Gradiente Estocástico (Algoritmo 2). En primer lugar, en este caso los parámetros que se deben actualizar son θ para los generadores y α para los discriminadores. El otro cambio se debe a que, en el caso de los generadores, puesto que buscan minimizar la función objetivo, tendrá que hacerse tal y como se expuso en el Algoritmo 2; mientras que, en el caso de los discriminadores, la actualización de los parámetros deberá ir a favor del gradiente en cada iteración, puesto que se trata de maximizar la función objetivo.

3.2. Relación con la divergencia de Jensen-Shannon

Una vez formulado el problema en analogía a lo expuesto en capítulos anteriores sobre la teoría estadística de aprendizaje y las redes neuronales, nos centraremos en buscar la relación con la divergencia de Jensen-Shannon que ya habíamos adelantado. En concreto, veremos a partir de los resultados de [5] que, bajo la condición de no restringir la familia de discriminadores, las GANs plantean un problema de minimización de la citada divergencia.

Supongamos, por tanto, una situación idealizada en la que la familia $\mathcal{D}$ no está restringida, es decir, equivale al conjunto de todas las funciones de Borel de E en $[0, 1]$ y que denotamos por $\mathcal{D}_\infty$.

Teniendo en cuenta la expresión (3.9) y que el espacio de llegada de los discriminadores es el $[0, 1]$, podemos realizar la siguiente acotación:

$$0 \geq \sup_{D\in\mathcal{D}_\infty} \ell(\theta, D) \geq -\log 2 \left(\int p^\star d\mu + \int p_\theta d\mu \right) = -\log 4, \tag{3.11}$$

que implica que

$$\inf_{\theta\in\Theta}\sup_{D\in\mathcal{D}_\infty}\ell(\theta,D)\in[-\log 4,0]. \tag{3.12}$$

En consecuencia,

$$\inf_{\theta\in\Theta}\sup_{D\in\mathcal{D}_\infty}\ell(\theta,D)=\inf_{\theta\in\Theta}\sup_{D\in\mathcal{D}_\infty:\ell(\theta,D)>-\infty}\ell(\theta,D), \tag{3.13}$$

lo cual nos permite definir los *discriminadores $\theta-$admisibles* como aquellos tales que $\ell(\theta,D)>-\infty$. En lo que sigue, consideraremos solamente los de este tipo.

Trabajando en $\mathcal{D}_\infty$ podemos escribir el $\sup_{D\in\mathcal{D}_\infty}\ell(\theta,D)$ tal y como sigue, lo cual nos dará la relación buscada con la divergencia de Jensen-Shannon:

$$\begin{aligned}\sup_{D\in\mathcal{D}_\infty}\ell(\theta,D)&=\sup_{D\in\mathcal{D}_\infty}\int[\log(D)p^\star+\log(1-D)p_\theta]\,d\mu\\&\leq\int_{D\in\mathcal{D}_\infty}\sup[\log(D)p^\star+\log(1-D)p_\theta]\,d\mu\\&=\ell\left(\theta,D_\theta^\star\right),\end{aligned} \tag{3.14}$$

donde,

$$D_\theta^\star=\frac{p^\star}{p^\star+p_\theta} \tag{3.15}$$

es el *discriminador óptimo*, es decir, el que se corresponde con la regla de Bayes. (Por convenio, $0/0=0$ y $\infty\times 0=0$).

Operando y utilizando propiedades de los logaritmos, se tiene que

$$\ell(\theta,D_\theta^\star)=2D_{JS}(p,p_\theta)-\log 4,\quad\forall\theta\in\Theta. \tag{3.16}$$

A partir de (3.14) y (3.16), llegamos finalmente a la igualdad

$$\sup_{D\in\mathcal{D}_\infty}\ell(\theta,D)=\ell\left(\theta,D_\theta^\star\right)=2D_{JS}\left(p^\star,p_\theta\right)-\log 4,\quad\forall\theta\in\Theta. \tag{3.17}$$

Esta expresión (3.17) resume lo que hemos deducido: $D_\theta^\star$ es el supremo óptimo de $\ell(\theta,D)$ en $\mathcal{D}_\infty$, el cual está relacionado con la divergencia de Jensen-Shannon entre $p^\star$ y p.

Respecto a la unicidad de $D_\theta^\star$, no es cierta en general. Veremos en lo que sigue que sí que se tiene en determinadas condiciones. En concreto, cumplirse que $\mu\left(\{p^\star=p_\theta=0\}\right)=0$.

Teorema 3.1. *Sean $\theta\in\Theta$ y $D\in\mathcal{D}_\infty$ tales que $\ell(\theta,D)=\ell(\theta,D_\theta^\star)$. Entonces, $D=D_\theta^\star$ $\mu-$c.s. en el complementario del conjunto $\{p^\star=p_\theta=0\}$.*

En particular, si este conjunto es de medida nula, $\mu\left(\{p^\star=p_\theta=0\}\right)=0$, entonces la función $D_\theta^\star$ es el único discriminador que tal que

$$\{D_\theta^\star\}=\operatorname*{argmax}_{D\in\mathcal{D}_\infty}\ell(\theta,D). \tag{3.18}$$

Demostración.

Sea $D\in\mathcal{D}_\infty$ un discriminador tal que $\ell(\theta,D)=\ell(\theta,D_\theta^\star)$. En particular, supongamos que D es $\theta-$*admisible*, es decir, $\ell(\theta,D)>-\infty$. Sea $A=\{p^\star=p_\theta=0\}$ y $f_\alpha=p^\star\log(\alpha)+p_\theta\log(1-\alpha)$, $\alpha\in[0,1]$. Así, tenemos que, de acuerdo con (3.9)

$$\ell(\theta,D)=\int f_D d\mu,$$

y

$$\ell(\theta, D_\theta^\star) = \int f_{D_\theta^\star} d\mu.$$

Por tanto,

$$\int_{A^c} \left(f_D - f_{D_\theta^\star}\right) d\mu = 0.$$

Además, por (3.17) se tiene que

$$\sup_{D \in \mathcal{D}_\infty} \ell(\theta, D) = \ell(\theta, D_\theta^\star), \quad \forall \theta \in \Theta,$$

con lo cual:

$$f_D \leq f_{D_\theta}^\star = \sup_{\alpha \in [0,1]} f_\alpha \quad \forall D \in \mathcal{D}_\infty.$$

Deducimos que $f_{D_\theta} = f_{D_\theta^\star}$ $\mu - c.s.$ en A^c. Por unicidad del máximo de $\alpha \mapsto f_\alpha$ en A^c, podemos concluir ya que $D = D_\theta^\star$ $\mu - c.s.$ en A^c.

Veamos ahora con un contraejemplo que la condición $\mu(A) = 0$ es necesaria para garantizar la unicidad de $D_\theta^\star$ buscada. Supongamos que $p_\theta = p^\star$. En este caso, $\forall \bar{D} \in \mathcal{D}_\infty$, el discriminador $D_\theta^\star \mathbf{1}_{\{p_\theta > 0\}} + \bar{D}\mathbf{1}_{\{p_\theta = 0\}}$ satisface

$$\ell\left(\theta, D_\theta^\star \mathbf{1}_{\{p_\theta > 0\}} + \bar{D}\mathbf{1}_{\{p_\theta = 0\}}\right) = \ell\left(\theta, D_\theta^\star\right),$$

lo cual implica que, si $\mu(A) \neq 0$, entonces $D_\theta^\star$ no es único. □

A partir de la expresión (3.17) tiene sentido definir ahora el parámetro $\theta^\star \in \Theta$ como aquel que cumple

$$\ell\left(\theta^\star, D_{\theta^*}^\star\right) \leq \ell\left(\theta, D_\theta^\star\right) \quad \forall \theta \in \Theta, \quad \text{si} \quad \mathcal{D} = \mathcal{D}_\infty, \tag{3.19}$$

o, equivalentemente,

$$D_{JS}\left(p^\star, p_{\theta^*}\right) \leq D_{JS}\left(p^\star, p_\theta\right) \quad \forall \theta \in \Theta, \quad \text{si} \quad \mathcal{D} = \mathcal{D}_\infty. \tag{3.20}$$

Por tanto, si trabajamos con $\mathcal{D}_\infty$, podemos interpretar $\theta^\star$ como *el mejor parámetro* en Θ para estimar la densidad desconocida $p^\star$ en términos de la divergencia de Jensen-Shannon. Es decir, el generador $G_{\theta^\star}$ es el generador óptimo y la densidad $p_{\theta^\star}$ es la que nos gustaría usar para generar las muestras falsas. Nótese que si $p^\star \in \mathcal{P}$, entonces $p^\star = p_{\theta^\star}$ y $D_{JS}\left(p^\star, p_{\theta^*}\right) = 0$, que cuadra con la idea de que, en este caso, las GANs tratan de minimizar esta divergencia. No obstante, recordemos que, en general, $p^\star \notin \mathcal{P}$.

Veamos a continuación bajo qué condiciones $\theta^\star$ existe y es único: sean P y Q probabilidades en E y trabajemos con la distancia de Jensen-Shannon definida en la sección 1.1.2.2 como $\delta = \sqrt{D_{JS}(P, Q)}$. Ya dedujimos del Teorema de Radon-Nikodym A.1, en (1.4), que podemos considerar $p^\star = \frac{dP^\star}{d\mu}$ y $p_\theta = \frac{dP_\theta}{d\mu}$.

Es necesario, en primer lugar, que $\mathcal{P} = \{P_\theta\}_{\theta \in \Theta}$ sea compacto para dicha métrica. Para garantizarlo, demostremos el siguiente lema.

Lema 3.1. *Supongamos que Θ es compacto, $\mathcal{P}$ es convexo y:*

1. *$\forall x \in E$, la función $\theta \mapsto p_\theta(x)$ es continua en Θ.*
2. *$\sup_{(\theta,\theta') \in \Theta^2} |p_\theta \log p_{\theta'}| \in \mathcal{L}^1(\mu)$.*

Entonces, $\mathcal{P}$ es compacto para la métrica δ.

Demostración.

Puesto que Θ es compacto, por la caracterización secuencial de la compacidad, es suficiente probar que, dada la sucesión $(\theta_n)_n \subset \Theta$, si converge a $\theta \in \Theta$, entonces $D_{JS}(p_\theta, p_{\theta_n}) \to 0$, donde, por definición

$$D_{JS}\left(p_\theta, p_{\theta_n}\right) = \int \left[p_\theta \log\left(\frac{2p_\theta}{p_\theta + p_{\theta_n}}\right) + p_{\theta_n} \log\left(\frac{2p_{\theta_n}}{p_\theta + p_{\theta_n}}\right)\right] d\mu.$$

Podemos concluir que converge hacia 0 aplicando el Teorema de la Convergencia Dominada de Lebesgue ([64], Teorema 1.34), puesto que $\mathcal{P}$ es convexo por hipótesis y estamos suponiendo *1* y *2*. □

La primera parte del siguiente teorema, interpretada en términos de distribuciones en lugar de parámetros, garantiza la existencia y la unicidad de $\theta^\star$.

Teorema 3.2. *Supongamos que el modelo $\mathcal{P} = \{P_\theta\}_{\theta\in\Theta}$ es convexo y compacto para la métrica δ. Si $p^\star > 0$ $\mu - c.s.$, entonces existe una única densidad $\bar{p} \in \mathcal{P}$ tal que*

$$\{\bar{p}\} = \arg\min_{p\in\mathcal{P}} D_{JS}\left(p^\star, p\right). \tag{3.21}$$

En particular, si el modelo $\mathcal{P}$ es identificable, entonces

$$\{\theta^\star\} = \arg\min_{\theta\in\Theta} L\left(\theta, D_\theta^\star\right), \tag{3.22}$$

o, equivalentemente,

$$\{\theta^\star\} = \arg\min_{\theta\in\Theta} D_{JS}\left(p^\star, p_\theta\right). \tag{3.23}$$

Nótese que la condición de ser un modelo paramétrico identificable expresada en la segunda parte del teorema es muy difícil de satisfacer a causa de la dimensión tan alta con la que trabajan las GANs.

Demostración.

La segunda parte se deduce inmediatamente de la anterior a partir de la expresión (3.17). Por tanto, basta probar que existe una única densidad $\bar{p}$ de $\mathcal{P}$ tal que

$$\{\bar{p}\} = \arg\min_{p\in\mathcal{P}} D_{JS}\left(p^\star, p\right).$$

Existencia: puesto que $\mathcal{P}$ es compacto para δ, basta ver que la función

$$\begin{aligned} &\mathcal{P} \to \mathbb{R}+ \\ &P \mapsto D_{JS}\left(P^\star, P\right) \end{aligned}$$

es continua. Esto es claro, pues, para todo $P_1, P_2 \in \mathcal{P}$,

$$\left|\delta\left(P^\star, P_1\right) - \delta\left(P^\star, P_2\right)\right| \leq \delta\left(P_1, P_2\right),$$

por la desigualdad triangular. Por tanto $\arg\min_{p\in\mathcal{P}} D_{JS}\left(p^\star, p\right) \neq \emptyset$.

Unicidad: para $a \geq 0$, consideremos la función F_a definida por

$$F_a(x) = a\log\left(\frac{2a}{a+x}\right) + x\log\left(\frac{2x}{a+x}\right), \quad x \geq 0.$$

(Por convenio, $0\log 0 = 0$). Puesto que

$$F_a''(x) = \frac{a}{x(a+x)},$$

F_a es estrictamente convexa siempre que $a > 0$.

Probemos ahora que $p \mapsto D_{JS}(p^\star, p)$, $p \in \mathcal{P} \subset \mathcal{L}^1(\mu)$ es también estrictamente convexa. Sea $\lambda \in (0,1)$ y $p_1, p_2 \in \mathcal{P}$ con $p_1 \neq p_2$, es decir, $\mu(\{p_1 \neq p_2\}) > 0$. Entonces

$$\begin{aligned} & D_{JS}\left(p^\star, \lambda p_1 + (1-\lambda)p_2\right) \\ & = \int F_{p^\star}\left(\lambda p_1 + (1-\lambda)p_2\right) \mathrm{d}\mu \\ & = \int_{\{p_1 = p_2\}} F_{p^\star}\left(p_1\right) d\mu + \int_{\{p_1 \neq p_2\}} F_{p^\star}\left(\lambda p_1 + (1-\lambda)p_2\right) d\mu. \end{aligned}$$

Puesto que $F_{p^\star}$ es estrictamente convexa en $\{p^\star > 0\}$, tenemos

$$\begin{aligned} & D_{JS}\left(p^\star, \lambda p_1 + (1-\lambda)p_2\right) \\ & < \int_{\{p_1 = p_2\}} F_{p^*}\left(p_1\right) d\mu + \lambda \int_{\{p_1 \neq p_2\}} F_{p^*}\left(p_1\right) d\mu \\ & + (1-\lambda) \int_{\{p_1 \neq p_2\}} F_{p^*}\left(p_2\right) d\mu, \end{aligned}$$

lo cual implica

$$D_{JS}\left(p^\star, \lambda p_1 + (1-\lambda)p_2\right) < \lambda D_{JS}\left(p^\star, p_1\right) + (1-\lambda) D_{JS}\left(p^\star, p_2\right).$$

Por tanto, la función $p \mapsto D_{JS}(p^\star, p)$, $p \in \mathcal{P} \subset \mathcal{L}^1(\mu)$ es estrictamente convexa y, en consecuencia su arg mín en el conjunto convexo $\mathcal{P}$ es, o bien el vacío, o bien un conjunto unipuntual. □

En conclusión, la divergencia de Jensen-Shannon es la que está realmente implicada en el problema que resuelven las GANs: en el **caso idealizado** en que **no restringimos la familia de discriminadores** $\mathcal{D} = \mathcal{D}_\infty$ puede formularse, asintóticamente, como un **problema de minimización de la divergencia de Jensen-Shannon** según (1.22). Es decir, más concretamente, trabajando con $\mathcal{D}_\infty$ puede interpretarse como la búsqueda del estimador óptimo de $\inf_{\theta \in \Theta} D_{JS}\left(p^\star, p_\theta\right)$, frente a la estimación de máxima verosimilitud, que tiene como objetivo, también asintóticamente, minimizar la divergencia de Kullback.

Sin embargo, suponer que podemos trabajar con D_∞ es una mera idealización. En la práctica, como ya hemos comentado, no todos los discriminadores están disponibles y se trabaja con una familia paramétrica $\mathcal{D} = \{D_\alpha\}_{\alpha \in \Lambda}$, $\Lambda \in \mathbb{R}^q$.

3.3. Garantías estadísticas para GANs

Hasta ahora hemos tratado el caso ideal en el que la familia de discriminadores no está restringida, aunque hemos hecho mención en varias ocasiones a que esta interpretación no es realista, por lo que nos gustaría asegurar que no se ha perdido generalidad en el desarrollo anterior a la hora de garantizar el correcto rendimiento en la práctica.

Pasemos, por tanto, a demostrar que considerando la formulación matemática anteriormente expuesta del problema, las GANs presentan ciertas propiedades estadísticas expuestas en [5] que ofrecen garantías sobre su buen funcionamiento.

Trabajaremos ahora en el caso real en que la clase de discriminadores está restringida a $\mathcal{D} = \{D_\alpha\}_{\alpha \in \Lambda}$, $\Lambda \in \mathbb{R}^q$. Por tanto, *el estimador en la práctica* es $\hat{\theta} \in \Theta$ y es el que se utiliza para crear muestras falsas mediante el generador $G_{\hat{\theta}}$. Para evitar confusiones con un discriminador dado, $D = D_\alpha$, usaremos la notación $\hat{\ell}(\theta, \alpha)$ (resp. $\ell(\theta, \alpha)$) en lugar de $\hat{\ell}(\theta, D)$ (resp. $\ell(\theta, D)$).

Además, puesto que lo utilizaremos en el posterior desarrollo, recordemos que denotábamos $D_\theta^\star$ (3.15) al *discriminador óptimo* (regla de Bayes) y $\theta^\star$ (3.19) al *mejor parámetro* en Θ para estimar la densidad desconocida $p^\star$ cuando se trabaja en $\mathcal{D}_\infty$.

Comencemos estudiando las propiedades de la distribución $p_{\hat{\theta}}$ en términos del *error de Jensen-Shannon*, $D_{JS}\left(p^\star, p_{\hat{\theta}}\right) - D_{JS}\left(p^\star, p_{\theta^\star}\right)$.

Supondremos en lo que sigue que Θ y Λ son subconjuntos compactos de $\mathbb{R}^p$ y $\mathbb{R}^q$, respectivamenteme. Además, asumiremos que $\mu(E) < \infty$, en cuyo caso todos los discriminadores son $\mathcal{L}^p(\mu)$ para todo $p \geq 1$

El siguiente teorema nos garantiza el buen funcionamiento de las GANs en el caso práctico, pues asegura que, asintóticamente, la diferencia $D_{JS}\left(p^\star, p_{\hat{\theta}}\right) - D_{JS}\left(p^\star, p_{\theta^\star}\right)$ no es mayor que un término proporcional a ε^2. A pesar de las limitaciones que tiene debido a la serie de hipótesis que se requiere, que podría considerarse en cierto modo restrictiva, es el **primer resultado** interesante sobre la teoría estadística de las GANs que aporta garantías teóricas sobre su buen funcionamiento. Su demostración se ha publicado en [5] y esta será la versión que utilicemos para probarlo a continuación.

Teorema 3.3. *Supongamos que, para cierto $M > 0$, $p^\star \leq M$ para todo $\theta \in \Theta$ y hagamos las siguientes suposiciones de regularidad:*

1. *Existe $\kappa \in (0, 1/2)$ tal que, para todo $\alpha \in \Lambda$, $\kappa \leq D_\alpha \leq 1 - \kappa$. Además, la función $(x, \alpha) \mapsto D_\alpha(x)$ es de clase $\mathcal{C}^1$ y su diferencial está uniformemente acotada.*
2. *Para todo $z \in \mathbb{R}^d$, la función $\theta \mapsto G_\theta(z)$ es de clase $\mathcal{C}^1$, uniformemente acotada y su diferencial también está uniformemente acotada.*
3. *Para todo $x \in E$, la función $\theta \mapsto p_\theta(x)$ es de clase $\mathcal{C}^1$, uniformemente acotada y su diferencial también está uniformemente acotada.*
4. *Existe $\varepsilon > 0$ y $m \in (0, 1/2)$ tales que, para todo $\theta \in \Theta$, existe $D \in \mathcal{D}$ acotado tal que $m \leq D \leq 1 - m$ y $\|D - D^\star_\theta\|_2 \leq \varepsilon$.*

Tomando $\varepsilon < 1/(2M)$, se tiene que:

$$\mathbb{E}D_{JS}\left(p^\star, p_{\hat{\theta}}\right) - D_{JS}\left(p^\star, p_{\theta^\star}\right) = \mathrm{O}\left(\varepsilon^2 + \frac{1}{\sqrt{n}}\right). \tag{3.24}$$

Comencemos notando que, bajo las tres primeras suposiciones, se tiene que $\hat{\ell}(\theta, \alpha)$ y $\ell(\theta, \alpha)$ son continuas. Además, gracias a que asumimos 1, todos los discriminadores de $D = D_\alpha$ son $\theta - admisible$ para todo θ. Recordemos que también suponíamos la compacidad de Θ. Por tanto, garantizamos la existencia de $\hat{\theta}$; la de $\theta^\star$, por su parte, sabemos que está asegurada por el Teorema 3.2. En segundo lugar, cabe destacar que la expresión (3.24) representa el exceso de riesgo medido en la distancia de Jensen-Shannon δ.

Pasemos a demostrar el teorema, para lo cual se recomienda consultar el apéndice B si se quiere obtener una mayor explicación sobre los conceptos que se van a emplear.

Demostración.

Fijemos $\varepsilon \in (0, 1/(2M))$. Como estamos suponiendo 4, podemos elegir $\hat{D} \in \mathcal{D}$ tal que $m \leq \hat{D} \leq 1 - m$ y $\|\hat{D} - D^\star_{\hat{\theta}}\|_2 \leq \varepsilon$. Se puede demostrar (ver en [5], theorem 4.1, pág. 1548) que existe una constante $c_1 > 0$ tal que

$$2D_{JS}\left(p^\star, p_{\hat{\theta}}\right) \leq c_1\varepsilon^2 + \ell(\hat{\theta}, \hat{D}) + \log 4 \leq c_1\varepsilon^2 + \sup_{\alpha \in \Lambda} \ell(\hat{\theta}, \alpha) + \log 4.$$

Por tanto,

$$\begin{aligned} 2D_{JS}\left(p^\star, p_{\hat{\theta}}\right) &\leq c_1\varepsilon^2 + \sup_{\theta \in \Theta, \alpha \in \Lambda} |\hat{\ell}(\theta, \alpha) - \ell(\theta, \alpha)| + \sup_{\alpha \in \Lambda} \hat{\ell}(\hat{\theta}, \alpha) + \log 4 \\ &= c_1\varepsilon^2 + \sup_{\theta \in \Theta, \alpha \in \Lambda} |\hat{\ell}(\theta, \alpha) - \ell(\theta, \alpha)| + \operatorname{ínf}_{\theta \in \Theta} \sup_{\alpha \in \Lambda} \hat{\ell}(\theta, \alpha) + \log 4 \\ &\leq c_1\varepsilon^2 + 2 \sup_{\theta \in \Theta, \alpha \in \Lambda} |\hat{\ell}(\theta, \alpha) - \ell(\theta, \alpha)| + \operatorname{ínf}_{\theta \in \Theta} \sup_{\alpha \in \Lambda} \ell(\theta, \alpha) + \log 4, \end{aligned}$$

donde en la primera igualdad se ha aplicado la definición de $\hat{\theta}$ (3.7). Recordemos que, por (3.7), $\hat{\theta}$ es el minimizador en Θ de $\sup_{\alpha \in \Lambda} \hat{\ell}(\hat{\theta}, \alpha)$.

Entonces, puesto que $\Lambda \subset \mathcal{D}_\infty$, podemos seguir desarrollando la desigualdad tal que así:

$$\begin{aligned} 2D_{JS}\left(p^\star, p_{\hat{\theta}}\right) &\leq c_1\varepsilon^2 + 2\sup_{\theta\in\Theta,\alpha\in\Lambda}|\hat{\ell}(\theta,\alpha)-\ell(\theta,\alpha)| + \inf_{\theta\in\Theta}\sup_{D\in\mathscr{D}_\infty}\ell(\theta,D)+\log 4 \\ &= c_1\varepsilon^2 + 2\sup_{\theta\in\Theta,\alpha\in\Lambda}|\hat{\ell}(\theta,\alpha)-\ell(\theta,\alpha)| + \ell\left(\theta^\star, D^\star_{\theta^\star}\right)+\log 4 \\ &= c_1\varepsilon^2 + 2D_{JS}\left(p^\star, p_{\theta^\star}\right) + 2\sup_{\theta\in\Theta,\alpha\in\Lambda}|\hat{\ell}(\theta,\alpha)-\ell(\theta,\alpha)|, \end{aligned} \tag{3.25}$$

donde, de nuevo, en la primera igualdad se ha aplicado la definición de $\hat{\theta}$ (3.7).

Finalmente, tomando $c_2 = c_1/2$, tenemos que:

$$D_{JS}\left(p^\star, p_{\hat{\theta}}\right) - D_{JS}\left(p^\star, p_{\theta^\star}\right) \leq c_2\varepsilon^2 + \sup_{\theta\in\Theta,\alpha\in\Lambda}|\hat{\ell}(\theta,\alpha)-\ell(\theta,\alpha)|. \tag{3.26}$$

Puesto que estamos suponiendo 1, 2 y 3, $(\hat{\ell}(\theta,\alpha)-\ell(\theta,\alpha))_{\theta\in\Theta,\alpha\in\Lambda}$ es un proceso sub-gaussiano separable para la distancia $d = S\|\cdot\|/\sqrt{n}$ (Definiciones B.2 y B.3), donde $\|\cdot\|$ es la distancia Euclídea en $\mathbb{R}^p\times\mathbb{R}^q$ y $S>0$ depende solamente de las cotas tomadas al realizar las suposiciones 1 y 2.

Sea $N(\Theta\times\Lambda, \|\cdot\|, u)$ el *u-número de cubrimiento* de $\Theta\times\Lambda$ para la distancia $\|\cdot\|$ (Definición B.1). Entonces, a partir de la *cota de entropía de Dudley* (Corolario B.1.1), se tiene que:

$$\mathbb{E}\sup_{\theta\in\Theta,\alpha\in\Lambda}|\hat{L}(\theta,\alpha)-L(\theta,\alpha)| \leq \frac{12S}{\sqrt{n}}\int_0^\infty \sqrt{\log(N(\Theta\times\Lambda,\|\cdot\|,u))}\mathrm{d}u. \tag{3.27}$$

Puesto que Θ y Λ están acotados, existe $r>0$ tal que $N(\Theta\times\Lambda,\|\cdot\|,u)=1$ para todo $u\geq r$, lo cual permite que integrar de 0 a r en (3.27), y tal que

$$N(\Theta\times\Lambda,\|\cdot\|,u) = \mathrm{O}\left(\left(\frac{\sqrt{p+q}}{u}\right)^{p+q}\right), \quad u<r, \tag{3.28}$$

que se justifica a partir del hecho de que, para todo $\varepsilon>0$, el u-número de cubrimiento en un compacto de dimensión d en $\mathbb{R}^d$ es de orden $1/\varepsilon^d$ (ver [33], Lema 5.13).

Por tanto, puesto que la integral de (3.27) es convergente, combinando esto con (3.26) y (3.28), llegamos finalmente a que:

$$\mathbb{E}D_{\mathrm{JS}}\left(p^\star, p_{\hat{\theta}}\right) - D_{\mathrm{JS}}\left(p^\star, p_{\theta^\star}\right) \leq c_3\left(\varepsilon^2 + \frac{1}{\sqrt{n}}\right),$$

para cierta constante positiva que escala como $p+q$.

Se concluye la demostración observando que, por (3.20) se cumple que

$$D_{JS}\left(p^\star, p_{\theta^*}\right) \leq D_{JS}\left(p^\star, p_{\hat{\theta}}\right) \quad \forall\theta\in\Theta.$$

□

Capítulo 4

Implementación

Una vez expuesto el fundamento teórico de las GANs, nos proponemos llevar a cabo la parte práctica del trabajo, consistente en utilizar principalmente Keras, una biblioteca desarrollada por François Chollet que puede emplearse tanto en R como en Python y que ha sido diseñada para aplicaciones específicas de aprendizaje profundo. En este capítulo daremos más detalles al respecto con el fin de comprender el código necesario para implementar una GAN en busca de lograr el objetivo de generar nuestras propias imágenes a partir de ciertos conjuntos de datos ya etiquetados que se encuentran disponibles en internet. Asimismo, nos centraremos principalmente en relacionar toda la teoría anteriormente expuesta con los pasos a seguir, de manera que restaremos importancia a las líneas de código y su correspondiente sintaxiss.

La implementación de las técnicas de aprendizaje profundo estudiadas merece nuestro atención porque está suponiendo una revolución tecnológica en la actualidad y se está llevando a cabo una importante labor investigadora al respecto; en particular en el ámbito de las redes neuronales se pueden citar como ejemplo [4, 11, 38, 54, 81]. Por este motivo, los cambios se están sucediendo de forma muy rápida, además de otros factores como el desarrollo y disponibilidad de potente hardware, especialmente GPUs (Unidades de Procesamiento Gráfico) así como de software específico [25] como Keras o Tensorflow. Además, cabe destacar la existencia de grandes cantidades de datos a partir de los cuales cualquier usuario de internet puede crear conjuntos de datos etiquetados que resultan necesarios para entrenar los modelos.

Para tratar todo ello, la organización del capítulo es la siguiente:

- Exposición a modo introductorio del panorama actual de las GANs. Se mencionan algunos de los tipos de GANs en que se investiga, así como sus aplicaciones prácticas, incidiendo especialmente en la generación de imágenes para motivar la aplicación que llevaremos a cabo.
- Descripción de algunos aspectos computacionales que se requiere entender para desarrollar la implementación final como son el tratamiento de imágenes, la importancia de la GPU y cómo construir modelos de aprendizaje profundo con Keras en R y/o Python.
- Desarrollo definitivo de la parte práctica del trabajo. Se generan imágenes de dígitos numéricos y de rostros humanos a partir de los conjuntos de datos *MNIST* [48] y *CelebA* [13]. Expondremos las explicaciones del código de forma descriptiva con el fin de restarle importancia a la sintaxis y centrarnos en comprender la relación con todo lo estudiado en este manuscrito.

Como referencias principales utilizaremos las contribuciones a la puesta en marcha de las técnicas de aprendizaje profundo mediante R y Python de François Chollet [15, 16], además de [43]. No obstante, remarquemos que para una comprensión correcta habrá que tener en todo momento presente la teoría anteriormente expuesta en el trabajo acerca de redes neuronales y GANs.

Además, la documentación de Keras para la sintaxis específica en R y en Python está disponible en [40, 60], respectivamente, aunque es muy similar. No la detallaremos pero, cualquier para cualquier respecto al código necesario al que hagamos referencia en lo que sigue, se puede recurrir a ella.

4.1. Introducción: aplicaciones de las GANs

Respecto a la implementación de las GANs, tres su reciente aparición en 2014 [31], cabe destacar que han logrado grandes éxitos en la generación de imágenes con varias arquitecturas que emplean redes neuronales convolucionales a partir de la exitosa aplicación de las *Deep Convolutional GAN (DCGAN)* [62]; por este motivo, las CNNs fueron explicadas en la sección 2.2. Por ejemplo, *StyleGan* [39] ha creado imágenes de rostros falsos indistinguibles de los reales. A partir de dicho modelo se ha creado el software *This person does not exist* [77] que muestra una cara generada nueva en cada recarga de la página, lo cual resulta curioso de probar. Otros éxitos en este contexto de generación de imágenes que podemos citar son [6, 80].

Además, como cabe esperar, para mejorar los defectos que se van encontrando se sigue investigando en el desarrollo de otras variantes, como *Wasserstein Gan* [1], *AutoGAN* [28] o *CycleGAN*[83].

Por tanto, cada vez se encuentran mejores resultados en tareas de visión computacional y generación o restauración de imágenes. Por ejemplo, se han propuesto algunas aplicaciones prácticas de utilidad recientemente como:

- Su utilización en medicina para generar imágenes a partir de resonancia magnética como las que se obtienen mediante tomografía computarizada, que se aprecian mejor, sin recurrir a esta última técnica ya que expone a los pacientes a mayores dosis de radiación [55].
- Restauración de fotografías tomadas bajo el mar de calidad aceptable [82] en las que se puedan realizar apreciaciones correctamente, lo que hasta ahora resulta una tarea complicada en el campo de la marina militar, ingeniería o investigación marina.
- Construcción de mejores imágenes en Radares de Apertura Sintética (SAR) [27], que son un tipo de radar que permite obtener imágenes de alta resolución a larga distancia, por ejemplo, desde el espacio.

Por último, aunque nos centraremos en la generación de imágenes, cabe mencionar que también se está investigando en el empleo de las GANs para generar audio [21, 24].

En definitiva, las GANs representan un nuevo concepo dentro del aprendizaje profundo que está ganando importancia a un ritmo muy rápido. El principal inconveniente que presentan es que aún requieren un alto costo computacional y son muy sensibles a ligeros cambios en los parámetros, lo cual aún no está controlado por completo desde el punto de vista teórico. Por tanto, el hecho de que se trate de una tecnología tan puntera y novedosa nos limitará a la hora de poder llevar a cabo la implementación de un código con el que mostrar algunos ejemplos propios.

Tras esta introducción en la que hemos podido mostrar algunas de las aplicaciones de las GANs, pasemos por fin a llevar a cabo algunas concretas con el fin de generar imágenes falsas.

4.2. Aspectos computacionales

En secciones anteriores ya hemos descrito algunos aspectos computacionales relativos a cómo se vectorizan las imágenes, a la importancia de determinadas elecciones de parámetros y técnicas con el fin de sacar el máximo partido de la GPU, etc. Ahora, detallaremos mejor su importancia y nos centraremos en la descripción de Keras y las ideas y herramientas principales en la construcción de modelos que llevaremos a cabo finalmente.

Keras es una biblioteca de código abierto creada por François Chollet y perteneciente actualmente a la plataforma de aprendizaje automático *TensorFlow*, desarrollada por Google. Puede utilizarse tanto en R como en Python y proporciona una forma útil y eficaz de definir y entrenar casi cualquier tipo de modelo de aprendizaje profundo.

Otras plataformas con el mismo objetivo que TensorFlow son *CNTK* desarrollada por Microsoft o *Theano* desarrollada por MILA. Para mayor información se puede consultar [25].

Inicialmente, las redes neuronales se entrenaban usando la CPU (Unidad Central de Procesamiento) de una sola máquina, es decir, lo que puede considerarse el procesador de la computadora. Actualmente, esto es insuficiente y se usa principalmente la GPU, que es un coprocesador dedicado al procesamiento de gráficos, altamente recomendable especialmente para el procesamiento de imágenes y CNNs, pues la velocidad de ejecución sería muy lenta en caso contrario.

Keras, via TensorFlow, permite ejecutar sin problemas tanto en CPUs como en GPUs. No obstante, las ventajas que ofrece la ejecución en GPUs solo es posible si el ordenador dispone de tarjetas gráficas, a lo cual no hemos tenido acceso en la realización del trabajo. Por tanto, se ha recurrido a utilizar *Google Colaboratory*, que permite ejecutar y programar en Python en el navegador con las ventajas de que no requiere configuración alguna adicional, da acceso gratuito a GPUs y permite compartir contenido fácilmente.

Pasemos ahora a explicar, en líneas generales, cómo se define un modelo de aprendizaje profundo con Keras, para lo cual utilizaremos una función concreta que permite añadir capas sucesivamente para crear las redes neuronales cuando tenemos solamente una entrada y una salida.

Recordemos que la estructura fundamental de las redes neuronales son las capas y la elección de estas se debe hacer en base al tipo de datos que estemos utilizando. Por ejemplo, si los datos son tensores dos-dimensionales de la forma `(muestras, etiquetas)`, se suelen utilizar capas densas. En cambio, las imágenes, tal y como vimos en la sección 2.2, se almacenan en tensores de cuatro dimensiones `(muestras, altura, anchura, canales)` y se procesan mediante capas convolucionales. En ese caso, hay que seleccionar convenientemente los núcleos, el paso, el relleno y la reducción, ya expuestos. Además, los valores de los píxeles varían en el rango $[0, 255]$ y conviene normalizarlos, bien al intervalo $[-1, 1]$, bien al intervalo $[0, 1]$, según convenga.

La idea es, en resumen, utilizar las capas adecuadas en función del tipo y de las dimensiones de los datos de entrada y de salida para conseguir las transformaciones deseadas.

Una vez que hemos definido la estructura por capas, debemos entrenar el modelo mediante la función correspondiente de la librería Keras. Esta permite elegir la función de pérdida que se debe optimizar y cómo aplicar el algoritmo SGD, seleccionando convenientemente el ratio de aprendizaje y la versión que deseemos aplicar (como sabemos, existen SGD con momento, SGD con momento de Nesterov o algoritmos con ratios de aprendizaje adaptables como AdaGrad, RMSProp o Adam). También hay que elegir el tamaño de los minilotes como potencias de dos, tal y como ya indicamos, así como el número de iteraciones. Para llevar a cabo las elecciones correctas conviene emplear algunas técnicas de regularización ya descritas, como la interrupción anticipada o el abandono.

Para la evaluación del modelo se pueden calcular ciertas métricas, en particular, utilizaremos la *exactitud* (o *accuracy*), que mide el porcentaje de predicciones realizadas correctamente.

4.3. Generación de imágenes

Pasemos, por fin, a implementar el código que requiere una GAN con el fin de generar imágenes, por un lado de dígitos y, por otro, de rostros humanos. Para entrenar el modelo utilizaremos, respectivamente, los conjuntos de datos disponibles en TensorFlow, *MNIST* [48] y *CelebA (CelebFaces Attributes Dataset)* [13].

Nos centraremos en hacer un tratamiento descriptivo de manera que se entienda el código en relación con toda la teoría anteriormente expuesta, distinguiendo dos alternativas:

- Generador y discriminador se implementan utilizando solamente redes neuronales prelimentadas.
- Se añaden, además, tanto en generador como en discrimindaor, redes neuronales convolucionales.

Veremos, como cabe esperar a estas alturas del trabajo, que la segunda proporciona mejores resultados al tratar con imágenes. En particular, compararemos ambas alternativas para el caso de la generación de dígitos y utilizaremos para los rostros humanos solamente la segunda, por motivos obvios.

Los pasos a seguir para construir un modelo de aprendizaje profundo en Keras son los que ya hemos descrito en la sección anterior y la mayoría de las decisiones se toman mediante prueba y error, aunque siempre podemos apoyarnos en los conocimientos teóricos de que disponemos. Igualmente, recordemos que la sintaxis concreta se puede consultar en la documentación [60] y [40].

Comencemos explicando cómo se debe crear la GAN, pues es común a los modelos que utilicemos. Sabemos que la idea es entrenar dos redes neuronales simultáneamente: el generador (que recibe un vector aleatorio y lo utiliza para sintetizar una imagen) y el discriminador (que recibe una imagen real o generada y trata de predecir de qué caso se trata).

El generador inicialmente solo generará ruido, por lo para entrenarlo con el fin de conseguir imágenes realistas, se utiliza el discriminador y se conectan ambos modelos mediante otro que constuitirá la GAN propiamente dicha.

Mientras que el generador se entrena vía el modelo GAN, es fundamental antes de llevar esta operación a cabo que el discriminador ya haya sido pre-entrenado mediante la combinación de imágenes reales y falsas (que se comienzan creando mediante píxeles seleccionados al azar), para lo cual se requiere otra función.

A la hora de entrenar la GAN se conectarán ambos modelos: generador y discriminador. En este proceso, los parámetros anteriormente obtenidos en el pre-entrenamiento del discriminador no deben actualizarse. De esta forma, será más propenso a predecir que la imagen es real cuando no lo es, ya que el generador sí que se estará entrenando mediante el modelo de la GAN a partir de vectores aleatorios con etiquetas indicando que todas las imágenes son reales (lo cual es, obviamente, falso). En resumen, podría decirse que el generador se entrena para "engañar" al discriminador.

Puesto que la GAN se enfrenta a un problema de clasificación binaria, la función de pérdida que se seleccionará es *entropía cruzada binaria* (1.51), pues se determina que es la que da lugar a los mejores resultados mediante prueba y error. Además, para mejorar la tarea de optimización se utiliza el algoritmo *Adam* [41], que sabemos que mejora la convergencia del SGD y es el más utilizado a la hora de llevar a cabo este tipo de aplicaciones, sobre todo si los conjuntos de datos son de gran tamaño.

Además, durante el entrenamiento de la GAN se puede observar en cada iteración la función de pérdida del discriminador y del generador; nos interesa que la segunda no aumenta demasiado mientras la primera tienda a cero. Una vez logrado, el generador habrá dejado de producir ruido y habrá creado imágenes prácticamente indistinguibles de las reales. También para evaluar el rendimiento del discriminador se pueden calcular ciertas métricas, como la exactitud (*accuracy*) en la predicción de imágenes, que mide el porcentaje de aciertos.

Pasemos ahora a describir en cada caso particular cómo construir los modelos del generador y del discriminador según convenga para la consecución del objetivo propuesto.

4.3.1. Dígitos numéricos

El objetivo es implementar una GAN con la que logremos generar dígitos manuscritos artificiales. Para entrenar el modelo utilizaremos el conjuntos de datos disponibles en TensorFlow, *MNIST* [48], que se trata de 70000 dígitos escritos a mano. Son imágenes de tamaño 28×28 en blanco y negro, por lo que solo habrá un canal: $(28, 28, 1)$.

En primer lugar, implementaremos el generador y el discriminador como redes neuronales prealimentadas utilizando como referencia el código [43] (*Chapter 3*). Es importante recalcar que muchos de los pasos que iremos dando a partir de este primer ejemplo se deberán repetir en el resto, por lo que llegado

el momento, detallaremos principalmente las diferencias que existan.

Por simplicidad, bastará utilizar como generador una red neuronal con una sola capa oculta que tome como entrada vectores aleatorios y produzca una imagen $28 \times 28 \times 1$. Se utilizará la función de activación *Leaky ReLU* (2.9) que, al contrario que en ReLU (2.8), no todas las entradas negativas tienen imagen nula, con lo que se consiguen gradientes positivos bajos y se evita así su desvanecimiento. En la capa de salida se seleccionará la función de activación *tangente hiperbólica* (2.7) puesto que su utilización, mediante prueba y error, se determina que produce imágenes con mayor nitidez.

Por su parte, el discriminador tendrá una estructura muy similar, aunque en casos más complejos esto no suele ocurrir. Las principales diferencias son que como entrada tomará una imagen $28 \times 28 \times 1$ y, como salida, dará una probabilidad, luego la función de activación debe ser la *sigmoide* (2.5), ya que su imagen está en el intervalo $[0, 1]$.

Respecto al entrenamiento de la GAN, las ideas son las anteriormente expuestas, pero hemos de notar que, debido a que hemos utilizado la función tangente hiperbólica (2.7) en la salida del generador y su imagen estará en el intervalo $[-1, 1]$, habrá que *normalizar* todos los píxeles de forma que se encuentren entre esos valores.

Además, habrá que elegir convenientemente los *hiperpárametros* que aparecen: el ratio de aprendizaje, el número de iteraciones y el tamaño de los minilotes, para lo cual ya hemos estudiado distintas técnicas.

Veamos los resultados obtenidos:

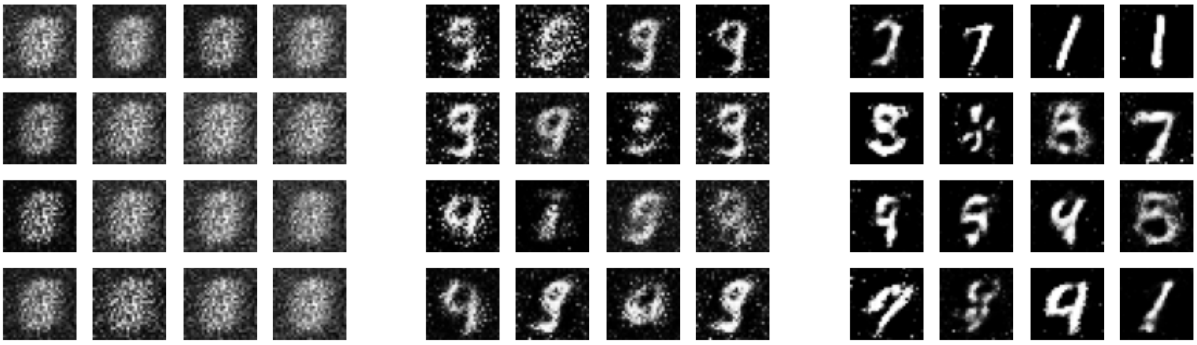

Figura 4.1: Ejemplos de dígitos generados a medida que avanzan las iteraciones, utilizando FFNNs.

Notemos que, al inicio, el generador solamente produce ruido y la calidad de las imágenes va mejorando progresivamente, a pesar de la simplicidad del modelo. Sin embargo, si comparamos con un ejemplo real de los utilizados para el entrenamiento, como el de la figura 4.2, vemos que los dígitos generados distan de ser perfectos.

Figura 4.2: Ejemplo real.

A continuación, empleando capas convolucionales demostraremos que los resultados son mejores, idea que proviene del éxitoso desarrollo de las DCGANs (*Deep Convolutional GAN*) [62].

Ya sabemos cómo operan estas capas pero, además, vimos que para generar imágenes era fundamental la *convolución traspuesta*. Por tanto, en este caso, el generador recibirá como entrada un vector aleatorio que será convertido a un vector tridimensional mediante una capa densa y posteriormente redimensionado de forma progresiva mediante dicha operación hasta alcanzar el tamaño de las imágenes que se quieren generar, $28 \times 28 \times 1$. Igual que en el caso anterior, se emplea la función de activación Leaky ReLU (2.9) y en la capa de salida, la tangente hiperbólica (2.7).

Por su parte, en lugar de convoluciones traspuestas, el discriminador realizará la *operación convolución* tal y como la describimos en su momento. Como entrada recibirá una imagen $28 \times 28 \times 1$ y, como salida, dará una probabilidad, por lo que la última capa será densa y con función de activación sigmoide (2.5); el resto, utilizarán la función Leaky ReLU (2.9).

Además, es importante destacar que en ambos casos las operaciones llevadas a cabo por las capas convolucionales se ajustan convenientemente seleccionando los núcleos y los pasos dependiendo de las dimensiones deseadas. Por su parte, el *relleno* lo ajustaremos mediante *same* en todos los casos, lo cual implica que se añaden ceros para que la salida tenga el mismo tamaño que la entrada.

Asimismo, como novedad, se aplica *normalización por lotes* [36]. En cuanto al entrenamiento de la GAN no hay cambios con respecto a lo anteriormente expuesto. A modo de curiosidad y para intuir mejor cómo las CNNs procesan los dígitos, se puede probar a ejecutar la animación [34].

Finalmente, comparando la figura 4.3 con las figuras 4.1 y 4.2, concluimos que los ejemplos generados ahora sí que son muchos de ellos prácticamente indistinguibles.

Figura 4.3: Ejemplos de dígitos generados utilizando CNNs.

4.3.2. Rostros humanos

El propósito de esta sección es conseguir entrenar una GAN que genere rostros lo más indistinguibles posibles de los reales. Para ello, sacando partido de las ventajas ya conocidas en tareas de visión computacional de las CNNs y de que acabamos de probar que utilizándolas tanto en el generador como en el discriminador de las GANs se obtienen mejores resultados, pasemos a aplicarlo al conjunto de datos *CelebA* [13], que se trata de más de 200000 imágenes de rostros de celebridades, a color y de tamaño 218×178. Nos inspiraremos en el código de Chollet [14] y también en [45].

Se trata de fotografías a gran escala, por lo que el costo computacional sería enorme si no las redujéramos, y no tenemos medios para afrontarlo eficientemente. Por tanto, las redimensionaremos a un tamaño $64 \times 64 \times 3$, que ya supone una diferencia notable con respecto al caso de los dígitos numéricos y lo veremos reflejado en un importante aumento del tiempo de cómputo, que evidencia una vez más nuestras limitaciones a la hora de llevar a cabo la parte práctica del trabajo.

Con respecto a la estructura del generador y del discriminador, es muy similar a la descrita en el caso anterior cuando se han incluido CNNs. La diferencia reside en especificar el tamaño de las nuevas entradas y en que en el discriminador ahora se recomienda utilizar, como técnica de regularización, el abandono (*dropout*) que ya describimos. Entonces, a la hora de evaluar el rendimiento de la GAN en este caso, si viéramos que la pérdida del discriminador comenzara a tender a cero mientras se dispara la del generador, deberíamos probar a aumentar el ratio de abandono del discriminador, así como a reducir su ratio de aprendizaje.

Los resultados obtenidos pueden verse en las figuras 4.4 y 4.7. Es evidente que distan de ser tan realistas como los rostros que se pueden obtener con el software *This person does not exist* [77], pero tal calidad queda fuera de nuestro alcance tanto por limitaciones técnicas como por la dificultad intrínseca que supone el hecho de que entrenar una GAN sea un proceso dinámico, en lugar de solamente un problema de minimización de una determinada función de perdida. Esto involucra una ingente cantidad de parámetros e hiperparámetros que se deben ajustar la mayoría de las ocasiones mediante astutos trucos que se adquieren con la práctica, para lo cual hemos consultado las referencias citadas. En cualquier caso, nos sirve como ejemplo de aplicación de las GANs en la generación de imágenes y para poner en práctica todo lo anteriormente expuesto.

Figura 4.4: Ejemplos de caras generadas utilizando el código de referencia [45].

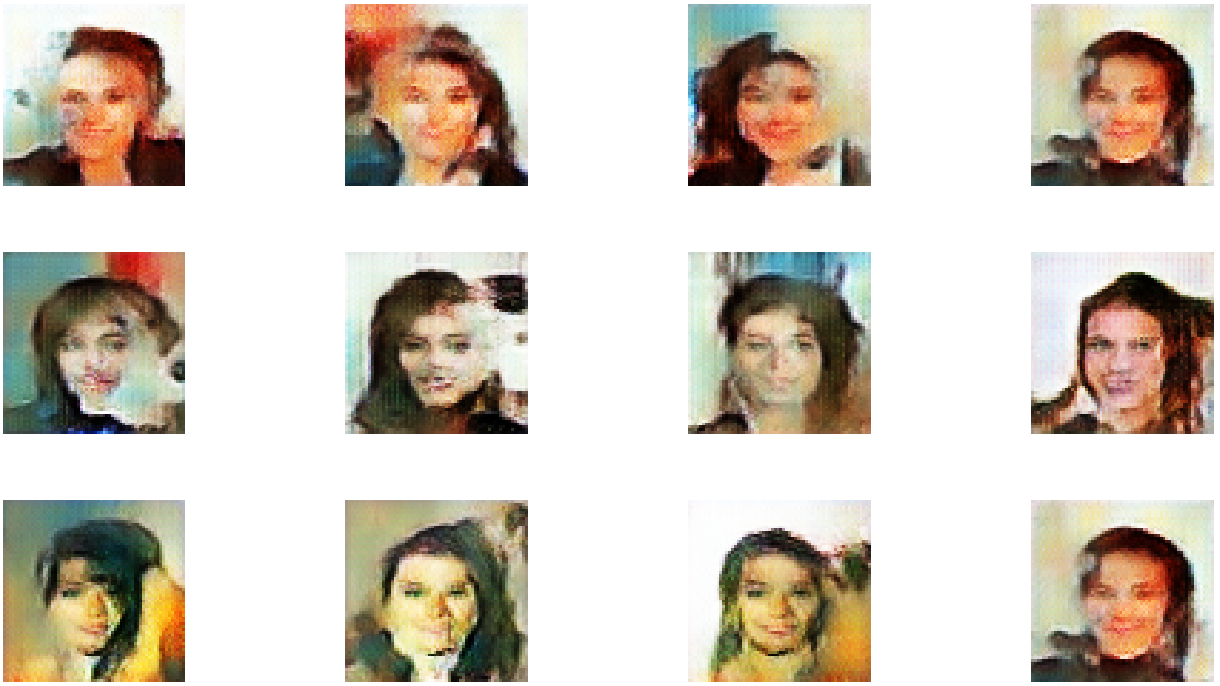

Figura 4.7: Ejemplos de caras generadas utilizando el codigo de referencia [14].

Conclusiones

Las Redes Generativas Adversariales han supuesto un hito en la historia del aprendizaje profundo y se está llevando a cabo una interesante labor investigadora al respecto, conscientes de las potentes aplicaciones que pueden llegar a tener.

Para sacarles el máximo partido es de vital importancia lograr entenderlas desde el punto de vista teórico de manera rigurosa, pues, como hemos visto, su implementación es de enorme dificultad y la falta de garantías conlleva delegar demasiadas responsabilidades en trucos que se adquieren mediante prueba y error.

En esta tarea, es evidente que las Matemáticas juegan un papel fundamental, pues la adaptación de resultados clásicos hemos visto cómo ha evolucionado hasta llegar a proporcionar explicaciones precisas sobre estas tecnologías punteras; también a nivel práctico, pues las líneas de código necesarias no han supuesto más que una forma de escribir lo que ya habíamos formulado.

A este respecto, hemos visto cómo el clásico problema estadístico de discriminación está presente en las GANs de forma más compleja y requiere la utilización de ciertas divergencias y métricas. En particular, la minimización de la distancia de Jensen-Shannon, que proviene de una simetrización de la divergencia de Kullback, íntimamente relacionada con el también clásico método de estimación de máxima verosimilitud. Pero esto solo ocurre como tal en el caso ideal de que las clase de discriminadores no esté restringida; en la práctica, no es así, pero se demuestra que gracias a ciertas propiedades, el buen rendimiento está garantizado.

En cuanto a la tarea de aprendizaje, formulado en términos estadísticos, se trata de la minimización del riesgo empírico, es decir, de optimizar la función que cuantifica el error en las predicciones a partir de los datos disponibles. Dicho problema de optimización se resuelve actualizando los parámetros del riesgo empírico mediante el algoritmo de Descenso de Gradiente, pero esto no es tarea fácil cuando interviene una gran cantidad de parámetros, tal y como ocurre en el caso de las redes neuronales que, en consecuencia, deben implementar el algoritmo de back-propagation para computar los gradientes. En particular, se ha demostrado que a la hora de llevar a cabo la tarea de aprendizaje, las GANs utilizan una función objetivo muy similar a la de la regresión logística.

Para formular el funcionamiento del generador y del discriminador en las GANs en el citado problema de aprendizaje, son fundamentales las redes neuronales. Más concretamente, hemos comprobado en la parte de implementación práctica que a la hora de tratar con imágenes, es conveniente emplear CNNs en lugar de FFNNs. Así, se han generado imágenes de dígitos numéricos y de rostros humanos, en la medida de nuestras posibilidades, debido tanto a las dificultades intrínsecas en el entrenamiento de las GANs como a limitaciones técnicas.

En definitiva, se espera haber proporcionado en este trabajo los conocimientos necesarios para comprender el funcionamiento de las GANs y de las variantes en que se está investigando, además de haber despertado el interés al respecto como una puntera aplicación de las Matemáticas.

Apéndice A

Teorema de Radon-Nikodym

Definición A.1. Sea (Ω, σ) un espacio medible. Se dice que una medida μ es *absolutamente continua con respecto a* otra medida ν y se denota $\mu \ll \nu$ cuando $\nu(A) = 0 \Rightarrow \mu(A) = 0$, para todo $A \in \sigma$.

Teorema A.1. ***(Teorema de Radon-Nikodym)***
Sea (Ω, σ, ν) un espacio de medida σ-finito, μ una medida compleja definida en el espacio medible (Ω, σ) y absolutamente continua respecto de ν. Existe una función integrable $f \in \mathcal{L}^1(\nu, \mathbf{C})$ que cumple:

$$\mu(A) = \int_A f d\nu, \text{ para todo } A \in \sigma.$$

Se dice que f es la derivada de Radon-Nikodym de μ respecto de ν. Cualquier otra derivada es igual en casi todo Ω a esta y se le denota por $\frac{d\mu}{d\nu}$.

Demostración.

Se puede consultar en [64] (pág. 138, Teorema 6.10-b). □

Este teorema también es aplicable, en particular, a probabilidades, ya que estas son medidas σ-finitas tales que su integral en todo el espacio es 1. Esta aplicación concreta es la que interesa en el desarrollo teórico del presente trabajo.

Apéndice B

Procesos sub-gaussianos

Definición B.1. Sea (T, d) un espacio métrico. El **número de cubrimiento** se denota por $N(T, d, \varepsilon)$ y representa el menor número de bolas de radio ε necesarias para cubrir por completo T.

Por tanto, podemos interpretar $N(T, d, \varepsilon)$ como una medida del grado de compacidad de (T, d).

Nótese que siempre tenemos que $N(T, d, \varepsilon) = 1$ cuando $\varepsilon > diam(T)$.

Definición B.2. Un proceso aleatorio $\{X_t\}_{t\in T}$ en el espacio métrico (T, d) se denomina **proceso sub-gaussiano** si $\mathbb{E}[X_t] = 0$ y

$$\mathbb{E}\left[e^{\lambda\{X_t - X_x\}}\right] \leq e^{\lambda^2 d(t,s)^2/2} \quad \forall t, s \in T, \quad \lambda \geq 0$$

Definición B.3. Un proceso aleatorio $\{X_t\}_{t\in T}$ se conoce como **proceso separable** si existe un conjunto numerablee $T_0 \subset T$ tal que

$$X_t \in \lim_{\substack{s\to t_0 \\ s\in T_0}} X_s \quad \forall t \in T c.s.,$$

donde $x \in \lim_{s\to t} x_s$ significa que hay una sucesión $s_n \to t$ tal que $x_{s_n} \to x$.

La suposición de separabilidad es meramente técnica y se satisface trivialmente casi siempre. Por ejemplo, si $t \mapsto X_t$ es continua *c.s.*, podemos tomar T_0 como cualquiera subconjunto contable denso en T.

Teorema B.1. *Sea $\{X_t\}_{t\in T}$ un proceso aleatorio sub-gaussiano separable en el espacio métrico (T, d). Entonces, tenemos:*

$$\mathbb{E}\left[\sup_{t\in T} X_t\right] \leq 6 \sum_{k\in\mathbb{Z}} 2^{-k} \sqrt{\log N\left(T, d, 2^{-k}\right)}.$$

Demostración.

Se puede consultar en [33] (pág. 133, Teorema 5.24). □

Corolario B.1.1. *(**Cota de entropía de Dudley**) Sea $\{X_t\}_{t\in T}$ un proceso aleatorio sub-gaussiano separable en el espacio métrico (T, d). Entonces, tenemos:*

$$\mathbb{E}\left[\sup_{t\in T} X_t\right] \leq 12 \int_0^\infty \sqrt{\log N(T, d, \varepsilon)} d\varepsilon.$$

Demostración.

Se puede consultar en [33] (pág. 134, corolario 5.25). □

Bibliografía

[1] M. Arjovsky, S. Chintala y L. Bottou, *Wasserstein GAN*, 2017. arXiv: 1701.07875 [stat.ML].

[2] O. Beaumont, J. Herrmann, G. Pallez y A. Shilova, "Optimal Memory-aware Backpropagation of Deep Join Networks," *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, vol. 378, 2019.

[3] Y. Bengio, *Practical recommendations for gradient-based training of deep architectures*, 2012. arXiv: 1206.5533.

[4] M. Bettoni, G. Urgese, Y. Kobayashi, E. Macii y A. Acquaviva, "A Convolutional Neural Network Fully Implemented on FPGA for Embedded Platforms," en *2017 New Generation of CAS (NGCAS)*, 2017, págs. 49-52.

[5] G. Biau, C. Benoît, M. Sagnier y U. Tanielian, "Some theoretical properties of GANs," *The Annals of Statistics*, vol. 48, n.º 3, págs. 1539-1566, 2020.

[6] A. Brock, J. Donahue y K. Simonyan, *Large Scale GAN Training for High Fidelity Natural Image Synthesis*, 2019. arXiv: 1809.11096 [cs.LG].

[7] G. Brown, "Basic principles for construction and application of discriminators," *Jour. Clinical Psych.*, vol. 6, págs. 58-61, 1950.

[8] M. Brundage, S. Avin, J. Clark, H. Toner, P. Eckersley, B. Garfinkel, A. Dafoe, P. Scharre, T. Zeitzoff, B. Filar, H. Anderson, H. Roff, G. C. Allen, J. Steinhardt, C. Flynn, S. Ó. hÉigeartaigh, S. Beard, H. Belfield, S. Farquhar, C. Lyle, R. Crootof, O. Evans, M. Page, J. Bryson, R. Yampolskiy y D. Amodei, *The Malicious Use of Artificial Intelligence: Forecasting, Prevention, and Mitigation*, 2018. arXiv: 1802.07228 [cs.AI].

[9] S. Bubeck, *Convex Optimization: Algorithms and Complexity*, 2015. arXiv: 1405.4980 [math.OC].

[10] Y. Cai y L.-H. Lim, *Distances between probability distributions of different dimensions*, 2020. arXiv: 2011.00629 [math.ST].

[11] V. Canals, A. Morro, A. Oliver, M. L. Alomar y J. L. Rosselló, "A New Stochastic Computing Methodology for Efficient Neural Network Implementation," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 27, n.º 3, págs. 551-564, 2016.

[12] G. M. Caron, J. Pinto, G. Farnadi y M. Bronzi, *Deep Learning Essentials*, Course number: IVADO-DL1EN.

[13] *Celeba. Tensorflow datasets,* dirección: https://www.tensorflow.org/datasets/catalog/celeb_a (visitado 25-06-2021).

[14] F. Chollet. (2019). "DCGAN to generate face images," dirección: https://keras.io/examples/generative/dcgan_overriding_train_step/ (visitado 25-06-2021).

[15] ——, *Deep Learning with Python*. Manning, 2018.

[16] ——, *Deep Learning with R*. Manning, 2018.

[17] J. Cramer, "Logit Models from Economics and Other Fields: The origins and development of the logit model," 2003.

[18] Y. le Cun, "Generalization and Network Design Strategies," Department of Computer Science, University of Toronto, inf. téc., 1989.

[19] ——, (2018). "Quora Session: What are some recent and potentially upcoming breakthroughs in deep learning?" Dirección: https://www.quora.com/What-are-some-recent-and-potentially-upcoming-breakthroughs-in-deep-learning (visitado 25-06-2021).

[20] Y. le Cun, L. Bottou, G. Orr y K.-R. Müller, "Efficient BackProp," 2000.

[21] P. P. Ebner y A. Eltelt, *Audio inpainting with generative adversarial network*, 2020. arXiv: 2003.07704 [eess.AS].

[22] D. Eck y S. J., "Learning the long-term structure of the blues," inf. téc., 2002, págs. 284-289.

[23] D. Endres y J. Schindelin, "A new metric for probability distributions," *IEEE Trans. Inform. Theory*, vol. 49, págs. 1858-1860, 2003.

[24] J. Engel, K. K. Agrawal, S. Chen, I. Gulrajani, C. Donahue y A. Roberts, *GANSynth: Adversarial Neural Audio Synthesis*, 2019. arXiv: 1902.08710 [cs.SD].

[25] B. Erickson, P. Korfiatis, Z. Akkus, T. Kline y P. Kenneth, "Toolkits and Libraries for Deep Learning," *Journal of Digital Imaging*, vol. 30, págs. 400-405, 2017.

[26] *File:MaxpoolSample2.png, Computer Science Wiki*, dirección: https://computersciencewiki.org/index.php/File:MaxpoolSample2.png (visitado 25-06-2021).

[27] F. Gao, Y. Yang, J. Wang, J. Sun, E. Yang y H. Zhou, "A Deep Convolutional Generative Adversarial Networks (DCGANs)-Based Semi-Supervised Method for Object Recognition in Synthetic Aperture Radar (SAR) Images," *Remote Sensing*, vol. 10, n.º 6, 2018.

[28] X. Gong, S. Chang, Y. Jiang y Z. Wang, "AutoGAN: Neural Architecture Search for Generative Adversarial Networks," en *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, 2019.

[29] I. Goodfellow, "Multidimensional, Downsampled Convolution for Autoencoders," inf. téc., 2010.

[30] I. Goodfellow, Y. Bengio y A. Courville, *Deep Learning*. MIT Press, 2016, http://www.deeplearningbook.org.

[31] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, X. Bing, D. Warde-Farley, S. Ozair, A. Courville e Y. Bengio, "Generative Adversarial Nets," *Advances in Neural Information*, vol. 2, págs. 2672-2680, 2014.

[32] A. Graves, A. Mohamed y G. Hinton, "Speech recognition with deep recurrent neural networks," Department of Computer Science, University of Toronto, inf. téc., 2013.

[33] R. van Handel, "Probability in High Dimension," inf. téc., 2016, APC 550 Lecture Notes. Princeton University, págs. 120-134.

[34] A. W. Harley, "An Interactive Node-Link Visualization of Convolutional Neural Networks, 3D Visualization of a CNN," en *ISVC*, 2015, págs. 867-877. dirección: https://www.cs.ryerson.ca/~aharley/vis/conv/.

[35] T. Hastie, R. Tibshirani y J. Friedman, *The Elements of Statistical Learning*, 2.ª ed. Springer, 2008, http://web.stanford.edu/~hastie/Papers/ESLII.pdf.

[36] S. Ioffe y C. Szegedy, *Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift*, 2015. arXiv: 1502.03167 [cs.LG].

[37] A. Jalal, A. Ilyas, C. Daskalakis y A. G. Dimakis, *The Robust Manifold Defense: Adversarial Training using Generative Models*, 2019. arXiv: 1712.09196 [cs.CV].

[38] H. Jang, A. Park y K. Jung, "Neural Network Implementation Using CUDA and OpenMP," en *2008 Digital Image Computing: Techniques and Applications*, 2008, págs. 155-161.

[39] T. Karras, S. Laine, M. Aittala, J. Hellsten, J. Lehtinen y T. Aila, *Analyzing and Improving the Image Quality of StyleGAN*, 2020. arXiv: 1912.04958 [cs.CV].

[40] *Keras: the Python deep learning API*. dirección: https://keras.io/ (visitado 25-06-2021).

[41] D. Kingma y J. Ba, "Adam: A Method for Stochastic Optimization," *International Conference on Learning Representations*, 2014.

[42] D. P. Kingma y M. Welling, *Auto-Encoding Variational Bayes*, 2014. arXiv: 1312.6114 [stat.ML].

[43] J. Langr y V. Bok, *GANs in action*. Manning, 2019.

[44] T. Lucas, "Deep Generative models: over-generalisation and mode-dropping," 2020, cap. 2.9.

[45] W. Maffione. (2019). "DCGAN with Keras," dirección: https://www.kaggle.com/waltermaffy/dcgan-with-keras (visitado 25-06-2021).

[46] S. Mallela, I. Dhillon y R. Kumar, "A divisive information-theoretic feature clustering algorithm for text classification," *Journal of Machine Learning Research*, vol. 3, págs. 1265-1287, 2003.

[47] R. Mello y M. Ponti, *Machine Learning. A practical Approach on the Statistical Learning Theory.* American Mathematical Society, 2003.

[48] *MNIST. Tensorflow datasets,* dirección: https://www.tensorflow.org/datasets/catalog/mnist (visitado 25-06-2021).

[49] P. Moreno, P. Ho y V. N., "A kullback-leibler divergence based kernel for svm classification in multimedia applications," inf. téc., 2004, HP Laboratories, Cambridge, MA, págs. 79-86.

[50] E. Mourier, "Étude du choix entre deux lois de probabilité," *C.R. Acad. Sci. Paris*, vol. 223, págs. 712-714, 1946.

[51] G. Muhammad y M. Melhem, "Pathological voice detection and binary classification using MPEG-7 audio features," *Biomedical Signal Processing and Control*, vol. 11, págs. 1-9, 2014, ISSN: 1746-8094.

[52] K. P. Murphy, *Machine Learning. A Probabilistic Perspective.* Mit Press, 2012, http://noiselab.ucsd.edu/ECE228/Murphy_Machine_Learning.pdf.

[53] N. Nagwani y S. A., "SMS spam filtering and thread identification using bi-level text classification and clustering techniques," *Journal of Information Science*, vol. 43, págs. 75-87, 1 2017.

[54] G. Neubig, C. Dyer, Y. Goldberg, A. Matthews, W. Ammar, A. Anastasopoulos, M. Ballesteros, D. Chiang, D. Clothiaux, T. Cohn, K. Duh, M. Faruqui, C. Gan, D. Garrette, Y. Ji, L. Kong, A. Kuncoro, G. Kumar, C. Malaviya, P. Michel, Y. Oda, M. Richardson, N. Saphra, S. Swayamdipta y P. Yin, *DyNet: The Dynamic Neural Network Toolkit*, 2017.

[55] D. Nie, R. Trullo, J. Lian, C. Petitjean, S. Ruan, Q. Wang y D. Shen, "Medical Image Synthesis with Context-Aware Generative Adversarial Networks," en *Medical Image Computing and Computer Assisted Intervention MICCAI 2017. Lectures in Computer Science.* Springer, 2017, vol. 10435, págs. 417-425.

[56] H. Nielsen, H. Madsen y T. Nielsen, "Using quantile regression to extend an existing wind power forecasting system with probabilistic forecasts," *Wind Energy*, vol. 9, págs. 95-108, 2006.

[57] Y. Oshima, H. Shinzawa, T. Takenaka, C. Furihata y S. Hidetoshi, "Discrimination analysis of human lung cancer cells associated with histological type and malignancy using Raman spectroscopy," *Journal of Biomedical Optics*, vol. 15, 1 2010.

[58] C. Perlich, "Learning Curves in Machine Learning," en *Encyclopedia of Machine Learning and Data Mining*, C. Sammut y G. I. Webb, eds. Boston, MA: Springer US, 2017, págs. 708-711.

[59] Phung y Rhee, "A High-Accuracy Model Average Ensemble of Convolutional Neural Networks for Classification of Cloud Image Patches on Small Datasets," *Applied Sciences*, vol. 9, pág. 4500, 2019.

[60] *R interface to Keras,* dirección: https://keras.rstudio.com/ (visitado 25-06-2021).

[61] D. R., *Graph Theory.* Springer, 217.

[62] A. Radford, L. Metz y S. Chintala, *Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks*, 2016. arXiv: 1511.06434 [cs.LG].

[63] M. Rezapour, S. Nazneen y K. Ksaibati, "Application of deep learning techniques in predicting motorcycle crash severity," *Wiley Online Library*, pág. 5, 2020.

[64] W. Rudin, *Análisis real y complejo*, 3.ª ed. McGraw Hill, 1988.

[65] D. Rumelhart, G. Hinton y R. Williams, "Learning representations by back-propagating errors," *Nature*, 1986.

[66] K. S. y R. Leibler, "On information and sufficiency," *Ann. Math. Statistics*, vol. 3, págs. 79-86, 1951.

[67] D. Saxena y J. Cao, *Generative Adversarial Networks (GANs): Challenges, Solutions, and Future Directions*, 2020. arXiv: 2005.00065 [cs.LG].

[68] S. Shalev-Shwartz y S. Ben-David, *Understanding Machine Learning: from Theory to Algorithms.* Cambridge University Press, 2014, http://www.cs.huji.ac.il/~shais/UnderstandingMachineLearning.

[69] S. Skansi, *Introduction to Deep Learning.* Springer, 2018.

[70] M. Sonka, H. V. y B. R., *Image Processing, Analysis, and Machine Vision*, 4.ª ed. Cengage Learning, 2015.

[71] N. Srivastava, G. Hinton, K. A., I. Sutskever y R. Salakhutdinov, "Dropout: A Simple Way to Prevent Neural Networks from Overfitting," vol. 15, págs. 1929-1958, 2014.

[72] T. Szandała, "Review and Comparison of Commonly Used Activation Functions for Deep Neural Networks," Wroclaw University of Science y Technology, Poland, inf. téc., 2020.

[73] M. T., M. Karafiát, L. Burget, E. J. y S. Khudanpur, "Recurrent neural network based language model," *INTERPSEECH*, 2010.

[74] A. B. Tsybakov, *Introduction to Nonparemetric Estimation*. Springer, 2009.

[75] A. M. Turing, "I.—COMPUTING MACHINERY AND INTELLIGENCE," *Mind*, vol. LIX, n.º 236, págs. 433-460, 1950.

[76] C. Villani, *Topics in Optimal Transportation*. American Mathematical Society, 2003, cap. 7.

[77] P. Wang. (2019). "This person does not exist," dirección: https://thispersondoesnotexist.com/ (visitado 25-06-2021).

[78] L. Wasserman, *All of Statistics. A Concise Course in Statistical Inference*. Springer, 2004.

[79] B. Welch, "Note on discriminant functions," *Biometrika*, vol. 31, págs. 218-219, 1939.

[80] T. Xu, P. Zhang, Q. Huang, H. Zhang, Z. Gan, X. Huang y X. He, *AttnGAN: Fine-Grained Text to Image Generation with Attentional Generative Adversarial Networks*, 2017. arXiv: 1711.10485 [cs.CV].

[81] P. Yao, H. Wu, B. Gao, J. Tang, Q. Zhang, W. Zhang, J. J. Yang y H. Qian, "Fully hardware-implemented memristor convolutional neural network," *Nature*, vol. 577, págs. 641-646, 2020.

[82] X. Yu, Y. Qu y M. Hong, "Medical Image Synthesis with Context-Aware Generative Adversarial Networks," en *Pattern Recognition and Information Forensics. ICPR 2018. Lecture Notes in Computer Science*. Springer, 2018, vol. 11188, págs. 66-75.

[83] J.-Y. Zhu, T. Park, P. Isola y A. A. Efros, *Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks*, 2020. arXiv: 1703.10593 [cs.CV].

[84] J. Zietz y E. Zietz, "Determinants of House Prices: A Quantile Regression Approach," *The Journal of Real Estate Finance and Economics*, vol. 37, págs. 317-333, 2008.