Universidad de Valladolid

FACULTAD DE CIENCIAS

TRABAJO FIN DE GRADO

Grado en Matemáticas

Métodos generativos basados en transporte óptimo

Autor: Martínez Martínez, Manuel
Tutor: del Barrio Tellado, Eustasio
Año 2023

Métodos generativos basados en transporte óptimo

Manuel Martínez Martínez

Dedicado a mis padres:
Agustín y María Jesús;
mis hermanos:
José Antonio, Santiago,
Pilar, Juan e Inés;
y a todos mis amigos.

Resumen

En los últimos 10 años hemos visto grandes avances en el campo de la inteligencia artificial. Estas aplicaciones emergentes no serían posibles sin el desarrollo previo de los autocodificadores, redes capaces de generar objetos nuevos extrapolando distribuciones de probabilidad de conjuntos de datos, utilizando técnicas que reducen la dimensión de los datos. Esto nos lleva a preguntarnos cómo funcionan estas redes y cómo es la realidad actual de las aplicaciones más punteras.

Para entender mejor el estado en el que se encuentra la tecnología, este trabajo presenta las redes que conforman la base de los métodos actuales, utilizados en la actualidad para la generación de elementos escritos y visuales. En el trabajo se explica el funcionamiento de los autocodificadores y los resultados que produjeron implementaciones de las mismas basadas en el transporte óptimo. Esto demostrará lo mucho que se ha avanzado en poco tiempo, además de ser una buena introducción para todo aquel que quiera entender el funcionamiento de las redes de aprendizaje profundo más punteras.

Palabras clave: aprendizaje automático, aprendizaje profundo, redes neuronales, autocodificadores, *WAE*, autocodificadores de *Wasserstein*.

Abstract

The last 10 years have seen great advances in the field of artificial intelligence. These emerging applications would not be possible without the prior development of autoencoders, networks capable of generating new objects by extracting probability distributions from datasets, using dimensional reduction techniques over the sets. This leads us to ask ourselves how these networks work and what is the current reality of state-of-the-art applications.

To better understand the state-of-the-art, this paper presents this networks, which form the basis of the current methods used today for the generation of written and visual elements. The paper explains how autoencoders work and implementations of them based on optimal transport. This will demonstrate how much progress has been made in a short time, as well as being a good introduction for anyone who wants to understand the behavior of state-of-the-art deep learning networks.

Keywords: machine learning, deep learning, neural networks, autoencoders, *WAE*, *Wasserstein* autoencoders.

Índice general

Índice de figuras

Notación, siglas y abreviaturas

Notación

Pese a que se definirá lo que significa cada una de las funciones o elementos que se utilicen, en general se siguen unas reglas generales de notación:

- $\mathbf{x}$ o x representarán vectores o escalares, respectivamente.
- f se utilizará para describir funciones lineales o no lineales, y en una sola variable o en varias (lo cual se mencionará durante la definición). Además, reservamos la letra ℓ para definir a las funciones de pérdida.
- A representará matrices, que se definirán, en la mayoría de casos, de la siguiente manera: $A = (a_{ij})_{1:n,1:m}$ (en el caso de matrices $n \times m$). En el caso de que la matriz sea cuadrada se escribirá $A = (a_{ij})_{1:n}$.
- $\mathcal{A}$ servirá para denotar conjuntos, como lo son los conjuntos de datos observados (atributos), los conjuntos de predicciones (etiquetas) y las familias de funciones.
- $\nabla_{\mathbf{x}} f$ representará el gradiente de una función f respecto de las componentes indicadas por $\mathbf{x}$. En el caso de ∇f hablamos del gradiente de f respecto de todas sus variables.
- $\odot$ será utilizado para representar el producto de Hadamard de matrices, es decir, el producto por componentes.

Generalmente, F denota la distribución, y lo adecuado es utilizar f como su densidad cuando esta existe. Sin embargo, por mantener una notación acorde a la que veremos en el trabajo, escribiremos las densidades con p, aunque su distribución pueda denotarse por F.

Siglas y abreviaturas

Vamos a detallar una lista de las siglas que más se utilizarán a lo largo del trabajo. Aún así, estas se introducirán o recordarán cuando se crea conveniente:

- *Adaptive Moment Estimation*: *ADAM*.
- Análisis de componentes principales (*Principal Component Analysis*): *PCA*.
- Aprendizaje automático (*Machine learning*): *ML*.
- Aprendizaje profundo (*Deep learning*): *DL*.
- Autocodificador (*Autoencoder*): *AE*.
- Autocodificador variacional (*Variational autoencoder*): *VAE*.
- Autocodificador variacional de *Wasserstein* (*Wasserstein variational autoencoder*): *WAE*.
- *Evidence lower bound*: *ELBO*.
- Inferencia variacional (*Variational inference*): *IV* o *VI*.
- Inferencia variacional estocástica (*Stochastic variational inference*): *SVI*.
- Inteligencia Artificial: IA.
- *Markov Chain MonteCarlo*: *MCMC*.
- Máquinas de vector-soporte (*Support Vector Machine*): *SVM*.
- Minimización del riesgo empírico (*Empirical risk minimization*): *ERM*.
- *Rectified Linear Unit*: *ReLU*.
- Red neuronal: RN (RRNN para el plural).
- Red neuronal artificial (*Artificial Neural Network*): *ANN*.
- Redes generativas adversariales (*Generative Adversarial Network*): *GAN*.
- Unidad de procesamiento gráfico (*Graphical Processing Unit*): *GPU*.

Parte I

Introducción y conceptos generales

Capítulo 1

Introducción

1.1. IA en la actualidad

En plena era de la información es natural encontrar noticias, libros, artículos científicos y profesionales e incluso opiniones contrapuestas sobre cualquier avance tecnológico, todo gracias a Internet.

Aparte del hecho de que la información nunca haya sido tan accesible como lo es hoy en día, la Red ha permitido a miles de millones de usuarios compartir información en diversos formatos: vídeos, imágenes, texto, audio... Esto ha aumentado sustancialmente la cantidad de datos de que se dispone y ha ayudado al desarrollo científico fuertemente, pues en la actualidad es mucho más sencillo divulgar el conocimiento a través de páginas web, blogs, canales de *YouTube* y muchos medios más.

Internet se ha convertido en un vehículo de información a gran escala de forma masiva, información que es codificada, transmitida y disponible para todos sus usuarios. La mayoría de esta información tiene un formato predefinido y, aunque se representen de distintas formas, es referida generalmente como datos.

Centremos nuestra atención en un tipo muy específico de dato, como son las imágenes. Desde el inicio de la humanidad, las formas mayoritarias de expresión como especie han sido el dibujo, el arte, y más recientemente la fotografía e imágenes digitales. En la actualidad, las imágenes son uno de los formatos de dato más utilizados para la transmisión de información en Internet.

A su vez, las ramas, más populares recientemente, de *Big Data* y *Machine Learning* aprovechan la accesibilidad a los datos disponibles en esta Red para cumplir con objetivos de análisis masivos de datos, clasificación de objetos por clases, reconocimiento de objetos en imágenes y mucho más.

En 2014 se presentó el artículo [18], que rompía con algunos de los esquemas establecidos en el campo de la IA hasta el momento. Dicho artículo presentaba la idea de las redes generativas adversariales (GAN). Estas redes, para la mayoría de la gente, son un misterio y nos acercan a la ciencia ficción, pues son capaces de generar texto como si habláramos con un humano, producir imágenes bajo petición, e incluso algunas son capaces de hacer ambas cosas a la vez.

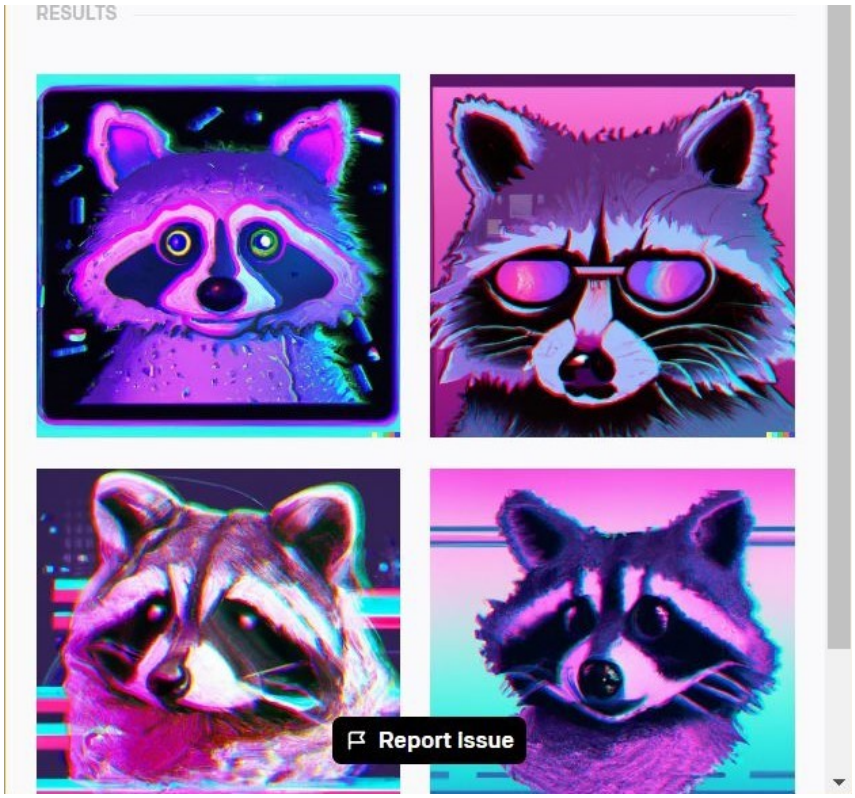

Figura 1.1: Ejemplos de imágenes generadas bajo petición con *DALL-E 2*

Alrededor de Octubre de 2022 se ha podido observar un creciente interés en estas redes generativas, más específicamente, aquellas que son capaces de extrapolar características de los datos u objetos con los que se trabaja y generar nuevos objetos similares (como lo son *DALL-E 2* o *Chat-GPT*). Se podría pensar que hemos llegado a un punto en el que verdaderamente hemos infundido de creatividad y libertad a nuestros agentes inteligentes, pero, como veremos en este proyecto, los conceptos detrás de estas redes se basan en formas de tomar muestras de probabilidades y medidas de similitud, y no en cualidades típicamente humanas.

El comienzo de estas redes se basa en un único formato, en el que la red generativa muestra el nuevo contenido, pero no interpretando el lenguaje natural, no pudiendo generar contenido "a la carta".

Actualmente esta barrera parece superada hace mucho tiempo por el paradigma de la multimodalidad, ahora en boca de todos. Esta sensación viene, sencillamente, de la velocidad a la que se han comenzado a desarrollar nuevas e imponentes herramientas como lo son *GPT-4* de *OpenAI* o su implementación, anunciada en una conferencia online y con información disponible en [1], que pretende revolucionar la manera de enfocar el trabajo reduciendo la carga de trabajo mediante un asistente automático.

Estas nuevas herramientas utilizan el aprendizaje multimodal, como ya hemos mencionado, del cual hablaremos como potenciación de las redes generativas y se puede obtener

más información leyendo [6] o [2].

En este trabajo pondremos la atención en un tipo específico de estas redes generativas, los autocodificadores, aunque la arquitectura de los *Wasserstein Autoencoders* serán el tema central del proyecto, pues son capaces de generar contenido a partir de la resolución de un problema de optimización de distancias y un proceso de decodificación. Para ello, primero recorreremos una serie de conceptos básicos que son necesarios para la buena comprensión de su funcionamiento, y luego se describirán estas redes. Para terminar, se mostrarán los resultados de entrenar y ejecutar distintos modelos generativos *WAE*. Centramos la atención en estas redes autocodificadoras dado que, computacionalmente, las redes generativas adversariales son más caras, lo que es su principal desventaja a la hora de competir con los autocodificadores.

1.2. Objetivos del trabajo

El principal objetivo de este trabajo es explicar detalladamente el funcionamiento de los autocodificadores variacionales, en especial el *WAE*, basado en el transporte óptimo.

Para ello se han definido las siguientes directrices que se han de tratar a lo largo del trabajo, de modo que los conceptos se desarrollen completamente, además de proporcionar al trabajo una clara línea sobre la que trabajar. Para mantener dicha línea de una manera más estricta se consideran los siguientes objetivos fundamentales:

- **OBJ-1** Presentación de los conceptos básicos.
  - **OBJ-1.1** Introducción de los distintos tipos de aprendizaje.
  - **OBJ-1.2** Descripción de medidas del error, funciones de pérdida y riesgo.
  - **OBJ-1.3** Explicación del núcleo de las redes profundas, las redes neuronales.
  - **OBJ-1.4** Algoritmos para el entrenamiento de una red neuronal.
- **OBJ-2** Introducción a la inferencia variacional.
  - **OBJ-2.1** Descripción del *ELBO*.
  - **OBJ-2.2** Introducción al funcionamiento de la *VI*.
  - **OBJ-2.3** Problemas de la *VI*.
- **OBJ-3** Descripción de los autocodificadores variacionales y del *WAE*.
  - **OBJ-3.1** Detalle del proceso de funcionamiento de un autocodificador.
  - **OBJ-3.2** Introducción y explicación de los autocodificadores variacionales, la premisa detrás del *WAE*.
  - **OBJ-3.3** Descripción teórica del funcionamiento de una red *WAE*.
  - **OBJ-3.4** Implementación y prueba de dos tipos de *WAE*.

El cumplimiento de estos objetivos asegura un desarrollo adecuado y ordenado de los conceptos, y provee de una buena estructura al trabajo, además de completitud. Como añadido, estos objetivos sirven de métrica para las conclusiones del trabajo, lo que permite comprobar si ha habido un desarrollo adecuado de estos, y en caso de ser necesario explicar las discrepancias respecto de ellos y las razones de alejamiento de los mismos.

Comentario: El código implementado se puede encontrar en el siguiente repositorio de *GitHub*: TFG-Autocodificadores-Wasserstein.

Capítulo 2

Conceptos técnicos previos

Aclarados los objetivos que se ha planteado conseguir, realicemos un repaso de los conceptos básicos para poder comprender bien el trabajo. Para ello, introduciremos las ideas de *Machine* y *Deep Learning*, redes neuronales, tipos de aprendizaje automático y las técnicas de regularización más utilizadas en el campo del *DL*, entre otras cosas.

Recordemos que se pueden consultar las siglas que no se reconozcan en el apartado de Notación, siglas y abreviaturas.

2.1. *Machine Learning*

El aprendizaje automático, *Machine Learning*, o simplemente *ML*, es una de las ramas de la IA, cuyo objetivo final es simular el funcionamiento del cerebro humano al aprender conceptos y reglas para poder reproducirlo en las máquinas.

Esta rama se centra en desarrollar algoritmos que resuelvan problemas y tareas basados en la experiencia que se puede obtener de los datos iniciales, es decir, algoritmos que obtienen conocimiento a partir de una serie de entradas. La forma de obtener la experiencia necesaria sobre los datos iniciales se basa en la mejora de su toma de decisiones de forma iterativa, extrayendo la información de las entradas.

La forma iterativa para la mejora en la toma de decisiones se llama entrenamiento del algoritmo y, una vez finalizado, permite extraer información generalizada de los datos iniciales. Matemáticamente esto se representa por el ajuste de pesos en matrices y funciones no lineales, que transforman un objeto (generalmente un vector) en otro distinto, aunque esto se detallará mejor en el apartado 2.7.

Una vez entrenado el algoritmo se obtienen salidas a partir de nuevas entradas. Para poder medir la bonanza del algoritmo entrenado, se mide la distancia entre lo obtenido por la ejecución del algoritmo y lo que esperábamos obtener. Para poder realizar este cálculo, se utiliza una función, llamada función de pérdida, que devuelve un valor que ha de interpretarse. Generalmente, el objetivo es la reducción del valor de esta función, y se procura que durante el entrenamiento se reduzca iterativamente el valor que generan las salidas con la función de pérdida. De esta manera se intenta asegurar una mejora en los resultados hasta conseguir los objetivos planteados para el algoritmo.

Cuando consideramos que el valor de la función de pérdida es adecuado, obtenemos un modelo que generaliza los datos y puede realizar tareas relacionadas con ellos. Estos modelos son muy útiles cuando no podemos codificar las reglas o existe una alta complejidad para la resolución del problema. Sin embargo, no siempre es óptimo utilizar un algoritmo y modelo de *ML* pues el entrenamiento requiere, en ocasiones, altos tiempos de computación. Hay ocasiones en las que es mejor implementar algoritmos tradicionales, basados en programación estructurada, que resuelvan un problema concreto más rápido que los algoritmos de *ML*.

2.2. Tipos de aprendizaje

Antes de introducir el pilar fundamental de este tipo de algoritmos (las redes neuronales), veamos que son los enfoques de aprendizaje, las funciones de pérdida y el riesgo empírico.

El aprendizaje de los algoritmos, tanto de *ML*, como *DL* puede seguir uno de los dos siguientes enfoques: si se conocen los posibles resultados y etiquetas para un conjunto de datos, se dice que el aprendizaje es supervisado. En el caso contrario, cuando no se trabaja con datos etiquetados hablamos de aprendizaje no supervisado.

En ambos aprendizajes se parte de un conjunto de datos (o muestra) que generalmente son representados por vectores p-dimensionales $\mathbf{x} = (x_1, ..., x_p) \in \mathbb{R}^p$. A este espacio lo denotaremos por $\mathcal{X}$. Es necesario también introducir las etiquetas, denotadas por $\mathcal{Y}$, un subconjunto de $\mathbb{R}$ que puede llegar a ser el total.

Cuando se conoce el conjunto $\mathcal{Y}$ de antemano entonces estaremos dentro del enfoque supervisado del aprendizaje. Esto es porque se pueden observar las salidas del algoritmo entrenado. Además, también se puede comprobar que las salidas producidas por el mismo son correctas o erróneas al clasificar o realizar regresión sobre los datos.

Estos dos problemas (clasificación y regresión) se diferencian principalmente en el espacio de etiquetas. En el caso de que $\mathcal{Y} = \mathbb{R}$, hablamos de regresión, y en el caso de un número finito de etiquetas, es decir, $\mathcal{Y} = \{1, 2, ..., K\}$ con $K \in \mathbb{N}$, hablamos de clasificación.

Entre algunos ejemplos de aplicación de este tipo de aprendizaje tenemos la regresión lineal, las redes neuronales o las *SVM*.

El enfoque no supervisado no maneja datos etiquetados, de forma que no se conocen a priori los resultados para los datos de entrenamiento, por eso es utilizado para la búsqueda de patrones, asociaciones entre los datos y la generación de elementos mediante algoritmos de *Deep Learning*.

Para este enfoque también se utiliza la letra $\mathcal{Y}$ como espacio de llegada, pues es común utilizar técnicas como la reducción de dimensión para poder abarcar la mayoría de los problemas, dado que no todos los datos serán significativos para el resultado.

Ejemplos de este tipo de aprendizaje son las técnicas de clustering y reducción de dimensiones. En lo que refiere a este trabajo, dentro de este marco de aprendizaje están los *VAE*, precursoras de los algoritmos generativos punteros actuales.

2.3. Tipos de modelo

Antes de centrarnos en los conceptos principales de la inferencia variacional, hay que conocer los distintos tipos de modelos de pueden aparecer: los paramétricos, los no paramétricos y los de variable latente.

Un modelo paramétrico, $\mathcal{P}$, es una familia de distribuciones de probabilidad que se pueden indexar mediante un número finito de parámetros. Denotaremos por θ al vector que contiene a los parámetros y por Θ al espacio donde se encuentran. Generalmente, un modelo paramétrico se describe de la siguiente manera:

$$\mathcal{P}_\Theta = \{F(\mathbf{x};\theta), \theta \in \Theta\}.$$

Un ejemplo sencillo sería una familia de distribuciones uniformes sobre intervalos $[a, b]$:

$$\mathcal{P}_{a,b} = \{F(x;a,b) = \frac{x-a}{b-a} : x \in [a,b]; a, b \in \mathbb{R}\},$$

donde a y b son los parámetros que definen la distribución y donde se asigna el valor 0 para elementos $x < a$ y el valor 1 para elementos $x \geq b$.

Los modelos no paramétricos son similares a los anteriores, con la salvedad de la indización en infinitos parámetros.

En este caso, a los parámetros del modelo se los considera como la realización de un proceso estocástico, lo que define una distribución de probabilidad sobre Θ, el espacio de parámetros, y permite entender a θ como una función aleatoria.

No es adecuado asumir inicialmente que los datos de los que se parte son independientes e igualmente distribuidos, después de todo, no pueden ser completamente independientes, pues deben estar relacionados aunque no sea de forma visible. A la relación entre los datos se la conoce como razón latente. Estas relaciones se suelen representar mediante variables aleatorias, generalmente mediante la letra Z. De esta forma, obtenemos una distribución conjunta $p(x, z)$, siendo $p(x)$ la distribución de los datos y z una observación de Z.

Tomando de forma adecuada la distribución marginal, podemos obtener la distribución de los datos a partir de la conjunta:

$$p(x) = \int p(x,z)dz = \int p(x|z)p(z)dz.$$

2.4. Funciones de pérdida

Como conocemos ya los tipos de aprendizaje, es el momento de buscar una manera de cuantificar la fiabilidad de los modelos entrenados, de modo que se observen los resultados y salidas esperados.

Para ello se utilizan funciones $f : \mathcal{X} \to \mathcal{Y}$, que actuarán sobre los datos con el fin de predecir salidas dentro del espacio de llegada, sea cual sea el enfoque del aprendizaje. Para definir f es necesario tomarla dentro de una familia de funciones, lo que no restringe que f sea resultado de la composición de funciones. Sin embargo, la familia en la que se encuentre la función de predicción ha de cumplir dos requisitos contrapuestos:

- Ha de ser suficientemente grande para que no exista un sobreajuste, es decir, el modelo se adapte bien a nuevos datos o muestras que se puedan presentar y que sean similares a los iniciales.

- No puede ser excesivamente grande, pues entonces la computación se vuelve extremadamente costosa y es imposible entrenar un modelo en un tiempo adecuado.

Además, se ha de buscar un equilibrio entre ambos objetivos.

Junto a la función de predicción, se utiliza otra función mediante la cual se mide su calidad. Esta función es la función de pérdida y se denota, generalmente, por $\ell(y, \mathbf{x}, f)$.

Las funciones de pérdida miden la discrepancia entre lo predicho y lo real, es decir, entre $f(\mathbf{x})$ e y. Si denotamos $f(\mathbf{x}) = \hat{y}$ a la predicción, podemos reescribir la notación de la función de pérdida:

$$\ell(y, \mathbf{x}, f) = \ell(y, \hat{y}),$$

que será, generalmente, la forma en que se mencionará a la función de pérdida en adelante.

Un ejemplo clásico de una función de pérdida es la llamada pérdida cuadrática:

$$\ell(y, \hat{y}) = (y - \hat{y})^2; \;\; y, \hat{y} \in \mathbb{R},$$

ampliamente utilizada en muchos problemas de clasificación y regresión.

A continuación, detallaremos brevemente lo que es la información y entropía, para poder introducir una de las funciones de pérdida más utilizadas: la entropía cruzada. Primero, conviene conocer el objetivo de la inferencia para entender mejor los objetivos del trabajo.

La inferencia estadística se refiere al proceso general mediante el cual se deducen distribuciones de probabilidad que queremos conocer (posiblemente condicionadas o marginales). Estas distribuciones modelaran completa o parcialmente los datos de los que se parte.

Comencemos tratando la idea de medidas de información. La teoría de la información se centra en responder preguntas como: ¿Cuánta información nos proveen los datos sobre los parámetros del modelo? ¿Cuánta información tiene una variable aleatoria? ¿Cómo medimos al información ganada mediante la observación de una variable aleatoria? ¿Qué significa la información?

Estas preguntas también son de interés para el campo del *ML* y los sistemas de computación.

Una de las principales formas de medir la información en el caso de sistemas inteligentes es la entropía. Para entender mejor estos conceptos se puede revisar alguna de estas referencias [13, 40]. También es importante tener en cuenta que los logaritmos que vamos a utilizar en esta sección son en base 2.

La definición estándar de la entropía se toma como la medida no negativa de la información esperada contenida en una variable aleatoria. Esta información cuantifica cómo de sorprendentes son los resultados producidos: cuanto menos probable es un evento aleatorio, más información aporta sobre los datos iniciales. Por otra parte, la información está

relacionada con la probabilidad de eventos específicos, de forma que se puede concluir que la entropía es dependiente de una variable aleatoria y su distribución de probabilidad.

La información se mide de la siguiente manera, considerando una función de masa de probabilidad $p(x)$ (x es una observación de los datos):

$$h(x) = -\log(p(x)).$$

Matemáticamente, y utilizando la misma función de masa de probabilidad, la entropía se calcula en base a la anterior definición de información:

$$H(x) = -\sum_{i\in\mathcal{I}} p(x_i)\log(p(x_i)) = \mathbb{E}_X(-\log(p(x))), \tag{2.1}$$

donde se ha denotado por X a la variable aleatoria discreta con masa p, e $\mathcal{I}$ denota el conjunto de índices de elementos que definen dicha masa.

Sin embargo, no siempre la dependencia es de una única variable, o la variable de la que depende la entropía está condicionada por otra variable. Esto modifica la definición anterior, aunque no de manera drástica:

$$H(X|Y) = -\sum_{i,j\in\mathcal{I},\mathcal{J}} p(x_i,y_j)\log(\frac{p(x_i,y_j)}{p(y_j)}) = \mathbb{E}_{X|Y}(-\log(p(x|y))).$$

A este tipo de entropía se la conoce como entropía condicionada, y parte de la información esperada de $X|Y$ y la distribución conjunta $p(x,y)$.

Finalmente, y con el objetivo de abandonar las variables discretas, observemos que:

$$\lim_{\varepsilon_i\to 0} H(X) = -\sum_{i\in\mathcal{I}} f(x_i)\varepsilon_i\log(f(x_i)\varepsilon_i) = -\sum_{i\in\mathcal{I}} f(x_i)\varepsilon_i\log(f(x_i)) - \sum_{i\in\mathcal{I}} f(x_i)\varepsilon_i\log(\varepsilon_i) =$$
$$= -\int f(x)\log(f(x))dx - \log(\varepsilon_i)\int f(x)dx = -\int f(x)\log(f(x))dx - \lim_{\varepsilon_i\to 0}\log(\varepsilon_i) =$$
$$-\int f(x)\log(f(x))dx + \infty.$$

Sin embargo llegamos al problema de que un componente del anterior razonamiento es no finito, y la última igualdad es simbólica para mostrar este problema de forma más fehaciente.

La entropía, definida de la manera anterior, no es equivalente a la entropía para las distribuciones con densidad, además de tener un problema de divergencia para el caso continuo.

Debido a estos inconvenientes, C.Shannon [35], que intenta acercar la entropía a las variables con densidad ($f(x)$), define la entropía diferencial de una variable continua de la siguiente manera:

$$H(X) = -\int f(x)\log(f(x))dx = \mathbb{E}_X(-\log(f(x))).$$

El objetivo final de este capítulo es introducir el siguiente concepto: la entropía cruzada, una de las funciones de pérdida más comunes. Consideremos dos distribuciones de probabilidad, p y q, sobre la misma variable aleatoria. Se llama y denota a la entropía cruzada por:

$$H_q(p) = -\int p(x)\log(q(x))dx.$$

Esta entropía representa la longitud esperada de un mensaje transmitido que sigue la variable X cuando asumimos la distribución incorrecta $q(x)$, y será necesaria a la hora de definir la cota inferior para la divergencia de *Kullback-Leibler*.

Finalmente, mencionemos que el uso de distancias y normas entre probabilidades será de utilidad para medir la pérdida en los resultados, como veremos cuando tratemos los *autoencoders*.

2.5. Riesgo

Asociado a las funciones de pérdida está el riesgo. Este es un concepto cuyo objetivo es modelar la variabilidad de los datos, lo que transforma a las funciones de pérdida en variables aleatorias. Veamos como se puede modelar.

Se considera el vector aleatorio $(p+1)$-dimensional $(y, \mathbf{x})$. Con este modelado se predice la pérdida esperada respecto de la distribución de $(y, \mathbf{x})$, esto es, $E_{(y,\mathbf{x})\sim\mathcal{D}}[\ell(y, \mathbf{x}, f)]$. A la pérdida esperada se la conoce como riesgo asociado a f y se denota por $R_{\mathcal{D}}(f)$.

Para tener un modelo fiable se intenta encontrar una variabilidad baja en la distribución de los datos y predicciones. Esto se realiza, inicialmente, con una búsqueda de la resolución teórica del problema. De esta forma se puede encontrar una regla que minimiza el riesgo para cada función de pérdida, la cual es llamada regla de Bayes. Sin embargo, esta regla presenta un problema principal y es la dependencia de la distribución, a priori desconocida, del vector aleatorio $(y, \mathbf{x})$.

En general no se dispone de todo el vector, si no de un conjunto $\mathcal{T}$, que llamaremos de entrenamiento o *train*, el cual es un muestreo $(y_1, \mathbf{x}_1)$, ..., $(y_n, \mathbf{x}_n)$ del vector original.

Entonces, la solución a nuestro problema pasa por el cálculo del riesgo empírico, que por la Ley de los Grandes Números convergerá al riesgo teórico.

Para calcular el riesgo empírico, este se define de la siguiente manera:

$$R_{\mathcal{S}}(f) = \frac{1}{n}\sum_{i=1}^{n}\ell(y_i, \hat{y}_i),$$

donde $\hat{y}_i := f(\mathbf{x}_i)$.

Ahora, considerando $\mathcal{F}$, una clase de reglas, y que f, la función de predicción, varía dentro de esta clase, se realiza la búsqueda siguiente:

$$\arg\min_{f\in\mathcal{F}}(R_{\mathcal{S}}(f)).$$

Resolviendo este problema, si el conjunto $\mathcal{T}$ es suficientemente grande, se minimizará el riesgo $R_{\mathcal{D}}(f)$.

A la solución de este problema se le llama regla *ERM*.

Tras este proceso, se traslada el problema a la elección de una clase $\mathcal{F}$ suficientemente grande de funciones, para obtener una buena aproximación del riesgo real, y suficientemente flexible, para evitar el sobreajuste y ganar adaptabilidad a la hora de predecir resultados.

2.6. Divergencia de *Kullback-Leibler*

Una divergencia entre probabilidades, en términos coloquiales, es una función de pares de probabilidades que mide lo diferentes que son estas. Generalmente, esto se consigue asignando valores pequeños donde las probabilidades similares y valores grandes donde no lo son. Además, frecuentemente, se procura que estas tomen valores positivos únicamente.

Es por eso por lo que se suelen utilizar como herramienta teórica a la hora de desarrollar algoritmos que necesitan comparar distintas probabilidades, como es el caso de las redes generativas. Más específicamente, en este último caso, es necesario para comparar el modelo con la distribución de los datos, lo que nos dirá como de bueno es el modelo.

Una de las divergencias más importantes en el campo de la Estadística es la divergencia de *Kullback-Leibler*. Para definir esta medida de forma correcta, es conveniente introducir primero el concepto de continuidad absoluta de una medida respecto de otra.

Sean μ, ν dos medidas en un espacio medible (Ω, σ). Diremos que μ es absolutamente continua respecto de ν, o que ν domina a μ, si para cada conjunto medible A, se tiene que:

$$\nu(A) = 0 \Longrightarrow \mu(A) = 0.$$

Lo denotaremos por $\mu \ll \nu$.

Para caracterizar la forma de las medidas absolutamente continuas respecto de una medida σ-finita dada en casos generales se suele recurrir a la derivada de *Radon-Nikodym*. Aunque la demostración del teorema de *Radon-Nikodym* escapa del alcance del trabajo, es este el que define la derivada, por lo que se enunciara ahora. Si se quiere entender con mayor detalle, se puede ver en [8]. El enunciado del teorema es el siguiente: consideremos μ y ν, dos medidas en el mismo espacio medible, con μ finita y ν positiva y σ-finita, tales que $\mu \ll \nu$. Entonces, existe una función medible f con valores reales tal que:

$$\mu(A) = \int_A f d\nu. \tag{2.2}$$

Además, la función f es única ν-c.s, es decir, si $\tilde{f}$ es otra función que cumple lo anterior, entonces $f = \tilde{f}$ salvo en un conjunto de ν-medida nula a lo sumo. A mayores, si μ es positiva, entonces $f \geq 0$ en ν-casi todo punto.

A la función f se la conoce como derivada de *Radon-Nikodym* de μ respecto de ν, y se denota por $\frac{d\mu}{d\nu}$. Por lo anterior, sabemos que la derivada está definida ν-casi siempre.

Observamos que en las condiciones de (2.2), se tiene que para cada función g medible:

$$\int g d\mu = \int g \frac{d\mu}{d\nu} d\nu.$$

Esto a de interpretarse en el sentido siguiente: el lado izquierdo es integrable si, y sólo si, lo es el lado derecho, en cuyo caso ambas integrales coinciden.

Esto es sencillo de deducir de (2.2), que se cumple para las funciones indicadoras, por linealidad para las funciones simples, por límites monótonos para funciones positivas y por sumas de partes positivas y negativas para funciones de carácter general. También nos permite deducir otras propiedades de interés: si $\mu \ll \nu$ y $\nu \ll \rho$, entonces $\mu \ll \rho$ y:

$$\frac{d\mu}{d\rho} = \frac{d\mu}{d\nu}\frac{d\nu}{d\rho}.$$

Lo cual se traduce, para un conjunto A medible:

$$\mu(A) = \int_A \frac{d\mu}{d\nu} d\nu = \int_A \frac{d\mu}{d\nu}\frac{d\nu}{d\rho} d\rho$$

Tras la estos conceptos preliminares, podemos introducir la divergencia de *Kullback-Leibler*. Si P y Q son dos probabilidades definidas sobre el mismo espacio, con $P \ll Q$:

$$D_{KL}(P, Q) = \int \log \frac{dP}{dQ} dP.$$

Si P no es absolutamente continua respecto de Q, entonces se asigna el valor $+\infty$ a la divergencia. A esta divergencia se la conoce como entropía relativa de P respecto de Q.

Veamos alguna de sus propiedades. Si P y Q son dos probabilidades en el mismo espacio, entonces $D_{KL}(P, Q) \geq 0$, dándose la igualdad si, y sólo si, $P = Q$. Si $P \ll \mu$ y $Q \ll \mu$, siendo μ σ-finita y $P \ll Q$, se tiene que:

$$D_{KL}(P, Q) = \int \log \frac{\frac{dP}{d\mu}}{\frac{dQ}{d\mu}} \frac{dP}{d\mu} d\mu.$$

Y en el caso más importante, si tanto P como Q tienen densidades (p y q respectivamente), es decir, son absolutamente continuas de la medida de *Lebesgue*, entonces:

$$D_{KL}(P, Q) = \int \log \frac{p(\mathbf{x})}{q(\mathbf{x})} p(\mathbf{x}) d\mathbf{x}.$$

En el caso de probabilidades discretas el cálculo es similar:

$$D_{KL}(P, Q) = \sum_{i \in \mathcal{I}} p_i \log \frac{p_i}{q_i},$$

asumiendo que P y Q tienen soporte en una colección numerable de puntos $\{a_i\}_{i \in \mathcal{I}}$, con probabilidades p_i y q_i respectivamente. Aparte, en el caso discreto podemos ver la relación con la entropía de las distribuciones (2.1), pues tenemos:

- La entropía: $H(P) = -\sum_{i\in\mathcal{I}} p_i \log p_i$.
- La entropía cruzada: $H(P,Q) = -\sum_{i\in\mathcal{I}} p_i \log q_i$.

Y la relación entre la entropía cruzada, utilizada comúnmente y la divergencia:

$$H(P,Q) = H(P) + D_{KL}(P,Q)$$

Además, la divergencia guarda relación con la estimación de máxima verosimilitud, y como veremos en la segunda parte del trabajo, la verosimilitud es una parte importante de la inferencia variacional.

Centrémonos en esta relación. Sea $X_1, ..., X_m$ una muestra aleatoria simple de P, con densidad f. Nos planteamos estimar f con el modelo $\{f(x|\theta) : \theta \in \Theta\}$. Sea P_θ la probabilidad, fijado θ, asociada a $f(\cdot|\theta)$. Asumimos además que distintos θ dan lugar a distintas P_θ.

Por otra parte, el estimador máximo verosímil (EMV), $\hat{\theta}_n$, es el argumento que maximiza:

$$l_n(\theta) = \frac{1}{n}\sum_{i=1}^{n} \log(X_i|\theta).$$

También maximiza:

$$K_n(\theta) = \frac{1}{n}\sum_{i=1}^{n} \log(X_i|\theta) - \frac{1}{n}\sum_{i=1}^{n} \log(X_i) = -\frac{1}{n}\sum_{i=1}^{n} \frac{\log(X_i)}{\log(X_i|\theta)}.$$

A partir de esto, utilizando la Ley de los Grandes Números sabemos que:

$$K_n(\theta) \to^{c.s} K(\theta) = D_{KL}(P, P_\theta),$$

para cada $\theta \in \Theta$. Intuitivamente, como $K_n \to K$, los maximizadores también convergerán ($\theta_n \to \theta$), es decir, los maximizadores hacen converger las probabilidades P_{θ_n} hacia la probabilidad P_θ que muestre la menor divergencia respecto de P.

Lo anterior nos permite ver que calcular el EMV es equivalente a minimizar la D_{KL}, aunque esta forma de proceder presenta ciertas limitaciones. Por ejemplo, la interpretación se ve dificultada por el hacho de que la divergencia no es en realidad una métrica, pues no es simétrica.

La simetrización de esta divergencia se conoce como divergencia de *Jensen-Shannon*, aunque su definición se dará más adelante.

2.7. Redes neuronales

Previo al concepto de aprendizaje profundo o *Deep Learning* hemos de introducir las redes neuronales, pues son la premisa de las redes de aprendizaje profundo.

El objetivo de toda red neuronal es emular la sinapsis que ocurre en el cerebro humano cuando aprende o recuerda conceptos. Traducido al lenguaje utilizado sobre los datos, esto

significa que las redes neuronales se centran en extraer y reconocer patrones de los datos de $\mathcal{X}$ para poder generalizar información y predecir resultados en $\mathcal{Y}$. Esta predicción se suele obtener en forma de probabilidad de pertenencia a una clase o tipo de datos en el caso del aprendizaje supervisado, o en ordenación de datos por similitudes entre los mismos en el caso del aprendizaje no supervisado.

Si se tuviera que hacer una lista de los objetivos de la computación neuronal, un buen ejemplo sería el de las lecciones de [16]. Estos objetivos serían los siguientes:

- Entender el funcionamiento del cerebro, pues es muy frágil para un estudio en directo, mediante simulaciones por ordenador.
- Entender como las neuronas trabajan con la información, su forma de procesarla en paralelo y la adaptabilidad de sus conexiones.
- Resolver problemas prácticos utilizando algoritmos inspirados en el cerebro y la sinapsis.

También conviene notar que no todos los algoritmos se inspiran en el cerebro, y algunos de estos, por muy diferentes que sean, son muy útiles a la hora de trabajar. Ejemplos de algoritmo cuya forma de proceder no se basa en el funcionamiento biológico de las neuronas son las *GAN* o los *AE*.

Estudiemos el proceso neuronal artificial. Una red neuronal es una composición de transformaciones lineales y no lineales que actúan sobre un vector. Este vector de entrada provendrá de los datos observados. A su vez, hay que determinar cuantas transformaciones de ambos tipos son necesarias. Al número de transformaciones lo llamaremos el número de capas de la red.

Si $L \in \mathbb{N}$ es el número de capas de la red, se denota por f_j, $j \in \{1, ..., L\}$, a las transformaciones no lineales y W_j, $j \in \{1, ..., L\}$, a las transformaciones lineales. Aparte, $\mathbf{x} \in \mathcal{X}$ será nuestro vector de datos, como ya lo habíamos denotado en la primera subsección de este capítulo.

Veamos con más detalle el comportamiento de las transformaciones:

1. $f_j : \mathbb{R}^{d_j} \to \mathbb{R}^{d_{j+1}}$ serán funciones no lineales, también llamadas funciones de activación de las neuronas. Las funciones de activación más utilizadas son la sigmoide (o logística) y la $ReLU$.
2. $W_j = (w_{kl}^{(j)})_{1:d_j, 1:d_{j-1}}$ serán transformaciones lineales, representadas por matrices.

Lo anterior permite escribir la red neuronal mediante la composición de funciones:

$$g_\theta(\mathbf{x}) = f_L(W_L f_{L-1}(W_{L-1} \cdots f_1(W_1(\mathbf{x})) \cdots)), \tag{2.3}$$

donde $\theta = (W_1, ..., W_L)$ es un parámetro que nos permite identificar cada función de la clase.

La elección de las transformaciones lineales ayudará a minimizar la función de pérdida elegida dado que sus elementos, $w_{kl}^{(j)}$, se ajustan durante el entrenamiento de la red, razón

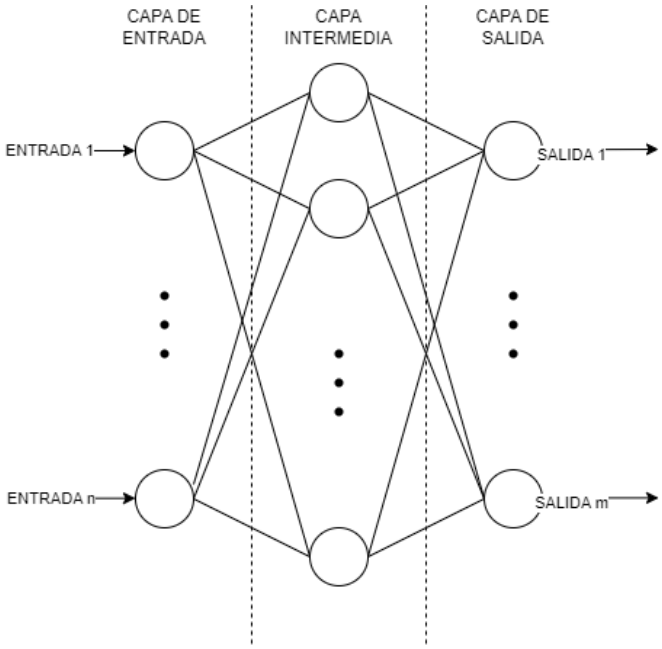

Figura 2.1: Diagrama de arquitectura de una RN

por la cual las matrices son llamadas matrices de pesos. El objetivo es refinar así los resultados obtenidos, buscando que se asemejen a las salidas esperadas de la red. Veamos ahora una formulación equivalente para la red neuronal. Para ello, observamos la salida que genera cada composición $f_j \circ W_j$ al aplicarla sobre un vector.

Dado que partimos de un vector, también se puede escribir la red como una secuencia de resultados:

$$\mathbf{a}_0 = \mathbf{x}.$$

$$\mathbf{a}_j = f_j(W_j \mathbf{a}_{j-1}), \quad j \in \{1, ..., L\}.$$

Y de este modo, se tiene que: $\mathbf{a}_L = g_\theta(\mathbf{x})$.

Esta conexión de neuronas, una tras otra, no es más que una función que transforma el vector de entrada $\mathbf{x} \in \mathcal{X}$ en un vector de salida $\mathbf{y} \in \mathcal{Y}$. Para poder definir la red de manera correcta, la distribución de neuronas en capas está conectada; la primera capa es por la que entrarán los datos, luego se pasa por una serie de capas intermedias hasta llegar a la última, que genera la salida.

Se puede observar este proceso en la figura 2.1, con las relaciones entre neuronas (círculos) cuando se conectan todas entre sí (líneas). Las líneas a puntos representan la posibilidad de más capas intermedias. También se puede observar la agrupación por capas de una RN, donde se ha separado cada capa por líneas discontinuas.

Hablemos sobre lo que significa entrenar una red neuronal. El proceso de entrenamiento es la serie de pasos que permite obtener los mejores pesos de las matrices de pesos para que cumplan una tarea específica. El objetivo final de la red neuronal será minimizar el riesgo o la pérdida para que los resultados se ajusten a los objetivos que definamos. A cada iteración del entrenamiento de una red se la llama época.

Veamos de que modo afecta el entrenamiento a la red neuronal. Durante el proceso de entrenamiento, vamos modificando los pesos de las neuronas para obtener salidas cada vez más cercanas a los objetivos que deseemos en base a la función de perdida que se aplica sobre el resultado de toda la red. Sigamos los pasos del entrenamiento y aprendizaje de las redes neuronales, ya que sabemos como afectan estos a la red una vez construida:

1. Inicializamos los pesos y el sesgo de las neuronas de la red, y preparamos los datos para el entrenamiento.

2. Pasamos los datos por la red y calculamos el error con la función de pérdida.

3. Utilizamos un optimizador para actualizar los parámetros de la red, de modo que se puedan obtener mejores resultados, es decir, optimizamos el valor de la función de pérdida.

4. Repetimos los pasos 2 y 3 hasta alcanzar una condición de parada que hayamos preestablecido.

Uno de los peligros del anterior entrenamiento es pensar que la condición de parada ha de ser el momento en que todos los ejemplos de los datos iniciales se clasifiquen correctamente. Esta clasificación correcta permite aprender sobre el conjunto de datos pero restringe la generalización de los datos, lo que se conoce como sobreajuste.

Los algoritmos de aprendizaje automático más sencillos tienen RRNN de una única capa intermedia. Cuando se mencionan las capas ocultas de una red, nos referimos a las capas intermedias que posee, pues son las que no producen resultados "visibles".

2.8. Descenso de gradiente

Ahora introducimos la técnica de descenso de gradiente, la más utilizadas para la optimización de funciones de pérdida dentro del marco del *Machine learning*.

El objetivo de esta técnica es buscar una solución mínima para la siguiente función objetivo:

$$G(\theta) = \frac{1}{n}\sum_{i=1}^{n} \ell(y_i, g_\theta(\mathbf{x}_i)),\ \theta \in \Theta. \tag{2.4}$$

En esta expresión se puede identificar lo introducido hasta ahora, es decir, la función objetivo es el promedio empírico de las pérdidas entre los resultados reales y las salidas obtenidas por la red con los datos de entrenamiento.

En este problema hay que tener en cuenta la función de pérdida definida, ℓ, el conjunto de entrenamiento con el que se trabaja, $\mathcal{T} = \{(y_1, \mathbf{x}_1), ..., (y_n, \mathbf{x}_n)\}$, y el conjunto de reglas donde se busca la mejor aproximación, $\mathcal{R} = \{g_\theta : \theta \in \Theta\}$, para el cual θ será un parámetro. Si se quiere simplificar el problema, se asumirá que $\Theta \subset \mathbb{R}^d$, que es lo que haremos.

Ahora, en el caso en que G sea suficientemente regular, esta se podrá aproximar utilizando su gradiente:

$$G(y) \simeq G(x) + \nabla G(x)(y - x),$$

donde si $y = x + \gamma u$ con $\|u\| = 1$ entonces:

$$G(y) \simeq G(x) + \gamma \nabla G(x) u.$$

A partir de lo anterior se extraen relaciones muy útiles para casos favorables de G. La elección de γ, llamado tasa de aprendizaje (*learning rate*), influirá en la velocidad de convergencia al igual que el gradiente de G.

Buscando la dirección de mayor descenso del gradiente, que será para la que $u = -\frac{\nabla f(x)}{\|\nabla f(x)\|}$, se obtiene que, considerando la siguiente sucesión (siendo θ_0 arbitrario):

$$\theta_{n+1} = \theta_n - \gamma \nabla G(\theta_n),$$

esto asegura una convergencia veloz siempre que G cumpla una serie de propiedades.

Las propiedades que ha de tener para G para dicha convergencia son las siguientes:

- Convexidad: pues si G es una función convexa con un mínimo local, ese mínimo será, en realidad, un mínimo global. Si G no fuera una función convexa, entonces la convergencia hacia un mínimo local no nos asegura la convergencia hacia un mínimo global.
- α-convexidad: G es una función α-convexa si $G(x) = \frac{\alpha}{2}\|x\|^2$ es una función convexa. En el particular, si $\alpha > 0$ la α-convexidad implica la convexidad.
- β-suavidad: se dice que una función es β-suave si es diferenciable y su gradiente es lipschitziano de constante β: $\|\nabla G(x) - \nabla G(y)\| \leq \beta \|x - y\|, x, y \in \Theta$.

Si G es α-convexa y β-suave, entonces el descenso de gradiente es eficiente, lo que aporta muchas ventajas a la hora de resolver el problema de optimización. Decimos que es eficiente porque la convergencia es exponencial, lo cual es, computacionalmente hablando, la causa por la que el algoritmo es veloz al ejecutarse. Sin embargo, este caso es ideal y no es lo que ocurrirá generalmente. Si se quiere ver una demostración de las razones que lo hacen eficiente, estas se pueden encontrar en la sección 3.4.2 de [11]. Este algoritmo es la base de los algoritmos que se utilizan actualmente para actualizar los parámetros de las redes neuronales.

El descenso de gradiente estocástico es una variación de este algoritmo que en vez de cambiar los parámetros cuando termina una etapa del entrenamiento, se realiza la actualización de los parámetros de la red con cada minilote de entrenamiento. Un minilote de entrenamiento es un subconjunto de los ejemplos de entrenamiento. Sigue la siguiente forma:

$$\theta_{n+1} = \theta_n - \gamma \frac{1}{m} \sum_{i=1}^{m} G(\theta_n; x_i),$$

donde se toman los índices de los ejemplos x_i de un conjunto $\mathcal{I}$ con tamaño $m \leq n$, siendo n el tamaño del conjunto de entrenamiento. En el caso de que $m = n$ tenemos el descenso de gradiente tradicional. Además, los índices que pertenecen a $\mathcal{I}$ se escogen de manera aleatoria, lo cual da el apellido estocástico al algoritmo.

La idea de actualizar en cada lote surge para evitar cálculos redundantes que realiza el descenso de gradiente tradicional, más frecuente en los casos con grandes cantidades de datos. Esta repetición la causan ejemplos similares.

La manera de evitar repetir cálculos es utilizar el descenso de gradiente estocástico (*SGD*), pues la actualización del gradiente depende de cada lote y las actualizaciones de parámetros se centran en los cambios principales entre ejemplos similares. Además, este cambio respecto al descenso de gradiente estándar aumenta la velocidad de entrenamiento, pues no se utilizan todos los ejemplos en cada época de entrenamiento. Sin embargo, aunque la velocidad de entrenamiento aumenta, la velocidad de convergencia hacia el mínimo que buscamos decrece, pues el gradiente de un subconjunto de entrenamiento no tiene porqué ser la dirección optima de descenso hacia el mínimo.

Muchas de las redes profundas actuales también utilizan optimizadores basados en el algoritmo de descenso de gradiente; uno de los más utilizados, y el que utilizaremos en la parte práctica, es el algoritmo *ADAM* (*Adaptive Moment Estimation*). Este algoritmo se basa en el uso del descenso de gradiente estocástico o *SGD*.

2.9. Algoritmo de retropropagación

En esta sección se introduce un algoritmo para calcular el descenso de gradiente de forma eficiente en redes neuronales, llamado algoritmo de retropropagación.

El objetivo del entrenamiento por retropropagación es el cálculo del gradiente de la función de pérdida, de esta forma, se puede aplicar la técnica de descenso de gradiente. Si se conoce $y \in \mathcal{Y}$, la etiqueta correcta para un dato, y se tiene $\hat{y}$, la etiqueta producida por la red neuronal, entonces se buscará el gradiente de la función $\ell(y, \hat{y})$.

Manteniendo la notación utilizada anteriormente y siendo $\mathbf{z}_j = W_j \mathbf{a}_{j-1}$, $\mathbf{a}_j = f(\mathbf{z}_j)$, se tiene que $\hat{y} = g_\theta(\mathbf{x}) = \mathbf{a}_L$. Ahora, siendo $\theta = (W_1, ..., W_L)$,y aplicando una vez la regla de la cadena se obtiene:

$$\nabla_{W_L} \ell(y, \hat{y}) = (\nabla_{\hat{y}} \ell \odot \mathbf{f}'_L)\mathbf{a}^t_{L-1}.$$

Recordemos que se ha denotado por $\odot$ el producto de *Hadamard* de matrices.

Ahora, utilizando una segunda vez la regla de la cadena se tiene lo siguiente:

$$\nabla_{W_{L-1}} \ell(y, \hat{y}) = ((W^t_L(\nabla_{\hat{y}} \ell \odot \mathbf{f}'_L)) \odot \mathbf{f}'_{L-1})\mathbf{a}^t_{L-2}.$$

Entonces, por iteración del proceso, para cada una de las L funciones lineales y no lineales de la red, se observa que sólo es necesario conocer los valores de $\mathbf{a}_j = f(\mathbf{z}_j)$, $\mathbf{f}'_j = f'_j(\mathbf{z}_j), j \in \{1, ..., L\}$, y $\nabla_{\hat{y}} \ell$.

Para obtener estos valores se realiza un cálculo progresivo.

Dados y, $\mathbf{x}$, $\theta = (W_1, ..., W_L)$:

1. $\mathbf{a}_0 = \mathbf{x}$
2. Si j recorre el conjunto $\{1, 2, ..., L\}$ en ese orden:

$$\mathbf{z}_j := W_j \mathbf{a}_{j-1},$$

$$\mathbf{a}_j := f(\mathbf{z}_j),$$
$$\mathbf{f}'_j := f'_j(\mathbf{z}_j).$$

3. $\nabla_{\hat{y}}\ell = \nabla_{\hat{y}}\ell(y, \mathbf{a}_L)$

Y ya calculados estos valores se procede con el algoritmo de retropropagación:

1. $\mathbf{g} = \nabla_{\hat{y}}\ell$
2. Si j recorre el conjunto $\{L, L-1, ..., 1\}$ en ese orden:

$$\mathbf{g} := \mathbf{g} \odot \mathbf{f}'_j,$$
$$\nabla_{W_j}\ell(y, \hat{y}) := \mathbf{g} \odot \mathbf{a}^t_{j-1},$$
$$\mathbf{g} := W^t_j \mathbf{g}.$$

La retropropagación es un algoritmo importante en el entrenamiento de las redes neuronales porque permite utilizar el descenso de gradiente como optimizador de la función de pérdida. Aún así, el algoritmo es costoso computacionalmente hablando, pues requiere de realizar muchos productos matriciales (los cuales se pueden paralelizar utilizando *GPU*s).

2.10. *Deep Learning*

Llegados a este momento, surge la pregunta: ¿qué ocurre con las RRNN con más de una capa intermedia? Son estas redes las que conforman el alcance del aprendizaje profundo o *Deep Learning* (*DL*).

El *DL* es un campo del *ML* centrado en los patrones complejos de datos mediante el uso de redes neuronales profundas. Su principal objetivo es extraer información de grandes volúmenes de datos, lo cual requiere generalmente altos tiempos de ejecución, donde la red aprende en cada capa. El diagrama 2.2 ilustra la arquitectura de una red profunda, pues tenemos más de una capa oculta.

Actualmente, estas redes tienen un gran protagonismo, pues las redes generativas, el reconocimiento del habla, o los grandes modelos del lenguaje son ejemplos de ellas, muy utilizados hoy en día.

Dado que las redes neuronales profundas tienen muchas capas y neuronas, se genera el siguiente problema: la red se adapta mejor al conjunto de entrenamiento. Aunque a priori pueda parecer una buena cualidad para las redes que se quiere construir, a posteriori se genera sobreajuste sobre el conjunto de entrenamiento. Para evitar este potencial sobreajuste, seguiremos algunas técnicas de regularización del entrenamiento de redes profundas:

- Parada temprana: esta técnica sigue una serie de pasos para evitar que la red entrene más de lo necesario. Partiendo del conjunto de entrenamiento, se divide en dos, un conjunto de entrenamiento más pequeño y un conjunto de validación. Con este paso

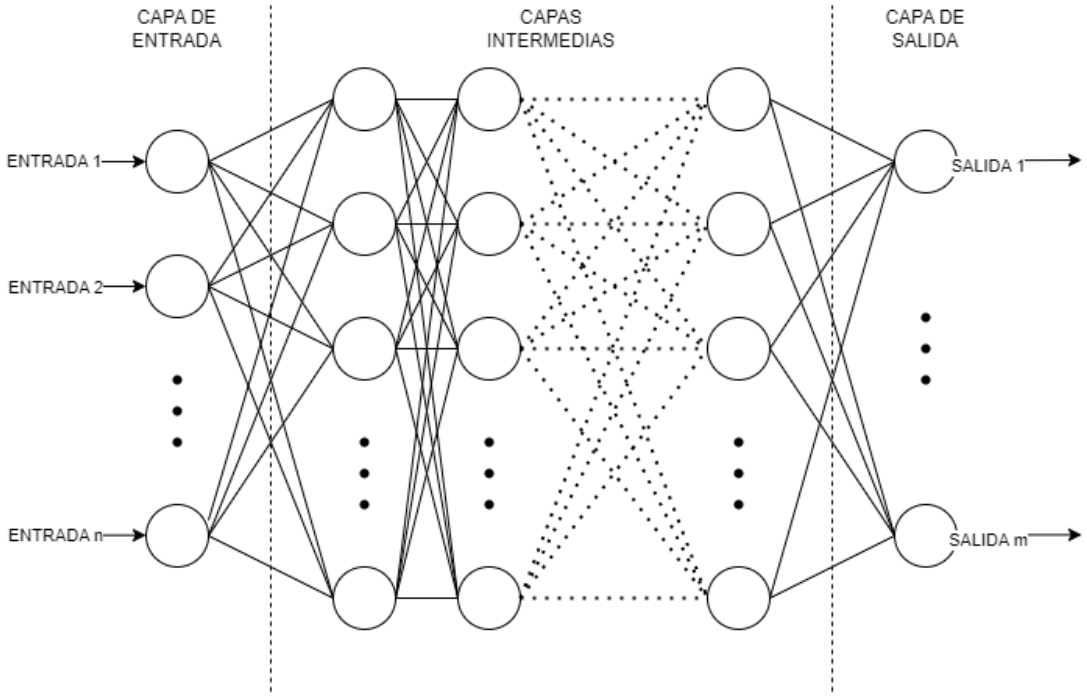

Figura 2.2: Diagrama de arquitectura de una RN profunda

realizado, se monitorizará el comportamiento de la red con el conjunto de validación, de modo que, cuando se aplique el optimizador de la red, se detenga el entrenamiento en el momento que haya un aumento en el error para la validación.

- Normalización de lotes: esta estrategia tiene como objetivo normalizar la entrada de las funciones de activación, lo cual no es más que asegurar que el conjunto de entradas tenga media 0 y varianza 1. Hay que tener en cuenta que esta estrategia no se debe aplicar siempre, pues la función de activación escogida depende de la dispersión y media de los datos. Por ejemplo, si la activación es una $ReLU$, entonces las entradas muy grandes tendrán gradiente 1 y las muy pequeñas gradiente 0, luego es adecuado realizar la normalización para obtener mejores resultados. Otro ejemplo serían las funciones de tipo sigmoide, dado que, al normalizar, se transforman los valores muy altos o bajos (que tendrían gradiente 0) en números cercanos al 0, obteniendo gradientes más altos. Para poder llevar a cabo esta técnica la red aprende dos parámetros adicionales para la normalización, la media y la varianza, y otros dos para el reescalado de los resultados tras la normalización.

Además, también existe otro problema, referente a la dimensión de los datos. Las redes profundas tienen más neuronas y más capas, lo que implica mayores tiempos de ejecución, sobre todo durante la fase de entrenamiento. Esto se debe a la gran cantidad de parámetros que han de ser actualizados en cada época, lo cual no se simplifica si los datos tienen una dimensión muy alta, como pueden ser las imágenes o la voz.

Para poder abarcar datos complejos (imágenes, audio o vídeo) es necesario utilizar

técnicas de espacio latente para los modelos profundos, que permiten simplificar la información de los datos mediante la reducción a espacios de dimensión menor. El espacio latente es un espacio de dimensión más pequeña que el original (preferiblemente mucho menor) que busca la abstracción de las características más importantes de los datos, para así poder extraer información de manera más sencilla. Esta información extraída es la que permite a estos modelos generar, adecuadamente, contenido que sigue el estilo de los ejemplos, pero añadiendo modificaciones nuevas sobre los mismos, y asegurando cierta variabilidad sobre el resultado, pues al reducir dimensión perdemos parte de la información inicial. El uso de técnicas de espacio latente ha probado ser también una buena práctica a la hora de evitar el sobreajuste, además de simplificar el entrenamiento de las redes profundas, pues obviando ciertos matices de las entradas, se generalizan aquellos que se consideren más importantes.

Parte II

Autocodificadores variacionales

Capítulo 3

Autocodificadores

Antes de poder explicar las redes principales que se han estudiado en este trabajo, expliquemos los autocodificadores, un tipo de red profunda que utiliza varias redes neuronales para conseguir sus objetivos.

Una red autocodificadora o *autoencoder* es un modelo de aprendizaje no supervisado que busca una representación comprimida de unas entradas dadas con el objetivo de extraer la información que se considera relevante y evitar la que no lo es.

El formato de las entradas para estas redes puede variar desde el habla hasta las imágenes y, generalmente, estas entradas son dadas en forma de un vector, denotado por $\mathbf{x}$. En el caso del habla se utilizan vectores acústicos y en el caso de las imágenes, vectores que representan cada posición y color de cada píxel de la imagen.

Para poder realizar el proceso y comparar si el proceso de compresión realizado se ajusta a las expectativas, se divide el autocodificador en tres partes: un codificador, un decodificador y una función de pérdida.

Sigamos el proceso de funcionamiento de un *autoencoder* para comprender mejor estas partes:

1. Primero, para el proceso de codificación, se realiza una reducción de la dimensión de los datos, d, transformando los datos de entrada en un vector de menor dimensión, d'. Si $f : \mathbb{R}^d \to \mathbb{R}^{d'}$ es la función de codificación a partir de la entrada $\mathbf{x}$ se obtiene una representación $\mathbf{z} := f(\mathbf{x})$ llamada representación latente o simplificada de los datos. La dimensión d' representa el número de características que se desean aprender de los datos de entrada.

2. Una vez realizada la codificación de los datos se trabajará con ellos, de modo que es la representación latente la que entrenará la red. Esto ayuda a ganar velocidad de cómputo cuando los datos manejados tienen dimensiones altas (un claro ejemplo serían las imágenes en alta definición).

3. Utilizada la representación simplificada para obtener información, se han de recuperar las entradas. Para recuperarlas, se utiliza otra función, $g : \mathbb{R}^{d'} \to \mathbb{R}^d$, tal que $g(\mathbf{z}) = \hat{\mathbf{x}}$, la cual se conoce como decodificador.

4. Finalmente, se realiza la comparación entre $\mathbf{x}$ y $\hat{\mathbf{x}}$ mediante una función de pérdida, $\ell(\mathbf{x}, \hat{\mathbf{x}})$. La función ℓ se encargará de medir la falta de similitud entre la decodificación y el dato inicial, de modo que se centrará en cuantificar el valor de la información perdida durante el proceso de codificación de las entradas.

El primer algoritmo en nacer que cumplía con estas características es el análisis de componentes principales (PCA). El objetivo de este algoritmo es reducir la dimensión de los datos, de modo que se descarta información irrelevante y permite trabajar con menos variables, lo que simplifica el problema y el modelado del mismo. Veamos su funcionamiento.

Partiremos de un conjunto de vectores $\mathbf{x}_1, ..., \mathbf{x}_n \in \mathbb{R}^d$. Además, consideraremos $\|\cdot\|$, la norma euclídea en este espacio, cuyo objetivo es medir la distancia entre los vectores iniciales y las salidas producidas. Partiremos de un conjunto de vectores $\mathbf{x}_1, ..., \mathbf{x}_n \in \mathbb{R}^d$. Además, consideraremos $\|\cdot\|$, la norma euclídea en este espacio, cuyo objetivo es medir la distancia entre los vectores iniciales y las salidas producidas. Como queremos reducir la dimensión de los vectores, así que recurrimos a una transformación lineal sobre los mismos. Si $W \in \mathcal{X}_{d',d}$ es la matriz de dicha transformación, tenemos que: $W : \mathbb{R}^d \to \mathbb{R}^{d'}$, donde la transformación específica es $\mathbf{x} \mapsto W\mathbf{x}$, donde $W\mathbf{x}$ es una representación en menor dimensión de $\mathbf{x}$. A su vez, necesitaremos una segunda transformación, en este caso $U \in \mathcal{X}_{d,d'}$, que utilizaremos para aproximar los vectores a partir de sus formas comprimidas. El objetivo es, que a partir de $\mathbf{y} \in \mathbb{R}^{d'}$, $\mathbf{y} = W\mathbf{x}$, se obtenga $\tilde{\mathbf{x}} = U\mathbf{y}$, la aproximación de $\mathbf{x}$.

Nuestro objetivo es, entonces, minimizar la distancia entre salidas y entradas para poder recuperar los datos con la menor pérdida de información. Esto se traduce en:

$$\underset{U \in \mathcal{X}_{d,d'}, W \in \mathcal{X}_{d',d}}{\arg\min} \sum_{i=1}^{n} \|\mathbf{x}_i - UW\mathbf{x}_i\|^2 . \tag{3.1}$$

La solución a este problema es la siguiente: si (U, W) es la solución, podemos tomar U de forma que tenga columnas ortogonales (es decir, $U^tU = I_d$) y $W = U^t$. Para verlo, fijamos U y W y consideramos la aplicación $\mathbf{x} \mapsto UW\mathbf{x}$. La imagen de esta aplicación, $\mathcal{I} = \{UW\mathbf{x} : \mathbf{x} \in \mathbb{R}^d\}$, es un subespacio de dimensión d' de $\mathbb{R}^d$. Sea ahora $V \in \mathcal{X}_{d,d'}$ una matriz cuyas columnas son una base ortonormal del subespacio $\mathcal{I}$, es decir, $V^TV = I_{d'}$. Con esto, cada vector de la imagen se puede escribir como $V\mathbf{y}$ donde $\mathbf{y} \in \mathbb{R}^{d'}$. Para cada $\mathbf{x} \in \mathbb{R}^d$ e $\mathbf{y} \in \mathbb{R}^{d'}$, tenemos que:

$$\|\mathbf{x} - V\mathbf{y}\|^2 = \|\mathbf{x}\|^2 + \mathbf{y}^TV^TV\mathbf{y} - 2\mathbf{y}^TV^T\mathbf{x} = \|\mathbf{x}\|^2 + \|\mathbf{y}\|^2 - 2\mathbf{y}^TV^T\mathbf{x}.$$

Minimizando esta expresión respecto de $\mathbf{y}$ comparándola con el gradiente respecto de $\mathbf{y}$ con 0, obtenemos que $\mathbf{y} = V^T\mathbf{x}$. Entonces, para cada $\mathbf{x}$ se tiene:

$$VV^T\mathbf{x} = \underset{\hat{\mathbf{x}} \in \mathcal{I}}{\arg\min} \|\mathbf{x} - \hat{\mathbf{x}}\|^2 ,$$

lo cual ocurre en particular para nuestro conjunto de vectores $\mathbf{x}_1, ..., \mathbf{x}_n$. Por lo tanto, podemos reemplazar U, W por V, V^T y que el objetivo no sea mayor:

$$\sum_{i=1}^{n} \|\mathbf{x}_i - UW\mathbf{x}_i\|^2 \geq \sum_{i=1}^{n} \left\|\mathbf{x}_i - V, V^T\mathbf{x}_i\right\|^2.$$

Esto lo satisfacen todas las matrices U, W, luego se cumple (3.1).

Con esto, se transforma el problema anterior en el siguiente:

$$\underset{U \in \mathcal{X}_{d,d'}: U^tU = I_d}{\arg\min} \sum_{i=1}^{n} \left\|\mathbf{x}_i - UU^t\mathbf{x}_i\right\|^2. \tag{3.2}$$

Lo cual se puede simplificar aún más. Si $\mathbf{x} \in \mathbb{R}^d$ y U es ortogonal, se tiene que:

$$\left\|\mathbf{x} - UU^t\mathbf{x}\right\|^2 = \|\mathbf{x}\|^2 - 2\mathbf{x}^tUU^t\mathbf{x} + \mathbf{x}^tUU^tUU^t\mathbf{x} =$$
$$\|\mathbf{x}\|^2 - \mathbf{x}^tUU^t\mathbf{x} = \|\mathbf{x}\|^2 - \text{traza}(U^t\mathbf{x}\mathbf{x}^tU).$$

Esto permite volver a transformar el objetivo de (3.2):

$$\underset{U \in \mathcal{X}_{d,d'}: U^tU = I_d}{\arg\max} \text{traza}(U^t \sum_{i=1}^{n} \mathbf{x}_i\mathbf{x}_i^tU), \tag{3.3}$$

pues la traza es un operador lineal.

Para finalizar la explicación del funcionamiento del PCA, veamos como obtener U. Sea $A = \sum_{i=1}^{n} \mathbf{x}_i\mathbf{x}_i^t$. Sabemos que A es simétrica y por ello se puede escribir en su descomposición espectral VDV^t (su descomposición en autovectores y autovalores). Además, se puede asumir, sin pérdida de generalidad que los autovalores son:

$$D_{1,1} \geq D_{2,2} \geq ... \geq D_{d,d} \geq 0,$$

pues A es también semidefinida positiva. La solución, por tanto, es que las columnas de U sean los d' autovectores relacionados con los d' mayores autovalores de A. Esto es porque estos autovectores están relacionados con als direcciones de mayor varianza de los datos, al escogerlos estaremos capturando la mayor cantidad de variabilidad de los datos de partida.

Esto se puede condensar en un resultado que se demuestra también en [34]: sean $\mathbf{x}_1, ..., \mathbf{x}_n \in \mathbb{R}^d$. Sea $A = \sum_{i=1}^{n} \mathbf{x}_i\mathbf{x}_i^t$ y sean $\mathbf{u}_1, ..., \mathbf{u}_{d'}$ los d' autovectores asociados a los d' mayores autovalores mayores de A. La solución al problema del PCA en (3.3) es elegir U de modo que $\mathbf{u}_1, ..., \mathbf{u}_{d'}$ sean sus columnas y elegir $W = U^t$. Veamos su demostración. Sea VDV^T la descomposición espectral de A. Fijamos una matriz $U \in \mathcal{X}_{d,d'}$ con columnas ortonormales, y sea $B = V^TU$. Entonces:

$$VB = VV^TU = U,$$

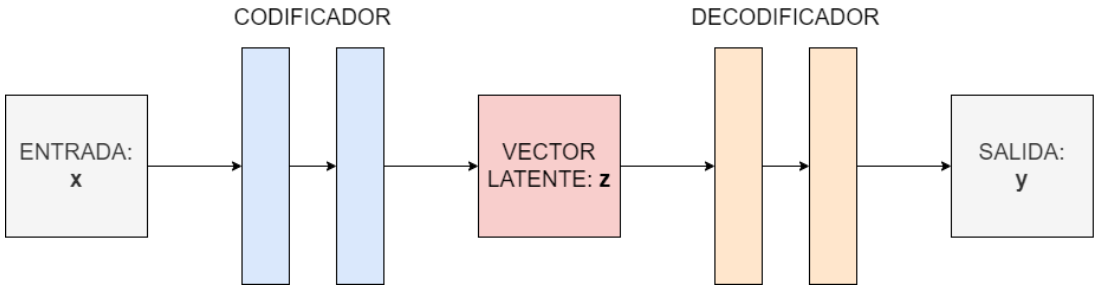

Figura 3.1: Estructura de un *AE*

lo que implica:

$$U^T AU = B^T V^T V D V^T V B = B^T D B.$$

Con esto, se obtiene:

$$\text{traza}(U^T AU) = \sum_{j=1}^{d} D_{j,j} \sum_{i=1}^{d'} B_{j,i}^2.$$

Debemos tener en cuenta que $B^T B = U^T V V^T U = I_d$, por lo que las columnas de B son ortonormales también. Con esto, sabemos que $\sum_{j=1}^{d} \sum_{i=1}^{d'} B_{j,i}^2 = d'$. Además, si $\hat{B} \in \mathcal{X}_{d,d}$ es una matriz tal que sus d' columnas sean las de B y tal que $\hat{B}^T \hat{B} = I_d$, entonces, para cada j tenemos que $\sum_{i=1}^{d} \hat{B}_{j,i}^2 = 1$, lo que implica que $\sum_{i=1}^{d'} \hat{B}_{j,i}^2 \leq 1$. De lo anterior se sigue que:

$$\text{traza}(U^T AU) \leq \max_{\beta \in [0,1]: \|\beta\| \leq d'} \sum_{j=1}^{d} D_{j,j} \beta_j = \sum_{j=1}^{d'} D_{j,j}.$$

Con esto se demuestra lo que buscábamos, pues para cada matriz $U \in \mathcal{X}_{d,d'}$ con columnas ortonormales se tiene que $\text{traza}(U^T AU) \leq \sum_{j=1}^{d'} D_{j,j}$, además de que en el caso en que se escojan los d autovectores de mayor valor de A, obtendremos que $\text{traza}(U^T AU) = \sum_{j=1}^{d'} D_{j,j}$.

Este algoritmo es la implementación más sencilla de lo que llamamos autocodificadores, aunque el rango que abarcan estas redes es mayor.

Además, si tanto el codificador, f como el decodificador, g, son funciones no lineales, se obtiene la ventaja de poder modelar las funciones como redes neuronales. El beneficio principal de la modelización con RRNN es la búsqueda de la minimización de la función de pérdida mediante retropropagación, para lo que solo se necesita que la función de pérdida sea diferenciable.

Hay que tener en cuenta que al usar estas redes, puesto que se utiliza un modelo con reducción de dimensión mediante variable latente, se tiene pérdida de información. Esta pérdida de información del modelo ha de controlarse de modo que no se eliminen rasgos importantes de las entradas y a la vez que la reducción de dimensión sea adecuada para un trabajo eficiente con los datos de entrada.

Tratando la entrada como un vector procedente de una distribución de probabilidad, tendremos un codificador aleatorio de los elementos de la distribución, $p(\mathbf{z}|\mathbf{x})$, y un deco-

dificador, también aleatorio, que procura reconstruir las codificaciones $q(\mathbf{x}|\mathbf{z})$. El resultado que produce el proceso de codificación y decodificación es una aproximación del vector de entrada, así que la denotaremos por $\hat{\mathbf{x}}$. Para medir la distancia entre las aproximaciones y los datos iniciales utilizaremos una pérdida sencilla, el error cuadrático medio:

$$\ell(\mathbf{x}, \hat{\mathbf{x}}) = \frac{1}{d}\sum_{i=1}^{d}(x_i - \hat{x}_i)^2,$$

que no es más que la pérdida cuadrática (*MSE*), donde x_i y $\hat{x}_i$ son las componentes del vector de partida $\mathbf{x}$ y la aproximación $\hat{\mathbf{x}}$.

El objetivo del *autoencoder* será extraer las características más importantes del vector de entradas con esta pérdida de información, para poder realizar predicciones o clasificaciones sobre nuevos datos que se proporcionen.

Algunos ejemplos de utilización de autocodificadores son: la eliminación de ruido o errores de los datos de entrada, la coloración o decoloración de imágenes o la generación de nuevos datos a partir de los de entrenamiento.

La mejora inmediata de estas redes son los *VAE*, que con una estructura parecida utilizan códigos latentes interpretables que generan proyecciones continuas. También utilizan estructuras parecidas las arquitecturas de tipo *Transformer*, muy utilizadas actualmente, y algunos ejemplos de multimodalidad, con el fin de aunar distintos tipos de datos con el mismo espacio de codificaciones.

Capítulo 4

Inferencia variacional

A lo largo de este capítulo introduciremos distintas ideas que conforman la base de la inferencia variacional (VI), y que nos servirán para comprender el concepto de autocodificador variacional y de Wasserstein. El objetivo es partir de los conceptos base de la inferencia y construir finalmente las redes VAE y WAE, tanto teórica como prácticamente.

Antes de centrarnos en este concepto hemos de comprender la siguiente interpretación, dentro del marco de la inferencia Bayesiana, de la ley de probabilidad condicionada (de Bayes):

$$p(z|x) = \frac{p(x|z)p(z)}{p(x)}, \tag{4.1}$$

donde z y x representan eventos de las variables aleatorias Z y X, las variables latentes y las observaciones, respectivamente. A cada elemento de la ley le damos un nombre específico:

- $p(z|x)$ la cual se conoce como distribución a posteriori y refleja el conocimiento sobre los parámetros después de conocer las entradas. Obtener información sobre esta distribución es el objetivo principal de la inferencia Bayesiana, como podemos observar en (4.1).
- $p(x|z)$, la verosimilitud, que mide como de probables son las observaciones según el modelo.
- $p(z)$ es la distribución a priori, que codifica en el modelo cualquier conocimiento previo que poseamos.
- $p(x)$ o la evidencia, es decir, la distribución de los datos observables. Esta distribución funcionará como normalizador de $\int p(x|z)p(z)dz$ de modo que la distribución a posteriori corresponde realmente a una distribución de probabilidad.

Teniendo en cuenta estos conceptos, centrémonos ahora en el propósito de la inferencia variacional (VI).

Sean $x_1, ..., x_n$, las observaciones, y $z_1, ..., z_m$, las variables latentes. Representaremos por $\mathbf{x}$ al vector $(x_1, ..., x_n)$ y por $\mathbf{z}$ al vector $(z_1, ..., z_m)$. También se considera $p(\mathbf{z}, \mathbf{x})$, la

densidad conjunta de ambas. El objetivo principal es construir una aproximación de la densidad condicionada de las variables latentes, $p(\mathbf{z}|\mathbf{x})$, teniendo en cuenta que se conocen los datos observados.

Para poder hallar la solución a este problema, la clave se centra en resolver un problema de optimización, donde se busca el miembro, dentro de una familia de densidades, que provee la densidad más cercana a la densidad condicional que se desea conocer. Para medir la distancia entre estas densidades, generalmente se utiliza la divergencia de *Kullback-Leibler*. Recordemos cómo calcular dicha divergencia:

$$D_{KL}(q,p) = \int q(\zeta)\log(\frac{q(\zeta)}{p(\zeta)})d\zeta.$$

La obtención de la densidad objetivo permitirá producir estimaciones de las variables latentes, además de generar intervalos para nuevos datos, o realizar operaciones con la información subyacente de los datos.

Abordemos el problema con un poco más de claridad. Podemos reescribir la densidad a posteriori de la siguiente manera:

$$p(\mathbf{z}|\mathbf{x}) = \frac{p(\mathbf{z},\mathbf{x})}{p(\mathbf{x})}. \tag{4.2}$$

El problema de esta forma de atacar el problema es el siguiente; obtener la evidencia no es siempre un problema computable.

Se puede pensar que una forma de resolver el problema es hallar la evidencia de la siguiente forma:

$$p(\mathbf{x}) = \int p(\mathbf{z},\mathbf{x})d\mathbf{z}. \tag{4.3}$$

Sin embargo, este cálculo no produce una fórmula cerrada o conlleva tiempos exponenciales para su resolución, como ya habíamos mencionado, luego no es una forma adecuada de proceder.

El principal objetivo de la inferencia variacional es, siguiendo estas pautas, encontrar una alternativa al *MCMC* (*Markov Chain MonteCarlo*) que asuma un coste computacional menor. Asumiendo que las variables latentes son las que presentan las cualidades de interés para poder reproducir los datos u operar con ellos, se introduce el concepto de la cota inferior de la evidencia, como veremos en la sección siguiente. Será una de las principales herramientas que utilizaremos en el marco de la inferencia variacional. Esto es para satisfacer la idea principal detrás de los codificadores variacionales: obtener decodificadores aleatorios para utilizar el espacio latente como generador de nuevos datos; aunque aún es pronto para entrar en detalle.

4.1. El *ELBO*

Una de las herramientas que necesitamos para desarrollar la *VI* es la cota inferior de la evidencia (*Evidence Lower Bound* o *ELBO*).

En el campo de la inferencia variacional, es necesario especificar una familia $\mathcal{Q}$ de densidades sobre las variables latentes. Esta familia es en la que buscaremos la mejor aproximación para la densidad condicionada $p(\mathbf{z}|\mathbf{x})$. Como las densidades dependen de las variables latentes, las denotaremos como $q(\mathbf{z}) \in \mathcal{Q}$.

La inferencia se resume, entonces, en resolver, en términos de la divergencia de *Kullback-Leibler*, el siguiente problema de optimización:

$$q^*(\mathbf{z}) = \arg\min_{q \in \mathcal{Q}} D_{KL}(q(\mathbf{z}), p(\mathbf{z}|\mathbf{x})).$$

Una vez calculada, q^* es la mejor aproximación de la densidad condicionada en $\mathcal{Q}$.

Sin embargo, esta expresión demuestra que no podemos minimizar directamente la divergencia, dado que necesitaríamos el logaritmo de la distribución a posteriori, el cual es intratable. Veamos esa dependencia, fijando $q(\mathbf{z}) \in \mathcal{Q}$:

$$D_{KL}(q(\mathbf{z}), p(\mathbf{z}|\mathbf{x})) = \mathbb{E}(\log[q(\mathbf{z})]) - \mathbb{E}(\log[p(\mathbf{z}|\mathbf{x})]). \tag{4.4}$$

Y si expandimos las esperanzas anteriores:

$$D_{KL}(q(\mathbf{z}), p(\mathbf{z}|\mathbf{x})) = \mathbb{E}(\log[q(\mathbf{z})]) - \mathbb{E}(\log[p(\mathbf{z}, \mathbf{x})]) + \log[p(\mathbf{x})]. \tag{4.5}$$

El último término depende de las observaciones, y, al considerar las esperanzas respecto de $q(\mathbf{z})$, vemos que es constante, luego su esperanza es el propio valor. Esto deja clara la dependencia de la evidencia a la hora de calcular la distancia entre las densidades que nos interesan.

Como nos hemos encontrado el impás de la intratabilidad del cálculo de la distancia entre $q(\mathbf{z})$ y $p(\mathbf{z}|\mathbf{x})$, buscamos otro enfoque para obtener la solución. Para ello, buscaremos un problema de optimización equivalente. Para no tener que utilizar la evidencia, consideramos:

$$ELBO(q) = \mathbb{E}(\log[p(\mathbf{z}, \mathbf{x})]) - \mathbb{E}(\log[q(\mathbf{z})]) \tag{4.6}$$

Maximizar el *ELBO*, descrito como en (4.6), es equivalente a minimizar la divergencia entre ambas densidades. Esto se debe a que la diferencia entre la divergencia y la cota es una constante respecto de las variables latentes, además de aplicar un cambio de signo. Además, es importante notar que es un problema equivalente más sencillo, pues no depende ya de la evidencia, que no podemos calcular, si no sólo de elementos sobre los que tenemos control, $p(\mathbf{z}, \mathbf{x})$, que se puede obtener de las variables latentes, y $q(\mathbf{z})$, que es la densidad con la que aproximaremos la densidad a priori. Buscaremos una densidad $q(\mathbf{z})$ que explique bien los datos y este suficientemente cercana a la distribución a priori, pues es la distribución real detrás de las variables latentes. Para poder hallarla, utilizaremos redes neuronales.

Conviene tener en cuenta que el *ELBO*, también tiene una propiedad muy interesante, y es que acota la logevidencia de la siguiente forma:

$$\log[p(\mathbf{x})] \geq ELBO(q).$$

Esto es consecuencia directa de (4.5):

$$\log[p(\mathbf{x})] = D_{KL}(q(\mathbf{z}), p(\mathbf{z}|\mathbf{x})) + ELBO(q), \tag{4.7}$$

y como la divergencia es siempre no negativa, obtenemos la desigualdad. Esta relación ha llevado a utilizar el *ELBO* como criterio de selección de modelos en el campo de la *VI*. Además, es una herramienta muy útil que nos permite trabajar con los datos que conocemos y realizar inferencia sobre las variables latentes, abandonando así las alternativas no computables.

Sin embargo, existen desventajas y problemas al utilizar este tipo de métodos, así que veamos los principales de los problemas de la *VI*.

Aunque el problema sobre la minimización de la divergencia de *Kullback-Leibler* es equivalente a la maximización del *ELBO*, no es sencillo conocer como de cerca no encontramos del óptimo en la formulación de la cota inferior. Cuando tratamos el problema mediante la divergencia, como sabemos que es positiva, está acotada inferiormente por 0, pero la cota inferior de la evidencia no está acotada. De esto deducimos que la aproximación de D_{KL} hacia 0 nos indica como de cerca estará la aproximación de la distribución a posteriori. En contraparte, no hay manera de saber mediante el valor del *ELBO* la cercanía de nuestra aproximación respecto de la distribución real de los datos, aunque sabemos que existe una convergencia asintótica.

Aparte, la aproximación de un modelo multimodal no es tan buena mediante estas técnicas, dado que la minimización de la divergencia con una asunción de independencia lleva la aproximación hacia una distribución objetivo unimodal.

Como ocurría en el caso del descenso de gradiente, existe una variación estocástica de la inferencia variacional, pero cubrir su funcionamiento escapa el alcance de este trabajo. La variación se puede encontrar en [19].

4.2. Autocodificadores variacionales

Antes de saltar al problema de transporte óptimo, que da lugar a los autocodificadores de *Wasserstein*, vamos a introducir el concepto y funcionamiento de los *VAEs* (*Variational Auto-Encoders*). El objetivo de este capítulo es comprender el funcionamiento de los autocodificadores variacionales y como utilizamos el proceso de codificación y decodificación para poder generar nuevas muestras fieles a los datos de entrenamiento.

Conocemos ya el funcionamiento de los autocodificadores tradicionales, pero recordémoslo brevemente. Un *AE* es una red profunda cuya actividad se basa en codificar la información recibida para poder reducir la dimensión del problema, extraer la información más importante de los objetos codificados y decodificar esta última para recuperar objetos similares a los que se recibieron.

Los *VAEs* son un tipo de modelo generativo y de reducción de dimensión, con el objetivo de generar elementos similares a las entradas mediante RRNN utilizando principalmente el decodificador junto con la distribución de los códigos latentes.

Comencemos explicando la base del problema. Partiremos de $\mathbf{x}$, un elemento dado por un vector aleatorio X que seguirá una distribución P_X, desconocida. También consideraremos la información latente relacionada con las observaciones, representada por $\mathbf{z}$ y generada por otro vector aleatorio Z para el cual elegimos una distribución específica $P(\mathbf{z})$, con densidad $p(\mathbf{z})$, definida en el espacio latente. Tanto los datos como los códigos latentes se representarán como vectores, pues se considera que utilizaremos transformaciones de datos con más de una dimensión.

El objetivo principal de estas redes es encontrar un decodificador $G : \mathcal{Z} \to \mathcal{X}$, que seguirá una distribución $P_G(\mathbf{x}|\mathbf{z})$ y que permita obtener objetos similares a los que ya conocemos en $\mathcal{X}$ a partir de muestras de $p(\mathbf{z})$. El decodificador se escoge dentro de una familia parametrizada y amplia, luego buscaremos uno que sea adecuado para resolver nuestro problema.

Si tenemos Z, mediante la transformación definida por el decodificador obtendremos un elemento $G(Z) = \hat{X}$, otro vector aleatorio, que será el vector aleatorio que definirá la ley de los elementos generados. Lo que buscaremos es que la ley que sigan los distintos elementos generados, $\mathcal{L}(G(Z)) = P_\theta$, sea similar a la ley que generen los elementos que conocemos, $\mathcal{L}(X) = P_X$. La notación de la distribución P_θ no es arbitraria, pero se entenderá mejor cuando parametricemos la familia de decodificadores de forma que sean redes neuronales. Volviendo a la aproximación entre leyes de probabilidades, planteamos su resolución mediante la estimación máximo verosímil:

$$\arg\max_{\theta\in\Theta} \frac{1}{n}\sum_{i=1}^{n} \log(P_\theta(x_i)) = \arg\max_{\theta\in\Theta} \frac{1}{n}\sum_{i=1}^{n} \log(P_\theta(x_i)) - \frac{1}{n}\sum_{i=1}^{n} \log(P_X(x_i)) =$$

$$\arg\min_{\theta\in\Theta} \frac{1}{n}\sum_{i=1}^{n} \log\left(\frac{P_X(x_i)}{P_\theta(x_i)}\right) \simeq \arg\min_{\theta\in\Theta} D_{KL}(P_X, P_\theta).$$

Es decir, el problema que estamos abordando es equivalente a minimizar la divergencia de *Kullback-Leibler* entre las distribuciones de interés.

Por el momento, disponemos del decodificador G, pero este es sólo la mitad de la red, de modo que necesitaremos también un codificador. Esta necesidad nace de que además de generar nuevos elementos, hemos de poder reconstruir los que ya tenemos, pues el entrenamiento de la red para codificar y decodificar estos datos es lo que nos permitirá minimizar la disimilitud entre P_X y P_θ y generar objetos similares a los de entrenamiento. Si lo explicáramos mediante un símil, estaríamos dando completa libertad a una persona que reproduce cuadros (el decodificador) a la hora de realizar una copia de uno, lo cual produciría copias inexactas o lejanas al cuadro que se quiere copiar, pues no intenta reconstruir el cuadro (seguir el proceso de codificación-decodificación).

Entonces, idealmente, para obtener el codificador, seguimos una serie de pasos. Primero, si $P_G(\mathbf{x}|\mathbf{z})$ es la distribución que sigue nuestro decodificador:

$$\int p(\mathbf{z})P_G(\mathbf{x}|\mathbf{z})d\mathbf{z} = p_\theta(\mathbf{x}),$$

con lo que podemos obtener P_θ. Además, también podemos obtener con el decodificador

y la distribución latente, la distribución conjunta:

$$p(\mathbf{z})P_G(\mathbf{x}|\mathbf{z}) = p(\mathbf{x}, \mathbf{z}).$$

Y juntando ambos elementos, podemos obtener, entonces, la distribución del codificador:

$$p_G(\mathbf{z}|\mathbf{x}) = \frac{p(\mathbf{x}, \mathbf{z})}{p_\theta(\mathbf{x})} = \frac{p(\mathbf{z})P_G(\mathbf{x}|\mathbf{z})}{\int p(\mathbf{z})P_G(\mathbf{x}|\mathbf{z})d\mathbf{z}}. \tag{4.8}$$

Sin embargo, esta formulación presenta un problema principal: incluso en el caso ideal, la densidad $p_\theta(\mathbf{x})$ definida mediante la integral no es siempre computable. Para poder obtener el codificador, habrá que buscar otro camino.

Seguimos un nuevo enfoque. Para poder conseguir un codificador, necesitaremos escoger un conjunto $\mathcal{Q}$ de codificadores y buscar en él un representante que difiera poco, en términos de la divergencia de *Kullback-Leibler*, de la distribución ideal $P_G(\mathbf{z}|\mathbf{x})$. Siguiendo esta idea, si $Q(\mathbf{z}|\mathbf{x}) \in \mathcal{Q}$:

$$D_{KL}(Q(\mathbf{z}|\mathbf{x}), P_G(\mathbf{z}|\mathbf{x})) = \mathbb{E}_{Q(\mathbf{z}|\mathbf{x})}\left[\log(Q(\mathbf{z}|\mathbf{x}))\right] - \mathbb{E}_{Q(\mathbf{z}|\mathbf{x})}\left[\log(P_G(\mathbf{z}|\mathbf{x}))\right] = D_{KL}(Q(\mathbf{z}|\mathbf{x}), P(\mathbf{z})) - \mathbb{E}_{Q(\mathbf{z}|\mathbf{x})}\left[\log(P_G(\mathbf{x}|\mathbf{z}))\right] + \log(P_G(\mathbf{x})).$$

De nuevo, replicando el argumento que veíamos para el *ELBO*, como el último sumando es constante, tenemos que el problema de minimización sobre lo anterior es equivalente a la minimización de:

$$D_{KL}(Q(\mathbf{z}|\mathbf{x}), P(\mathbf{z})) - \mathbb{E}_{Q(\mathbf{z}|\mathbf{x})}\left[\log(P_G(\mathbf{x}|\mathbf{z}))\right], \tag{4.9}$$

por lo que el mejor codificador será el que minimice (4.9).

Esto nos lleva a calcular la divergencia variacional entre P_X y P_G para los autocodificadores variacionales, que se centra en encontrar un codificador adecuado:

$$D_{VAE}(P_X, P_G) = \inf_{Q(\mathbf{z}|\mathbf{x})\in\mathcal{Q}} \mathbb{E}_{P_X}\left[D_{KL}(Q(\mathbf{z}|\mathbf{x}), P(\mathbf{z})) - \mathbb{E}_{Q(\mathbf{z}|\mathbf{x})}\left[\log(P_G(\mathbf{x}|\mathbf{z}))\right]\right]. \tag{4.10}$$

El objetivo de los *VAE*s será, entonces, encontrar un buen decodificador G que minimice (4.10). Esto nos llevará entonces al siguiente problema a resolver:

$$\inf_{G\in\mathcal{G}} \inf_{Q(\mathbf{z}|\mathbf{x})\in\mathcal{Q}} \mathbb{E}_{P_X}\left[D_{KL}(Q(\mathbf{z}|\mathbf{x}), P(\mathbf{z})) - \mathbb{E}_{Q(\mathbf{z}|\mathbf{x})}\left[\log(P_G(\mathbf{x}|\mathbf{z}))\right]\right], \tag{4.11}$$

donde $\mathcal{G}$ será una familia de funciones en la que buscar el decodificador. Esta familia será una familia parametrizada, y en nuestro caso, estará compuesta de redes neuronales. Entonces, realmente estaremos buscando una red decodificadora $G_\theta \in \mathcal{G}_\theta$, donde $\theta \in \Theta$ será un parámetro desconocido. Recordemos también cómo llamábamos a la distribución de los datos reconstruidos, P_θ, por lo que buscaremos minimizar, como en (4.11), la divergencia $D_{VAE}(P_X, P_\theta)$.

De forma más general, el objetivo de los *VAE*s es conseguir un error de reconstrucción pequeño a la vez que controlando la desviación con respecto a la distribución a priori

sobre el vector latente, $P(\mathbf{z})$, se respeta un cierto nivel de "regularidad". si reescribimos 4.11 correspondiendo a esta idea:

$$\inf_{G\in\mathcal{G}} \inf_{Q(\mathbf{z}|\mathbf{x})\in\mathcal{Q}} \mathbb{E}_{P_X}\left[-\mathbb{E}_{Q(\mathbf{z}|\mathbf{x})}\left[\log(P_G(\mathbf{x}|\mathbf{z}))\right] + D_{KL}(Q(\mathbf{z}|\mathbf{x}), P(\mathbf{z}))\right], \tag{4.12}$$

es sencillo ver que para G fijo, los VAEs buscan el codificador $Q(\mathbf{z}|\mathbf{x})$ que minimiza el error de reconstrucción, que toma la forma de log-verosimilitud negativa, tratando de forzar que los codificadores no separen la distribución de sus transformaciones de la distribución a priori, $P(\mathbf{z})$.

Sin embargo, la divergencia variacional definida en 4.12 es ideal, pues en el caso práctico partiremos de un conjunto de datos $\{\mathbf{x}_1, ..., \mathbf{x}_n\}$ y será necesario utilizar una versión muestral de la divergencia.

4.3. Autocodificadores adversariales

Dentro de los autocodificadores variacionales pondremos la atención en la arquitectura de los AAEs, los autocodificadores adversariales. Esto es debido a su importancia en el siguiente capítulo del trabajo, pues una de las arquitecturas posibles para los autocodificadores de *Wasserstein* sigue la de estas redes. Utilizaremos una notación similar a la de la sección anterior.

El objetivo de los autocodificadores adversariales es utilizar la inferencia variacional junto con técnicas adversariales, de modo que se pueda conseguir una codificación (distribución a posteriori) de los datos utilizando una distribución a priori arbitraria. De este modo se asegurará que los elementos representados son realmente representativos de los datos, y no sólo de la muestra. Como resultado, el decodificador aprende un modelo generativo profundo que transforma la distribución a priori a la distribución de los datos.

La principal utilidad de este tipo de redes es en el aprendizaje semisupervisado, destramado de estilos y de contenido en imágenes, *clustering* de datos no supervisado, reducción de dimensión y visualización de datos.

En [24] se propuso esta nueva arquitectura. El objetivo principal que presentaba este artículo era la transformación de los AEs en redes generativas, reconstruyendo el error de forma tradicional (como en un autocodificador clásico) y utilizando un criterio adversarial para poder comenzar el proceso con una distribución arbitraria. Mediante repetición del proceso, se producirán representaciones de los datos partiendo de una definición de forma aleatoria, pero según continúen las iteraciones, se acercarán a la evidencia de los datos.

Debido a que explicar por completo el funcionamiento de una red generativa adversarial se escapa al alcance del trabajo, explicamos de forma simplificada el funcionamiento de una. La parte adversarial de la red establece un juego entre dos redes que compiten entre sí, un modelo generativo $G : \mathcal{Z} \to \mathcal{X}$ y un modelo discriminador $D : \mathcal{X} \to [0, 1]$. El discriminador, D, es una red neuronal que calcula la probabilidad de que un punto $\mathbf{x} \in \mathcal{X}$ (muestra positiva) no esté generada por el generador, G (muestra negativa), es decir, calcula la probabilidad de que un elemento sea parte de la muestra de la distribución P_X, y no un elemento $\mathbf{z}$, proveniente de un vector aleatorio Z, que ha sido transformado

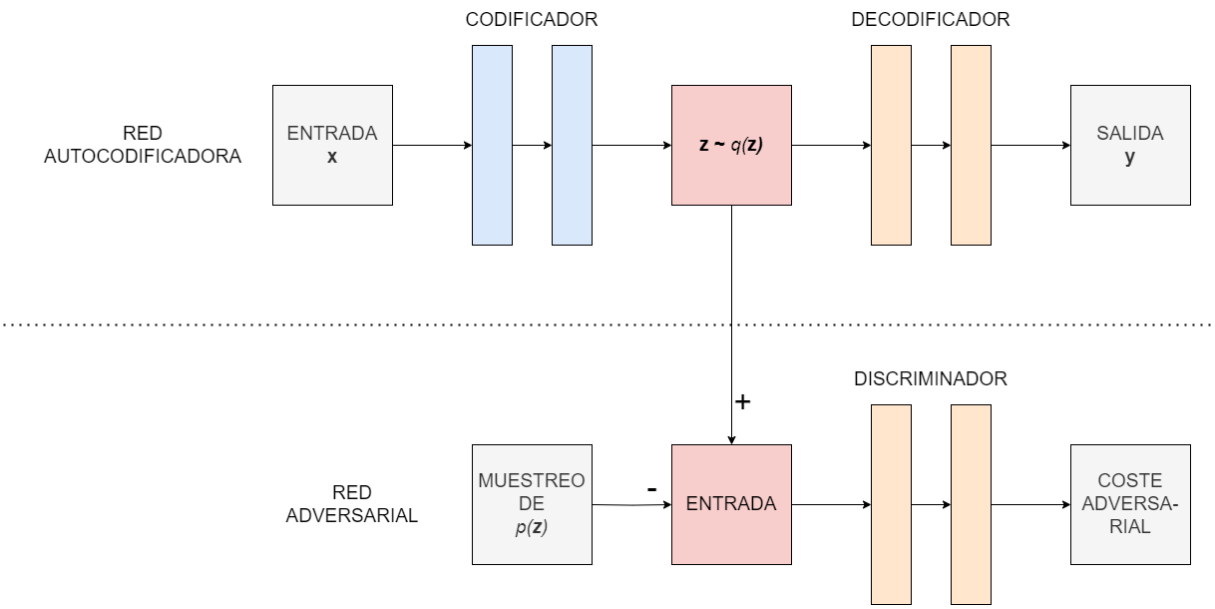

Figura 4.1: Diagrama de arquitectura de un *AAE*

por G. Por otra parte G es otra red que se entrena con el objetivo de confundir a D, su adversario, es decir, su objetivo es hacer que sus transformaciones $G(\mathbf{z}) = \hat{\mathbf{x}}$ no se puedan identificar como transformaciones. Siguiendo este procedimiento, se intenta aproximar la ley de los elementos generados, $\mathcal{L}(G(Z)) = P_\theta$ a la de los datos de entrenamiento, P_X.

Para entrenar tanto generador como discriminador se utiliza el descenso de gradiente estocástico en dos fases distintas:

1. La primera fase entrena al discriminador para diferir entre muestras falsas y verdaderas, como si fuera el entrenamiento de una red clasificadora.

2. La segunda fase congela los parámetros del discriminador y entrena al generador, intentando pasar una imagen falsa como si fuera una verdadera.

De esta forma, las redes compiten y cooperan hasta lograr su objetivo.

Podemos observar la arquitectura de esta red adversarial en la parte inferior del diagrama 4.1.

Este tipo de redes se diferencian de los *VAE*s en distintos puntos. Primero, en la utilización de métodos adversariales en lugar de el uso de divergencias como la D_{KL}. Otra diferencia es el hecho de que permiten realizar el muestreo de la distribución a priori para poder aproximar una distribución arbitraria inicial, $Q(\mathbf{z})$, a $P(\mathbf{z})$, la distribución de los códigos latentes, con el objetivo de no necesitar fijar de antemano la distribución a priori. Su relación con las redes generativas adversariales se observa en la arquitectura de la red completa 4.1.

Comprendiendo el funcionamiento y arquitectura general de los autocodificadores adversariales, veamos cómo es su función objetivo.

El objetivo de los *AAE* se asemeja al de los autocodificadores variacionales, salvo por la sustitución de la divergencia de *Kullback-Leibler* por otro regularizador:

$$D_{AAE}(P_X, P_\theta) = \inf_{Q(\mathbf{z}|\mathbf{x})\in\mathcal{Q}} D_{GAN}(Q(\mathbf{z}|\mathbf{x}), P(\mathbf{z})) - \mathbb{E}_{P_X}\left[\mathbb{E}_{Q(\mathbf{z}|\mathbf{x})}\left[\log P_G(\mathbf{x}|\mathbf{z})\right]\right]. \tag{4.13}$$

En (4.13), definimos D_{GAN} como el objetivo de las redes generativas adversariales, que es el siguiente:

$$D_{GAN}(P, Q) = \sup_{D\in\mathcal{D}} \mathbb{E}_{X\sim P}(\log\left[D(X)\right]) + E_{Z\sim Q}(1 - \log\left[D(G(Z))\right]), \tag{4.14}$$

donde $\mathcal{D}$ es una clase de funciones paramétricas (redes neuronales), D es el discriminador de la red ($D : \mathcal{X} \to [0, 1]$) y G es el generador de la red ($G : \mathcal{Z} \to \mathcal{X}$). La divergencia en (4.13) será el objetivo a minimizar mediante optimizadores de las redes neuronales que forman parte del autocodificador adversarial (como el *SGD*), y podemos ver que es similar a la de los *VAE*, salvo por el regularizador adversarial.

Al igual que en el caso de los *VAE*, la divergencia definida en 4.13 es ideal, pues de nuevo hemos de calcular, en la práctica, la divergencia respecto a una muestra $\{\mathbf{x}_1, ..., \mathbf{x}_n\}$. Se puede ver la definición teórica de esta divergencia junto a otras en [10].

Capítulo 5

Autocodificadores de *Wasserstein*

Para poder explicar el funcionamiento de los autocodificadores de *Wasserstein*, la red central de este trabajo, primero veremos un razonamiento sobre los objetivos de estas redes y explicaremos distintas distancias y divergencias para medir discrepancias entre probabilidades. Tras ello, explicaremos el funcionamiento teórico y luego, en el siguiente capítulo, observaremos dos implementaciones de la red. En [3] se introducían unas redes similares, basadas también en la distancia de *Wasserstein*, que se explicará más adelante. Aunque el problema por el que se preocupaba el artículo [3] es distinto, también se centra en el aprendizaje no supervisado. Principalmente, buscaba responder a la pregunta: ¿Qué significa aprender una distribución de probabilidad?

La respuesta clásica a esta pregunta se limita al aprendizaje de densidades de probabilidad. Esto se lleva a cabo definiendo una familia paramétrica de densidades $(p_\theta)_{\theta\in\mathbb{R}^d}$, sobre la cual se busca maximizar la verosimilitud de los datos iniciales. Si x_i fuera la muestra i, con $i \in \{1, ..., m\}$, entonces se buscaría resolver:

$$\max_{\theta\in\mathbb{R}^d} \frac{1}{m} \sum_{i=1}^{m} \log p(x_i).$$

Si la verdadera distribución de los datos $p(\mathbf{x})$ admitiera una densidad, y siendo P_θ la distribución de una densidad parametrizada, p_θ, entonces el problema es, asintóticamente, equivalente a minimizar la divergencia de *Kullback-Leibler* entre ambas distribuciones.

Este razonamiento presenta un inconveniente, y es que ha de existir la densidad p_θ, lo cual no está garantizado. Además, aunque esta exista, no se garantiza tampoco que esté definida correctamente la divergencia entre distribuciones o que esta tenga un valor finito.

La forma más común de solventar este problema es añadiendo ruido a la distribución del modelo paramétrico. Esta técnica ha sido muy recurrida por los modelos generativos, que generalmente incluyen un componente gaussiano de ruido en los mismos. Sin embargo, el ruido presenta otro problema, por lo menos en el caso de las imágenes; al añadir ruido, se generan muestras borrosas de las mismas.

Veamos otro acercamiento distinto. En vez de estimar la densidad de $p(\mathbf{x})$, se define Z, un vector aleatoria con una distribución fija, $p(\mathbf{z})$, que se transforma por medio de una función paramétrica: $g_\theta : \mathcal{Z} \to \mathcal{X}$, donde partimos del espacio del vector Z y llegamos

al espacio de la distribución objetivo. Esta función, generalmente, se puede implementar con una red neuronal, que generará las muestras siguiendo la distribución P_θ.

Si se hace variar θ, entonces se puede aproximar $p(\mathbf{x})$. Esto es de gran utilidad por dos razones:

1. Se pueden representar distribuciones en baja dimensión de forma codificada.

2. Se pueden generar muestras fácilmente, lo cual es más veloz que la búsqueda numérica de los valores de la distribución objetivo.

De este modo, el artículo pone su atención inicialmente en la forma de medir la distancia entre el modelo y la realidad, y en la forma de definir distancias y divergencias que sirvan a este propósito. El impacto de la selección de una buena distancia recae en la velocidad y convergencia de las secuencias de distribuciones hacia el objetivo.

Para poder ver este impacto, es importante recordar que una secuencia de probabilidades $(P_n)_{n\in\mathbb{N}}$ converge si, y sólo si, existe una probabilidad P_∞ tal que $d(P_n, P_\infty) \to 0$ cuando $n \to \infty$, siendo d una distancia o divergencia entre probabilidades. Como se puede observar, la dependencia de la distancia es estricta, de modo que una buena elección puede solventar muchos de los problemas que se puedan plantear.

Volviendo al problema paramétrico, para optimizar el parámetro θ, será mejor modelar el problema sobre P_θ de modo que la aplicación $\theta \mapsto P_\theta$ sea continua. ¿Por qué necesitamos esta continuidad? Si la aplicación que lleva el parámetro en el modelo es continua, podremos aplicar el criterio secuencial, de modo que se podrán aplicar algoritmos iterativos que lleven a la solución, pues si $\theta_n \to \theta$ $(n \to \infty)$, entonces $P_{\theta_n} \to P_\theta$ cuando $n \to \infty$.

Además de buscar la aplicación de algoritmos iterativos, como puede ser el entrenamiento de una red neuronal, la continuidad nos da otra propiedad; si d es nuestra distancia entre probabilidades, y ℓ es la función de pérdida, tal que $\ell(\theta) = d(P_\theta, p(\mathbf{x}))$, y esta también es continua, entonces, esto es equivalente a que $\theta \mapsto P_\theta$ sea continua utilizando d.

Antes de introducir formalmente el funcionamiento de los autocodificadores de *Wasserstein*, introduzcamos el concepto de distancia de *Wasserstein*, tal y como se introdujo en [3].

5.1. Distancia de *Wasserstein*

Además de para la distancia de *Wasserstein*, aprovecharemos esta sección para introducir otras alternativas, también utilizadas, de distancias y divergencias entre probabilidades.

Sea $\mathcal{X}$ un espacio métrico compacto. Consideremos β el conjunto de todos los subconjuntos de *Borel* de $\mathcal{X}$. Consideremos también $\mathbb{P}(\mathcal{X})$, el espacio de probabilidades de *Borel* sobre $\mathcal{X}$. Se definen las siguientes distancias entre $P, Q \in \mathbb{P}(\mathcal{X})$:

- La distancia de variación total (DVT):

$$\delta(P,Q) := \sup_{A\in\beta} |P(A) - Q(A)|.$$

- La divergencia de *Kullback-Leibler*:

$$D_{KL}(P,Q) = \int \log\left(\frac{p(x)}{q(x)}\right) q(x)d\mu(x),$$

donde P y Q son dos distribuciones absolutamente continuas (admiten densidades) respecto a la medida μ tomada en $\mathcal{X}$. Conviene tener en cuenta que la divergencia puede hacerse infinita en los puntos donde Q se anule y P sea positiva.

- La divergencia de *Jensen-Shannon*:

$$D_{JS}(P,Q) = \frac{1}{2}\left[D_{KL}(P,R) + D_{KL}(Q,R),\right]$$

para la cual se considera $R = \frac{P+Q}{2}$. Esta divergencia goza de la propiedad de simetría y siempre está definida pues se puede escoger $\mu = R$.

- La 1-distancia de *Wasserstein* o de movimiento de pilas de tierra:

$$W(P,Q) = \inf_{\gamma\in\mathcal{P}(P,Q)} \mathbb{E}_{x,y\sim\gamma} \|x-y\|.$$

La distancia de *Wasserstein* o métrica de *Kantorovich-Rubinstein* es una distancia definida entre distribuciones de probabilidad en $\mathcal{X}$.

Intuitivamente, si cada distribución de probabilidad se ve como una pila de tierra dentro del espacio, la métrica es el mínimo coste de transformar una pila en la otra, es decir, la cantidad de tierra a mover por la distancia media a moverla. La p-distancia de *Wasserstein* se define la siguiente manera:

$$W_p(P,Q) = \inf_{\gamma\in\mathcal{P}(P,Q)} [\mathbb{E}_{x,y\sim\gamma}(\|x-y\|^p)]^{\frac{1}{p}},$$

donde P y Q son distribuciones de probabilidad en $\mathcal{X}$.

Para poder definir la distancia se considera el conjunto de emparejamientos de P y Q, es decir, las distribuciones conjuntas con marginales P y Q:

$$\int_{\mathcal{X}} \gamma(x,y)dy = P(x),$$
$$\int_{\mathcal{X}} \gamma(x,y)dx = Q(y).$$

Este problema de las montañas de tierra se entiende mejor desde el punto de vista del transporte óptimo. Si consideramos una distribución P sobre un espacio $\mathcal{X}$, nuestro

objetivo será transformarla en otra distribución Q definida en el mismo espacio. En el caso de las pilas de tierra, el problema tiene sentido únicamente cuando tienen la misma masa, lo cual se puede comprender con las probabilidades, pues ambas distribuciones tendrán masa total igual a 1.

Ahora, para plantear el problema de transporte óptimo necesitaremos una función de coste, que denotaremos por c, tal que $c(x, y) \geq 0$, que nos permite saber el coste de transporte de una unidad de masa, desde el punto x hasta el punto y. El plan de transporte, para que este sea óptimo, ha de ser el de mínimo coste. Este plan lo describimos mediante $\gamma(x, y)$, que nos da la cantidad de masa a mover entre el origen y el destino. Además, ha de satisfacer dos propiedades:

1. La cantidad inicial de masa de la que se dispone ha de ser la cantidad total de masa que movemos:
$$\int \gamma(x, y)dy = P(x).$$
2. La cantidad de masa que tenemos en el destino, al terminar, ha de ser la masa que corresponde:
$$\int \gamma(x, y)dx = Q(y).$$

Lo cual nos pide que γ sea una distribución de probabilidad conjunta con distribuciones marginales P y Q, la cual no podemos asegurar que sea única.

Finalmente, hay que tener en cuenta el coste total del movimiento por unidad de masa, es decir: $c(x, y)d\gamma(x, y)$. Esto produce el coste total del movimiento:

$$\int\int c(x, y)\gamma(x, y)dxdy = \int c(x, y)d\gamma(x, y).$$

Con esta función de coste construimos entonces el problema de optimización, pues necesitamos el menor coste total del movimiento, teniendo en cuenta la no unicidad de γ. Si consideramos $\Gamma = \mathcal{P}(P, Q)$, entonces el coste óptimo será:

$$W(P, Q) = \inf_{\gamma\in\Gamma} \int c(x, y)d\gamma(x, y),$$

lo cual nos lleva a la 1-distancia de *Wasserstein* utilizando como coste la distancia euclídea.

Esta discusión sobre la elección de la distancia induce distintos resultados, cuya demostración se encuentra en los apéndices del trabajo (ver A.1). El primero de los resultados tiene que ver con la continuidad de la 1-distancia de *Wasserstein*. Sea $p(\mathbf{x})$ una distribución fija en $\mathcal{X}$. Sean Z una variable aleatoria sobre $\mathcal{Z}$ y $g : \mathcal{Z} \times \mathbb{R}^d \to \mathcal{X}$ una función de variables z y θ que parametrizaremos respecto de θ fijo ($g_\theta(z)$). Si P_θ es la distribución de $g_\theta(Z)$, entonces:

1. Si g es continua en θ, lo será $W(p(\mathbf{x}), P_\theta)$.
2. Si g es localmente lipschitziana y satisface ciertas condiciones de regularidad, entonces $W(p(\mathbf{x}), P_\theta)$ es continua en todo punto y diferenciable en casi todo punto.
3. Lo anterior no se cumple para D_{JS} ni D_{KL}.

Las condiciones de regularidad que ha de cumplir la función anterior son las siguientes: considerando g como se define en el resultado y bajo la asunción de ser localmente lipschitziana, diremos que satisface las condiciones de regularidad si para una distribución p sobre $\mathcal{Z}$ existen constantes de Lipschitz locales $L(\theta; z)$ tales que:

$$\mathbb{E}_{z\sim p}[L(\theta; z)] < +\infty. \tag{5.1}$$

Además, esto produce el siguiente corolario; si g_θ es una red neuronal parametrizada por θ y $p(z)$ es una distribución a priori con $\mathbb{E}_{\omega\sim p(z)}\left[\|\omega\|\right] < \infty$, entonces se satisfacen 1 y 2 en el teorema anterior.

Esto justifica el uso de redes neuronales junto con la distancia de *Wasserstein*, que nos permitirá conseguir un método iterativo para hallar la solución y una función de pérdida (la distancia) que no sólo será continua, si no diferenciable en casi todo punto.

Conviene también hablar de lo que es una f-divergencia. Para mantener la brevedad, sabemos que la discrepancia entre dos probabilidades P y Q se puede medir mediante divergencias. De este modo se define la clase de f-divergencias por:

$$D_f(P, Q) = \int f\left(\frac{p(x)}{q(x)}\right) q(x)dx, \tag{5.2}$$

donde $f : (0, \infty) \to \mathbb{R}$ es una función convexa que satisface $f(1) = 0$. Un ejemplo clásico de f-divergencias es la divergencia de *Kullback-Leibler* que utiliza la función log, una función que cumple con los requisitos impuestos para ser una f-divergencia.

5.2. Proceso de inferencia

Abordemos ahora el funcionamiento teórico de los autocodificadores de *Wasserstein*.

En [39] se introdujo este nuevo tipo de red, un algoritmo generativo para la distribución de los datos basado en la distancia de *Wasserstein*. Lo que buscaban estas redes era minimizar una penalización de la distancia entre la distribución del modelo y la distribución objetivo. Esto daba lugar a una regularización distinta a la presentada por los *VAE*s, y el nuevo regularizador intentaba que la distribución codificada se aproximara, durante el entrenamiento, a la distribución a priori.

En el artículo, también se comparaban los modelos *WAE* y *AAE* y se demostraba que el primero generaliza al segundo. Además, las nuevas redes compartían propiedades con los autocodificadores variacionales, aunque producían muestras de mejor calidad.

Conocemos ya los *VAE*s y las redes *GAN*, ambas aproximaciones bien establecidas para conocer la distribución de los datos. Sin embargo, estas aproximaciones presentan

varios problemas: los autocodificadores variacionales generan muestras borrosas, y las redes generativas no tienen codificador, luego trabajan con los datos originales, son más difíciles de entrenar y sufren de colapso modal, es decir, el modelo es incapaz de capturar la variabilidad completa de la distribución de los datos.

Entonces, se intenta un nuevo acercamiento, esta vez desde el punto de vista del transporte óptimo, siguiendo las ideas en la sección anterior. Con esto se consigue una red generativa que aúna las dos anteriores, consiguiendo así reducir el impacto tanto de las desventajas de los *VAE*s como de las *GAN*s. Este nuevo acercamiento tiene un mejor comportamiento que las redes anteriores, aunque necesita de restricciones o regularizaciones en la función objetivo. Además consigue medir la distancia entre dos distribuciones de probabilidad con mejores resultados.

Aún así, el objetivo principal para generar contenido sigue siendo el mismo que los autocodificadores ya presentados: es suficiente con fijar una distribución latente, $p(\mathbf{z})$, y entrenar un decodificador, $q(\mathbf{x}|\mathbf{z})$, que lleve elementos desde la distribución latente hasta el espacio de los datos, de modo que se generen nuevas muestras que se parezcan a los datos de entrenamiento.

Centrémonos en el objetivo principal. El problema principal que se intenta solventar con los autocodificadores de *Wasserstein* será minimizar la discrepancia entre:

1. La distribución de los datos $p(\mathbf{x})$.
2. La distribución del modelo P_G, cuya densidad es p_G.

Sin embargo, la mayoría de las divergencias no son computables, más en el caso en que la distribución de los datos es desconocida y el modelo se parametriza con redes neuronales profundas, lo que resulta el caso más común.

Los *VAE*s buscan minimizar la divergencia de *Kullback-Leibler*, aunque más generalmente, minimizan f-divergencias, $D_f(p(\mathbf{x}), P_G)$. Para poder resolver estos problemas, se suele recurrir a estructuras tipo *AAE*s para obtener los valores de la divergencia mediante autocodificadores y *f-GAN*s. Alejándose de este acercamiento están los *WAE*s, que se centran en el coste de problemas de transporte óptimo, utilizando distancias de *Wasserstein* y la dualidad de *Kantorovich-Rubinstein* (que se puede ver detalladamente en el apéndice A.2). Aún así, la arquitectura general es similar a la de los autocodificadores adversariales.

El trabajo [39] se centra en el uso de modelos de variable latente definidos mediante dos pasos:

1. Z, el código, se toma como muestra de una distribución, $p(\mathbf{z})$, en un espacio latente $\mathcal{Z}$.
2. Z se transforma para aproximar a la imagen $X \in \mathcal{X} = \mathbb{R}^d$ con una transformación aleatoria.

El objetivo de estos pasos es producir el resultado siguiente:

$$p_G(\mathbf{x}) = \int_{\mathcal{Z}} p_G(\mathbf{x}|\mathbf{z})p(\mathbf{z})d\mathbf{z}, \ \forall\, \mathbf{x} \in \mathcal{X}.$$

El resultado es una densidad, asumiendo que todos los elementos de la integral están bien definidos. Con esta densidad, tendremos un modelo decodificador con el que generar nuevas muestras a partir de códigos latentes.

Nos centraremos entonces en modelo generativos no aleatorios, que transforman de forma determinista Z a $\hat{X} = G(Z) \sim X$, para una función $G : \mathcal{Z} \to \mathcal{X}$ y variables aleatorias Z y X. Además, utilizaremos la notación $\mathbf{z}$ para los vectores que se tomen de muestra de Z.

A través de esta función, el problema de transporte óptimo toma una forma más simple, pues en vez de buscar una distribución conjunta con marginales adecuadas, como en la anterior sección, es suficiente con hallar una distribución condicional, $q(\mathbf{z}|\mathbf{x})$ tal que su marginal respecto de Z sea:

$$q_Z(\mathbf{z}) := \mathbb{E}_{X \sim p(\mathbf{x})} \left[q(\mathbf{z}|\mathbf{x}) \right],$$

y que esta sea idéntica a la distribución a priori $p(\mathbf{z})$. Esta distribución será la distribución de codificación.

Para continuar, se presenta el siguiente resultado: para P_G definida como antes con $P_G(\mathbf{x}|\mathbf{z})$ determinista y cualquier función $G : \mathcal{Z} \to \mathcal{X}$, se tiene que:

$$\inf_{\gamma \in \mathcal{P}(p(\mathbf{x}), P_G)} \mathbb{E}_{x,y \sim \gamma}(c(x,y)) = \inf_{q : q_Z = p(\mathbf{z})} \mathbb{E}_{p(\mathbf{x})} \mathbb{E}_{q(\mathbf{z}|\mathbf{x})} \left[c(x, G(\mathbf{z})) \right], \tag{5.3}$$

donde q_Z es la distribución marginal de Z cuando $X \sim p(\mathbf{x})$ y $Z \sim q(\mathbf{z}|\mathbf{x})$. Utilizamos también $\mathcal{P}(p(\mathbf{x}), P_G)$ para denotar el conjunto de probabilidades conjuntas con marginales $p(\mathbf{x})$ y P_G. En el caso que nos atañe, $P_G(\mathbf{x}|\mathbf{z})$ será el decodificador que buscamos para generar elementos. La demostración de (5.3) se puede encontrar en el apéndice B de [39].

Lo anterior nos permite optimizar sobre codificadores aleatorios, del tipo $q(\mathbf{z}|\mathbf{x})$, en vez de parejas de variables aleatorias X, Y. Para encontrar una solución numérica, se relajan las restricciones sobre q_Z, añadiendo una penalización sobre el objetivo, de modo que se simplifica la búsqueda del óptimo y obtenemos:

$$D_{WAE}(p(\mathbf{x}), P_G) = \inf_{q(\mathbf{z}|\mathbf{x}) \in \mathcal{Q}} \mathbb{E}_{p(\mathbf{x})} \mathbb{E}_{q(\mathbf{z}|\mathbf{x})} \left[c(x, G(\mathbf{z})) \right] + \lambda D_Z(q_Z, p(\mathbf{z})).$$

En esta divergencia se observan dos nuevos términos; $\mathcal{Q}$, el conjunto no paramétrico de codificadores probabilísticos, y λ es un hiperparámetro de escala para una divergencia arbitraria. Para modelar los codificadores y decodificadores que se utilizan, se utilizan, generalmente, redes neuronales profundas. En contraposición a los *VAE*s, los *WAE*s permiten a los codificadores no aleatorios transformar las entradas de forma determinista a sus códigos latentes.

Aun así, la elección de la penalización para la divergencia no es arbitraria. Se proponen dos penalizaciones principales para la red:

- La primera elección es una penalización con la divergencia de *Jensen-Shannon*, a la que se añade la utilización de métodos adversariales para el entrenamiento de la red para calcularla. Este aproximación da lugar al algoritmo *WAE-GAN*, que supone

una mejora para las redes *GAN*, debido a que trabaja mejora con distribuciones multimodales que estas. La función objetivo para este caso será:

$$D_{WAE}(p(\mathbf{x}), P_G) = \inf_{q(\mathbf{z}|\mathbf{x})\in\mathcal{Q}} \mathbb{E}_{p(\mathbf{x})}\mathbb{E}_{q(\mathbf{z}|\mathbf{x})}\left[c(\mathbf{x}, G(\mathbf{z}))\right] + \lambda D_{GAN}(q_Z, p(\mathbf{z})),$$

donde D_{GAN} es la divergencia utilizada por las redes generativas adversariales como veíamos en (4.14).

- La segunda penalización se basa en el método *MMD* (*Maximum Mean Discrepancy*). Veamos como se calcula el *MMD*, aunque primero explicaremos que es un núcleo. Si $\mathcal{X}$ es un espacio de *Hilbert*, consideramos:

$$\begin{aligned}\phi :&\mathcal{X}\longrightarrow\mathcal{H}\\ &\mathbf{x}\longmapsto\phi(x),\end{aligned}$$

de modo que $\langle \mathbf{x}, \mathbf{y}\rangle \mapsto k(\mathbf{x}, \mathbf{y}) = \langle\phi(\mathbf{x}), \phi(\mathbf{y})\rangle$ define la función k para pares de elementos del espacio de partida. Es esta función k a la que llamamos núcleo, la transformación de un producto escalar a través de una aplicación ϕ, sin restricciones sobre si esta es lineal o no. Para un núcleo $k : \mathcal{Z}\times\mathcal{Z}\to\mathbb{R}$, se tiene que:

$$MMD_k(p(\mathbf{z}), q_Z) := \| \int_{\mathcal{Z}} k(\mathbf{z}, \cdot)dp(\mathbf{z}) - \int_{\mathcal{Z}} k(\mathbf{z}, \cdot)dq_Z(\mathbf{z})\|_{\mathcal{H}_k},$$

donde $\mathcal{H}_k$ es un espacio de *Hilbert* reproductor de núcleos k de funciones reales que transforman $\mathcal{Z}$ en $\mathbb{R}$. Si se quiere ahondar en el concepto de núcleo y su funcionamiento, se recomienda leer [14], pues explica el concepto y funcionamiento de los espacios de *Hilbert* de reproducción de núcleos, que son los utilizados en esta penalización.

Se propone, entonces, $D_Z(q_Z, p(\mathbf{z})) = MMD_k(q_Z, p(\mathbf{z}))$, lo que nos permite utilizar el descenso de gradiente estocástico para optimizar el modelo. Esto transforma la función objetivo en:

$$D_{WAE}(p(\mathbf{x}), P_G) = \inf_{q(\mathbf{z}|\mathbf{x})\in\mathcal{Q}} \mathbb{E}_{p(\mathbf{x})}\mathbb{E}_{q(\mathbf{z}|\mathbf{x})}\left[c(\mathbf{x}, G(\mathbf{z}))\right] + \lambda MMD_k(p(\mathbf{z}), q_Z),$$

El modelo que nace de este método se conoce como *WAE-MMD*, y tiene la ventaja de obtener buenos resultados para grandes dimensiones (como pueden ser las imágenes en alta resolución). El núcleo que utilizaremos para la *WAE-MMD* se detallará durante la implementación de la red (ver (6.1)).

De nuevo, conviene puntualizar que estas divergencias dependen de integrales que en la práctica se transforman en cálculos sobre la muestra $\{\mathbf{x}_1, ..., \mathbf{x}_n\}$, por lo que son válidas para desarrollar la teoría, pero tomar otra forma en la práctica.

Con estas dos variantes de la red podremos obtener la distribución de los datos para generar nuevas muestras. El siguiente paso será implementar ambas para obtener resultados, y comprobar que los resultados que produzcan sean suficientemente buenos. Se puede observar la implementación, así como diagramas de la arquitectura de estos algoritmos en las próximas secciones (6.2 y 6.3).

Capítulo 6

Entrenamiento y resultados

En este capítulo nos centraremos en reproducir los experimentos de [39], utilizando el conjunto de datos *MNIST*.

En el artículo, se buscaban tres objetivos mediante los experimentos:

1. Reconstrucciones precisas de los datos (mediante la generación de imágenes del conjunto de datos).

2. Encontrar geometrías adecuadas para el espacio latente.

3. Obtener muestras de la distribución objetivo de suficiente calidad.

El modelo que obtengamos ha de generalizar bien, es decir, los dos primeros objetivos han de conseguirse tanto en el conjunto de entrenamiento como en el conjunto de prueba de la red.

En los experimentos, los espacios latentes son euclídeos ($\mathcal{Z} = \mathbb{R}^d$), donde la dimensión de estos es dependiente de la complejidad del conjunto de datos. Además también se utilizan distribuciones a priori gaussianas y coste cuadrático para el coste de transporte óptimo.

En los experimentos se utilizan codificadores y decodificadores deterministas, junto con el optimizador *ADAM* y redes convolucionales profundas para el proceso de codificación y decodificación. Debido a que se utilizan codificadores deterministas, escoger una dimensión latente mayor que la dimensión del conjunto de datos forzaría que fuera imposible igualar la distribución a priori y la del codificador. Esto puede llevar a inestabilidad en el entrenamiento, por lo que utilizaremos la dimensión latente recomendada en [39] para *MNIST*: $d' = 8$.

6.1. *Dataset MNIST*

Se puede obtener más información del *dataset MNIST* en la web [22].

El *dataset MNIST* se originó en [9], como respuesta a un problema generado en una competición de clasificación automática de imágenes.

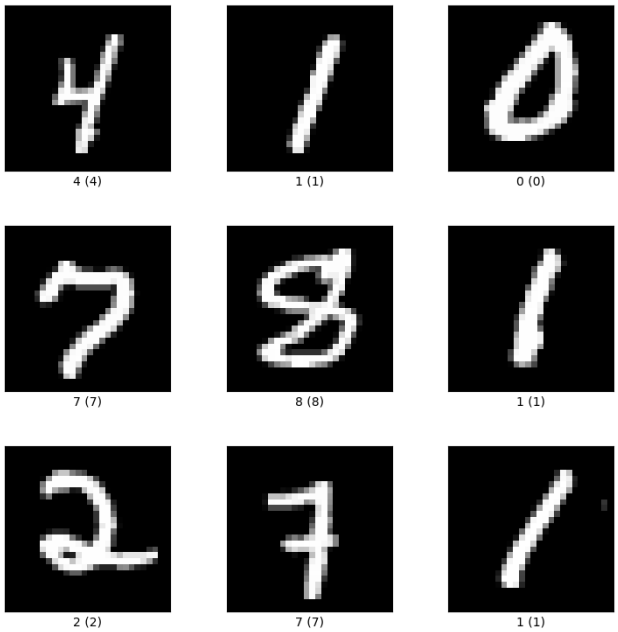

Figura 6.1: Imágenes de ejemplo de *MNIST*, tomadas de https://www.tensorflow.org/datasets/catalog/mnist?hl=es-419

El *US NIST* o, simplemente, *NIST*, es el Instituto Nacional de Estándares y Tecnología estadounidense. Este instituto produjo un conjunto de datos de entrenamiento y otro de validación llamados *NIST SD-3* y *NIST Test Data 1*, consistentes de caracteres escritos a mano, y organizó una competición para encontrar el mejor algoritmo de clasificación sobre escritura realizada a mano.

Los algoritmos de clasificación de la época demostraron ser muy buenos en la clasificación de los elementos del conjunto de entrenamiento, cuando se tomaba la validación de los mismos en este conjunto también. Sin embargo, al probar con el conjunto de validación dado por el *NIST*, se comprobaba que los resultados del entrenamiento de los algoritmos no eran del todo correctos, pues fallaban.

El problema de este conjunto de datos era el siguiente: el muestreo realizado para obtener los caracteres a mano alzada de ambas partes provenía de distintas distribuciones de probabilidad. Mientras que el conjunto de entrenamiento había sido realizado por trabajadores del censo estadounidense, el conjunto de validación se había generado con muestras de escritura realizada por estudiantes de universidad.

Los autores de [9] se dieron cuenta de que esto generaba, no sólo el problema de el origen de los datos, si no que, como los algoritmos de clasificación se basaban en la minimización del riesgo empírico, donde el número de parámetros se ajusta a la cantidad y complejidad de los datos, y ser las distribuciones distintas, los algoritmos no servían

sobre la partición dada. Con este problema a la vista, la solución que se propuso fue la siguiente: había que mezclar de nuevo todos los datos y reparticionar el conjunto completo lo que solventaba el problema de las dos distribuciones, pues la mezcla genera una nueva y única distribución de los datos.

De este modo se obtenía *Modified NIST training and test sets*, o, comúnmente, *MNIST*, un conjunto de datos que permitía aplicar los algoritmos de la época en la clasificación de caracteres escritos.

Abandonando el carácter histórico del *dataset*, centrémonos en sus cualidades.

Este conjunto de datos dispone de 70000 dígitos escritos a mano alzada, de los cuales 60000 se dedican al conjunto de entrenamiento y 10000 al de validación, con el objetivo de tener un conjunto de datos sencillo para el reconocimiento de patrones en imágenes.

Los dígitos de este *dataset* están normalizados en tamaño y centrados en la imagen, lo que facilita el reconocimiento de patrones. Esta es una cualidad que nos permite acelerar el entrenamiento de nuestras redes, pues eliminamos la necesidad de realizar preprocesado sobre las imágenes, y también permite esquivar el formateado de las mismas para que se adecúen a nuestra red. Todas las imágenes del conjunto son de tamaño 28×28 píxeles, lo cual también nos ayuda en el entrenamiento de nuestras redes, dado que no será necesario reescalar las imágenes que no se ajusten al tamaño de entrada, si no que, como todas tienen el mismo tamaño, basta únicamente con definir bien el tamaño de la entrada.

Aunque las redes que implementamos no se han diseñado para el reconocimiento de patrones, estrictamente hablando, es en esta capacidad de reconocer cualidades de las imágenes en la que se basa el aprendizaje de las redes generativas, pues lo que buscan es aprender las cualidades significativas de los elementos del conjunto, de modo que se puedan reproducir en nuevos elementos.

6.2. *WAE-GAN*

Tras explicar los detalles del conjunto de datos con que trabajaremos, procedemos a explicar el experimento que se ha realizado, las capas que se han utilizado en las arquitecturas de las redes y a mostrar los resultados de cada red entrenada, comparándolos con ejemplos del conjunto de datos.

Comenzaremos explicando el funcionamiento de la red autocodificadora de *Wasserstein*, utilizada junto a la arquitectura adversarial. Esta red se conoce como *WAE-GAN*.

La primera pieza que necesitaremos es un codificador, pues es la primera que interactuará con los datos de entrada. El codificador consta de cinco capas; las cuatro primeras serán capas de convolución, para poder aprender y tratar bien las imágenes, la última capa será una capa lineal, para poder transformar los mapas de características que produzcan las convoluciones en un vector, que será la salida codificada de esta red.

Sin embargo, no podemos conectar una salida de una capa a la entrada de la siguiente de forma inmediata. Es por eso que la salida de cada una de las capas de convolución tiene una función de activación *ReLU*, además de utilizar normalización de lotes en las capas ocultas.

El proceso que seguirá una imagen del conjunto de datos de entrenamiento será el siguiente:

1. Primeramente, la imagen se transformará en un mapa de características por la primera capa. Una vez se haya terminado el proceso de convolución, el resultado pasará por una función de activación.

2. Los siguientes tres pasos son idénticos, la entrada de la anterior capa volverá a sufrir una convolución, a cuya salida se aplicará una normalización de lotes y una activación.

3. Finalmente, se tomará la salida de la última capa oculta y se transformará por una capa lineal en un vector del espacio latente.

El diagrama del codificador se puede observar en mayor detalle en la figura 6.2.

Ya tenemos el codificador, luego lo que necesitamos es poder interpretar sus salidas. Por eso mismo, la segunda parte de nuestra red será el decodificador. El decodificador ha de deshacer los cambios producidos por el proceso de codificación, luego tendrá cuatro capas; la primera para transformar los vectores latentes en matrices (imágenes), y las siguientes para decodificar la entrada de la red en una imagen del conjunto de datos.

Vuelve a presentarse el mismo problema que en el caso anterior, no se puede conectar la salida de una capa a la siguiente de forma directa, luego volveremos a utilizar funciones de activación y normalización de lotes entre las distintas capas de la red decodificadora.

Para decodificar un vector latente seguiremos los siguientes pasos:

1. El proceso inicial es transformar el vector de entrada en una matriz que será la entrada de las capas de convolución traspuesta. La salida de esta capa estará controlada por una función de activación.

2. Utilizaremos dos convoluciones traspuestas con normalización de lotes y activación $ReLU$ como capas ocultas.

3. Para finalizar, se aplicará una última convolución traspuesta que nos devolverá una imagen del tamaño de las del conjunto de entrenamiento, esta salida estará controlada por una activación sigmoide.

Las capas de convolución traspuesta no son más que una forma de invertir la convolución realizada en la codificación de las imágenes del *dataset*.

Podemos observar el diagrama del decodificador en la imagen 6.3.

Disponemos ya de la parte autocodificadora de la arquitectura, luego la única pieza restante es la arquitectura generativa adversarial. Para conseguirla, recordemos que el generador será el codificador, luego lo único que queda por estructurar es el discriminador. El discriminador trabajará directamente sobre las codificaciones, de modo que no será necesario realizar convoluciones traspuestas, lo que aumenta la velocidad de su entrenamiento, dado que las convoluciones son operaciones que requieren de mucho tiempo para su entrenamiento. Al igual que en el caso de codificación y decodificación, necesitaremos

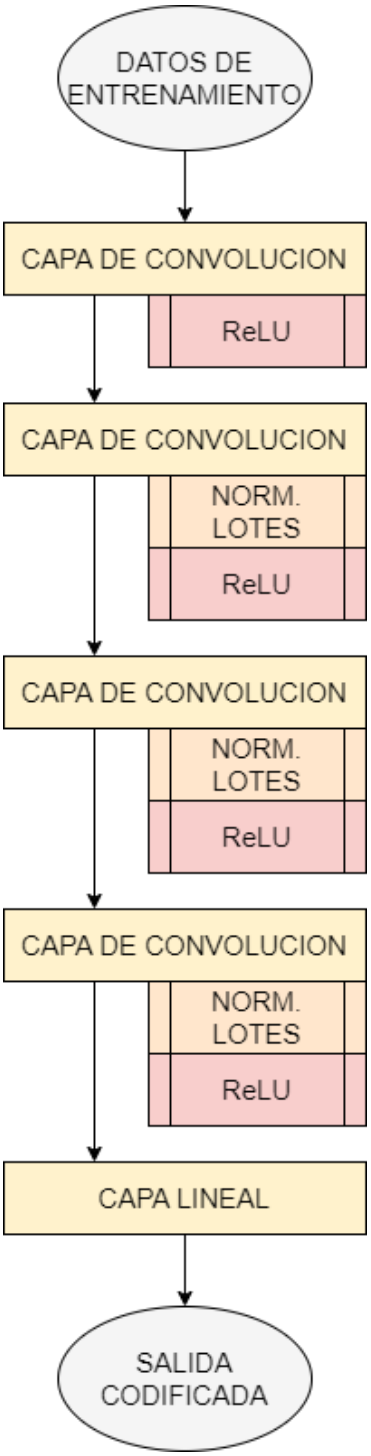

Figura 6.2: Arquitectura del codificador de la red *WAE*

utilizar funciones de activación entre capas, aunque en este caso sólo utilizaremos capas lineales y no convoluciones.

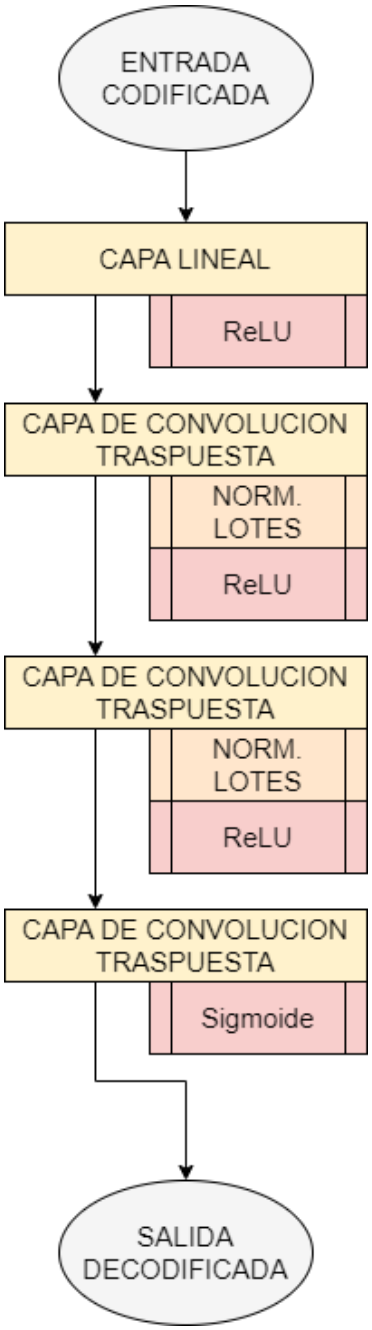

Figura 6.3: Arquitectura del decodificador de la red *WAE*

Una vez disponemos de la entrada codificada, ya sea generada o proveniente de los datos, seguiremos un proceso muy sencillo hasta llegar al veredicto; la entrada pasará por cinco capas lineales controladas por una activación $ReLU$. Si se quiere saber más del funcionamiento de una red generativa tradicional, lo que puede ayudar a entender la diferencia en el funcionamiento de esta parte, se puede recurrir a [18].

El esquema del funcionamiento de la red discriminadora se puede ver en la figura 6.4.

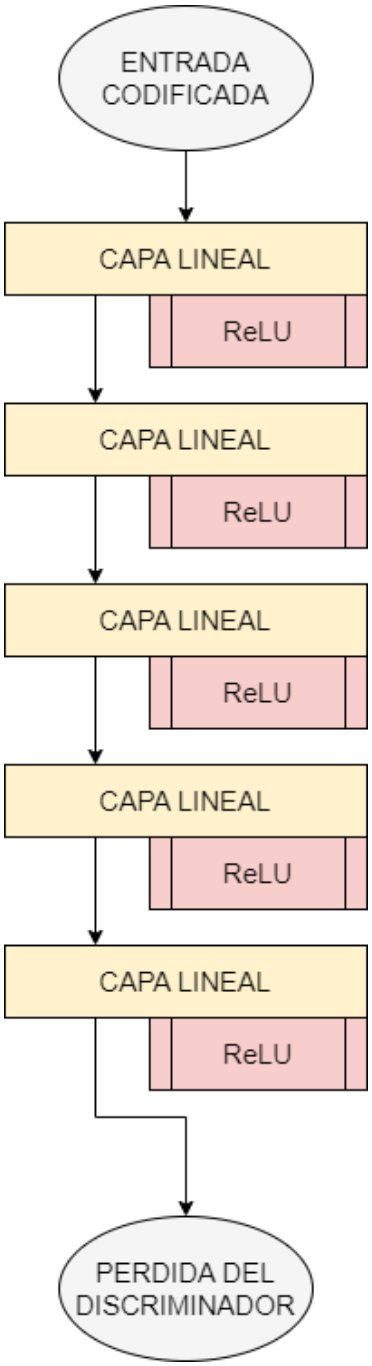

Figura 6.4: Arquitectura del discriminador de la red *WAE-GAN*

6.3. *WAE-MMD*

La segunda arquitectura implementada se basa en el uso del MMD, siguiendo una estructura de autocodificador aunque, sin la parte adversarial.

Aunque las redes no mantienen una estructura idéntica (por la parte adversarial), las

partes codificadora y decodificadora son las mismas para ambas redes.

De este modo, el codificador de la red *WAE-MMD* será igual que el de la red *WAE-GAN*, que podíamos observar en la imagen 6.2.

Recordemos que el proceso que seguía era el siguiente:

1. Primeramente, la imagen se transformará en un mapa de características por la primera capa. Una vez se haya terminado el proceso de convolución, el resultado pasará por una función de activación.

2. Los siguientes tres pasos son idénticos, la entrada de la anterior capa volverá a sufrir una convolución, a cuya salida se aplicará una normalización de lotes y una activación.

3. Finalmente, se tomará la salida de la última capa oculta y se transformará por una capa lineal en un vector del espacio latente.

Y como el codificador es idéntico, también ha de serlo el decodificador, que seguirá los mismos pasos:

1. El proceso inicial es transformar el vector de entrada en una matriz que será la entrada de las capas de convolución traspuesta. La salida de esta capa estará controlada por una función de activación.

2. Utilizaremos dos convoluciones traspuestas con normalización de lotes y activación $ReLU$ como capas ocultas.

3. Para finalizar, se aplicará una última convolución traspuesta que nos devolverá una imagen del tamaño de las del conjunto de entrenamiento, esta salida estará controlada por una activación sigmoide.

Y que veíamos en la figura 6.3.

La otra parte diferente de esta arquitectura es el núcleo reproductor. En [39] se dan recomendaciones para la construcción de las redes, en especial, se recomienda un núcleo reproductor para la implementación de la red. Específicamente, el núcleo que se recomienda para el buen funcionamiento es el núcleo racional cuadrático:

$$k(\mathbf{x}, \mathbf{y}) = \frac{2d'\sigma_z^2}{2d'\sigma_z^2 + \|\mathbf{x} - \mathbf{y}\|^2}, \tag{6.1}$$

donde $d' = 8$, pues es la dimensión latente, y σ_z^2 es la varianza de la distribución latente, que escogeremos a priori de forma que sea una distribución multivariante gaussiana.

6.4. Resultados del entrenamiento

Para implementar y entrenar las redes que se han presentado en las anteriores secciones, nos hemos basado en los programas diseñados en este repositorio.

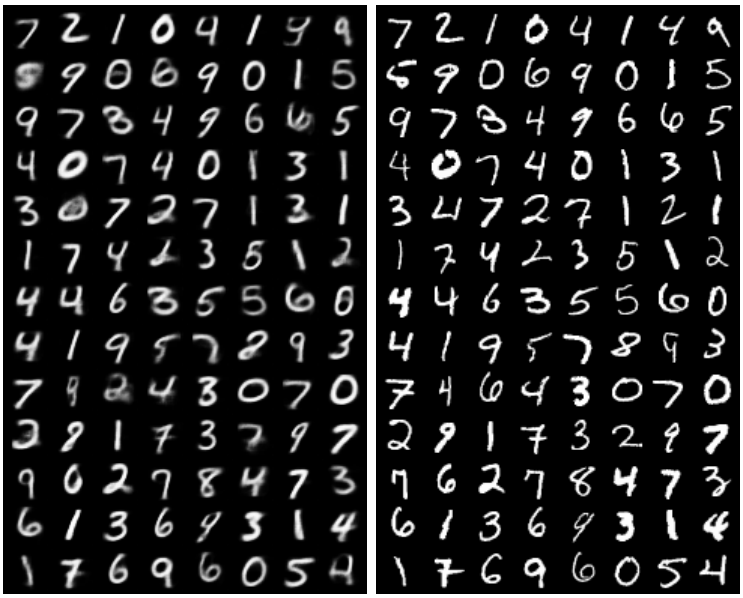

Figura 6.5: A la izquierda la reconstrucción de la red *WAE-GAN*, a la derecha los ejemplos que se intentan reconstruir de *MNIST*

El código implementado se puede encontrar en el siguiente repositorio de *GitHub*: TFG-Autocodificadores-Wasserstein.

Para realizar el entrenamiento, se han realizado 100 épocas con lotes de 100 imágenes en una tarjeta *NVIDIA 950M GTX*, lo que ha llevado 13 minutos de media por época, en un total de 21 horas, aproximadamente, de entrenamiento para cada red. Además, las imágenes generadas por el código se han dividido en dos carpetas para poder generar animaciones que muestran la evolución de la reconstrucción.

Con el objetivo de ver la reconstrucción conseguida con cada una de las redes de los datos, se mantuvo fija una semilla aleatoria, y podremos ver la comparativa entre ambas redes y entre los resultados de las redes y del conjunto de datos para llegar a las conclusiones.

Veamos primero la labor final de reconstrucción conseguida por la red *WAE-GAN*.

Como se puede observar de la figura 6.5, los resultados que se obtienen se asemejan pero son mucho más borrosos que los ejemplos, y, además, se ve que, en la mayoría de casos, la reconstrucción no consigue su objetivo, transformando algunos números en otros. A mayores, se observa como la reconstrucción realizada por esta red genera muestras no legibles, de objetos que no parecen números.

Ahora, observemos la comparación entre los ejemplos de reconstrucción y la reconstrucción generada por la otra red (*WAE-MMD*).

Como podemos observar en la figura 6.6, la reconstrucción de los números es más nítida en este caso, para el mismo número de épocas, que en el caso anterior. Aunque en este segundo caso también encontramos que la reconstrucción es fallida para algunos ejemplos, la mayoría aciertan en la reconstrucción y genera menos imágenes borrosas. Además, en contraposición con la reconstrucción de la red *WAE-GAN*, es más difícil

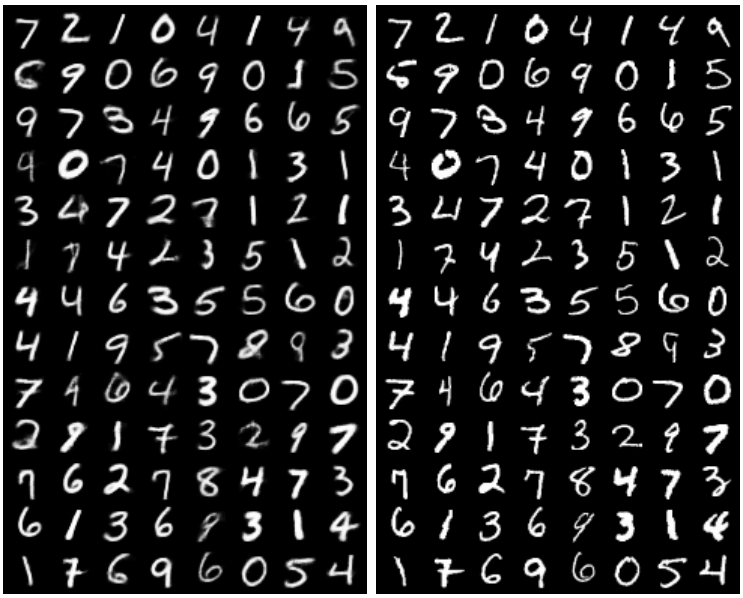

Figura 6.6: A la izquierda la reconstrucción de la red *WAE-MMD*, a la derecha los ejemplos que se intentan reconstruir de *MNIST*

observar elementos ilegibles en la reconstrucción realizada por esta segunda red.

Aún así, ambas redes tienen mayor margen de mejora, pues un hecho observable de ambas es claro: estas redes confunden números que presentan lazos, como son el número 8, el 6 y el 2, en ocasiones. De hacho, llevado al límite, hay ocasiones en las que también se genera confusión en el número 3, por los lazos, y entre los números 4 y 9.

Ahora, consideramos otra tarea distinta, que se centrará en generar nuevas muestras del conjunto, en vez de reconstruir los ejemplos fijos por la semilla aleatoria. Esta tarea servirá para poder comparar ambas redes entre sí, aunque no generen las mismas muestras, en un marco generativo, en vez de en el marco de la codificación y decodificación de los datos.

Comenzaremos presentando las imágenes generadas, a las 10, 50 y 100 épocas, por la *WAE-GAN*, que se pueden ver en la figura 6.7. La idea de presentar los resultados generados en distintas épocas permite ver la evolución de la generación de elementos mediante el entrenamiento realizado, de modo que se pueda observar si un enfoque distinto podría producir los mismos resultados

Ahora, presentaremos los resultados, de forma idéntica, de la red *WAE-MMD*, también tras realizar 10, 50 y 100 épocas de entrenamiento. Podemos ver los resultados en la figura 6.8.

Observando los resultados de esta segunda red nos damos cuenta de que para generar números, aunque mejora con las épocas, ya se consiguen resultados creíbles tras 10 épocas de entrenamiento de la red. Aunque tras 100 épocas se observan aún, algunos dígitos no reconocibles, tras 50 épocas, no se observa prácticamente una mejora, de modo que podría haberse realizado un entrenamiento con un menor número de iteraciones de modo que se satisficieran los objetivos de generación de elementos. Incluso si se hubiera necesitado un

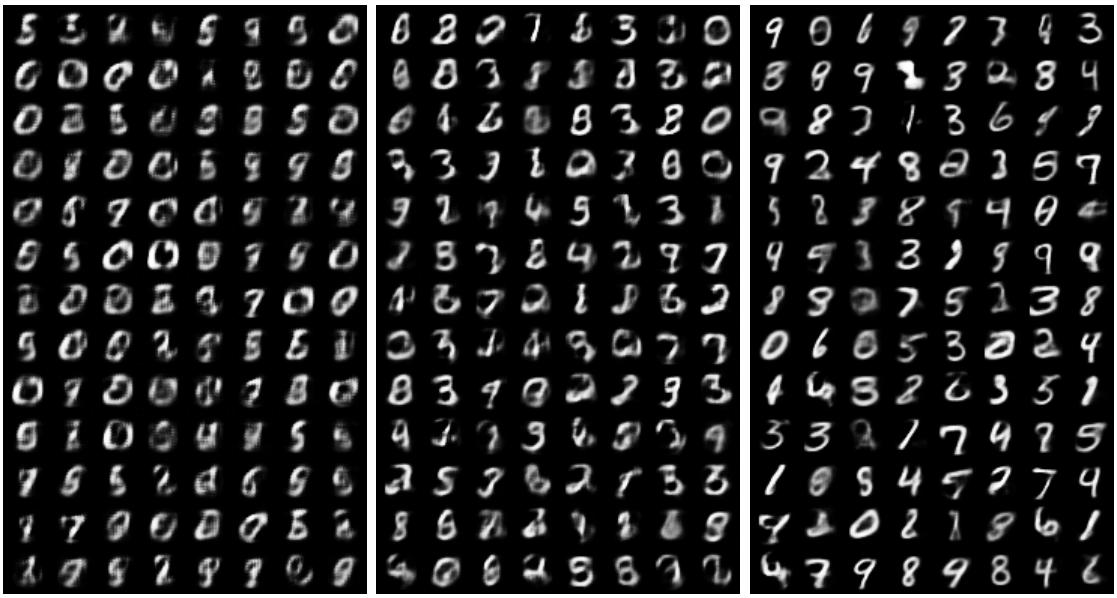

Figura 6.7: De izquierda a derecha, el entrenamiento de la red *WAE-GAN* en 10, 50 y 100 épocas, respectivamente

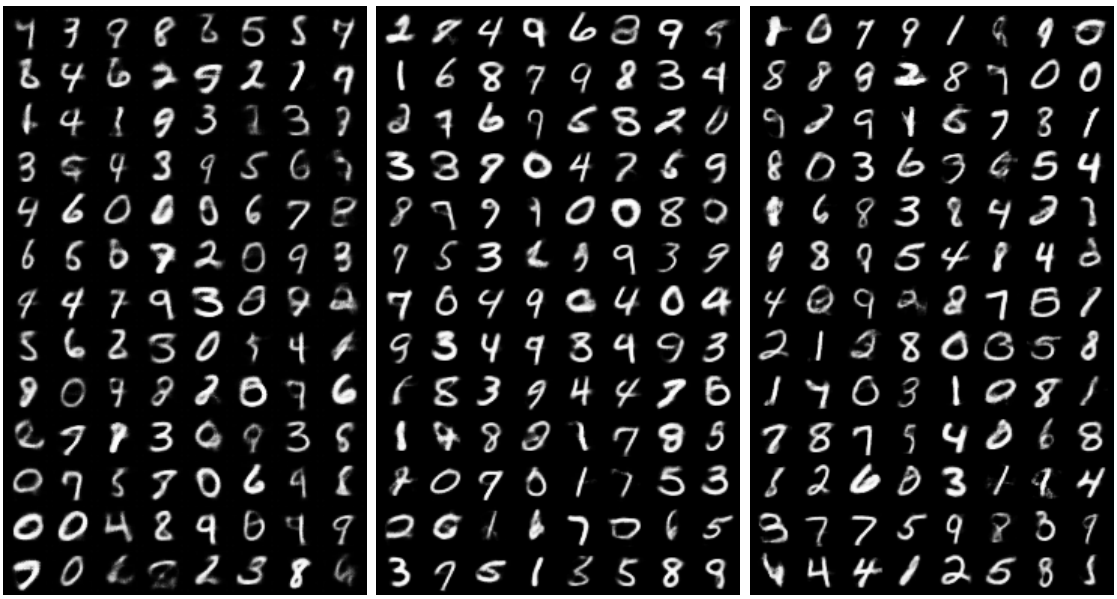

Figura 6.8: De izquierda a derecha, el entrenamiento de la red *WAE-MMD* en 10, 50 y 100 épocas, respectivamente

resultado aún más pulido que tras 50 épocas, habría sido suficiente un menor número de iteraciones.

Si comparamos los resultados finales que se han producido entre ambas redes, tras las 100 iteraciones de entrenamiento, se puede observar que la red *WAE-MMD* obtiene también mejores resultados, al igual que en el caso de la reconstrucción de elementos,

pues los resultados producidos osn más nítidos y reconocibles que en el caso de la red *WAE-GAN*.

Capítulo 7

Conclusiones

Entramos al último capítulo del trabajo, en el cual observaremos la consecución de los objetivos planteados, la valoración personal sobre el trabajo y las posibles líneas de trabajo futuro a partir de este.

Se recuerda que los objetivos planteados para el trabajo se pueden encontrar en la sección: 1.2. Con el objetivo de hacer un análisis objetivo del trabajo, discutimos ahora la consecución de los objetivos planteados en la sección mencionada.

El primero de los objetivos hace referencia a la introducción de conceptos para los lectores menos familiarizados con el funcionamiento de las redes neuronales. Este primer grupo constaba de cuatro subobjetivos que permiten comprobar si realmente se ha alcanzado el objetivo principal.

El primero de los subobjetivos hace referencia a los distintos tipos de aprendizaje, cuestión que se presenta en la sección 2.2. Pese a que se podía haber introducido el concepto de aprendizaje semisupervisado en la sección, para completarla, este tipo de aprendizaje es un paso intermedio entre los dos que se presentan: el supervisado y el no supervisado. Se concluye entonces que este primer subobjetivo se ha cumplido.

Siguiendo con el segundo subobjetivo del primer grupo, las medidas de error, funciones de pérdida y riesgo se presentan también en el capítulo 2, entrando en suficiente detalle, por ejemplo, en la divergencia entre probabilidades como forma de medir el error, de modo que se puede concluir que también se logra alcanzar este subobjetivo.

En el caso del tercer subobjetivo de este grupo, se explica en detalle el funcionamiento del *ML* y *DL*, junto con la estructura de una red neuronal corriente, que es lo que buscaba este subobjetivo. Concluimos que este objetivo también se ha alcanzado.

Para terminar con el primer grupo de objetivos, en las secciones 2.8 y 2.9 se explican los principales algoritmos de entrenamiento de redes neuronales. Aunque se podía haber entrado en mayor detalle a la hora de explicar elementos como el descenso de gradiente estocástico o el optimizador *ADAM*, no se ha considerado necesario, pues sólo se utilizan como herramientas, y su funcionamiento, aunque diferente, se basa en el algoritmo de descenso de gradiente.

Con este último objetivo cumplido, podemos decir que se ha alcanzado el marco del primer objetivo principal: la explicación de los conceptos básicos para entender la base

del trabajo.

Observemos ahora la consecución del segundo objetivo que se ha presentado: la introducción a la inferencia variacional. Igual que para la consecución del primer objetivo, observemos los subobjetivos para concluir sobre el cumplimiento del mismo. Aunque las secciones relacionadas con este objetivo (capítulo 4) pueden parecer escuetas, en ellas se encuentra la información necesaria para comprender los conceptos de cota inferior de la evidencia, así como el funcionamiento y objetivos de la inferencia variacional e inferencia variacional estocástica; así como el problema principal de la *VI*. Esto cubriría los tres subobjetivos, pese a que es cierto que con mayor investigación se podría haber ampliado la sección referente a los problemas de la inferencia variacional.

Concluimos entonces que el segundo objetivo se alcanza y, aunque tiene margen de mejora, es suficiente para entender la mayoría de los conceptos del trabajo que se relacionan con su capítulo.

Finalmente, nos centraremos en el tercer objetivo, el cual comprende la idea principal del trabajo y comprende los capítulos 3, 4.2,5 y 6.

Cada uno de estos capítulos cubre uno de los subobjetivos presentados como partes para la consecución de **OBJ-3**, y están explicados en suficiente detalle como para cubrir los conceptos principales del trabajo, pues se explican de forma ordenada y escalando hacia los conceptos más complejos desde los más básicos. Comenzando por los autocodificadores, se estructura el trabajo de modo que tras ellos se explica la inferencia variacional, los *VAE* y los autocodificadores de *Wasserstein*. Tras la explicación teórica, se presenta la arquitectura de las redes a implementar (*WAE-GAN* y *WAE-MMD*), y se realiza la implementación y comparación de resultados con los objetivos.

Como se podía observar en 6.4, el desempeño de la red *WAE-MMD* ha sido mejor que en el caso de la red *WAE-GAN*. La ventaja obtenida por la red *WAE-MMD* ha sido en ambas tareas y en igualdad de condiciones en cuanto a épocas de entrenamiento y parámetros para el entrenamiento elegidos.

Se concluye que se consiguen los objetivos, pues con la presentación de los distintos tipos de autocodificador se puede comprender la totalidad del trabajo y con la implementación se pueden estudiar los resultados obtenidos.

7.1. Valoración personal

En cuanto a la valoración personal sobre el desempeño del trabajo, considero que el trabajo es suficientemente ambicioso, y aunque se centra en explicar conceptos ya conocidos y estudiados, sirve de apoyo sin abarcar más de la cuenta. También me ha servido como ayuda para comprender el funcionamiento estadístico y matemático detrás de las redes neuronales, pues hasta el momento sólo conocía la parte informática de las redes neuronales.

Además, personalmente, me ha ayudado a conseguir poner unas metas claras y entender la necesidad de no abarcar un contenido excesivo sobre ellas, pues entonces el trabajo no sería claro ni sencillo.

Finalmente, he comprobado que las ideas detrás de los grandes modelos de inteligencia artificial se basan en conceptos de codificación-decodificación, además de haber ganado conocimiento sobre el funcionamiento de las redes autocodificadoras.

7.2. Líneas de trabajo futuro

Para terminar con el trabajo, se detallan aquí algunas de las posibles líneas de trabajo futuro, una vez terminado este trabajo.

La primera línea tiene que ver con mejoras sobre el propio trabajo. Es posible que con un mayor número de iteraciones se obtengan mejores resultados para ambas redes, pudiendo comparar los resultados obtenidos entre distintos números de épocas con diferentes parámetros. También podría ser necesario ahondar sobre los conceptos de inferencia variacional, así como sobre el funcionamiento de los optimizadores más utilizados, como lo es *ADAM*.

Como segunda línea tenemos la investigación sobre conceptos relacionados con el trabajo, pero más modernos. El estudio de arquitecturas *Transformer*, basadas en métodos de atención, o de los grandes modelos del lenguaje, es una manera de continuar, escalando la dificultad de los conceptos, dado que estas arquitecturas nacen de la necesidad de simplificar las arquitecturas complejas que ya se presentaban hace unos años. De esta forma se aumentaría el conocimiento sobre el funcionamiento de los modelos más actuales, aunque también necesitaría de la primera línea para poder obtener un conocimiento más completo sobre ellos.

Finalmente, la tercer línea de trabajo futuro posible se centraría en la aplicación de este tipo de agentes inteligentes a la robótica, con el fin de entender la visión computacional, el funcionamiento de los actuadores de un robot complejo y las matemáticas que residen detrás de ellos.

Parte III

Apéndices

Apéndice A

Teoremas utilizados

A.1. Demostración de resultados sobre la distancia de *Wasserstein*

Recordemos el resultado que queremos demostrar: sea $p(\mathbf{x})$ una distribución fija en $\mathcal{X}$, un espacio métrico compacto. Sean Z una variable aleatoria sobre $\mathcal{Z}$ y $g : \mathcal{Z} \times \mathbb{R}^d \to \mathcal{X}$ una función de variables z y θ que parametrizaremos respecto de θ fijo ($g_\theta(z)$). Si P_θ es la distribución de $g_\theta(Z)$, entonces:

1. Si g es continua en θ, lo será $W(p(\mathbf{x}), P_\theta)$.
2. Si g es localmente lipschitziana y satisface ciertas condiciones de regularidad, entonces $W(p(\mathbf{x}), P_\theta)$ es continua en todo punto y diferenciable en casi todo punto.
3. Lo anterior no se cumple para D_{JS} ni D_{KL}.

Hay que tener en cuenta que las condiciones de regularidad que se han de satisfacer en 2 se pueden ver en 5.1.

Veamos la demostración. Sean θ y θ' dos vectores de parámetros en $\mathbb{R}^d$. Primero buscaremos acotar la distancia $W(P_\theta, P_{\theta'})$, y a partir de esto terminaremos la demostración. El elemento principal de la prueba es el uso del emparejamiento γ de las distribuciones $g_\theta(Z)$ y $g_{\theta'}(Z)$, el cual pertenece a $\mathcal{P}(P_\theta, P_{\theta'})$, el conjunto de distribuciones conjuntas de marginales (P_θ y $P_{\theta'}$), respectivamente.

Siguiendo la definición de la distancia de *Wasserstein*, tenemos que:

$$W(P_\theta, P_{\theta'}) \leq \int_{\mathcal{X}\times\mathcal{X}} \|x - y\| \, d\gamma = \mathbb{E}_{x,y\sim\gamma} \left[\|x - y\|\right] = \mathbb{E}_z \left[\|g_\theta(z) - g_{\theta'}(z)\|\right].$$

Si g es continua en θ, entonces $g_\theta(z) \to g_{\theta'}(z)$ si $\theta \to \theta'$, luego $\|g_\theta - g_{\theta'}\| \to 0$ puntualmente (como funciones de z). Como $\mathcal{X}$ es un conjunto compacto, esta distancia ha de estar uniformemente acotado, luego $\|g_\theta(z) - g_{\theta'}(z)\| \leq M$, para M constante, y para cualesquiera θ y z. Por el teorema de la convergencia dominada:

$$W(P_\theta, P_{\theta'}) \leq \mathbb{E}_z \left[\|g_\theta(z) - g_{\theta'}(z)\|\right] \to 0,$$

siempre que $\theta \to \theta'$.

Esto nos permite escribir que:

$$|W(P_r, P_\theta) - W(P_r, P_{\theta'})| \leq W(P_\theta, P_{\theta'}) \to 0,$$

siguiendo las condiciones anteriores. Con esto, queda probada la continuidad de $W(P_r, P_\theta)$.

Ahora, sea g localmente lipschitziana. Entonces, para un par (θ, z) existen una constante $L(\theta, z)$ y un abierto U tales que $(\theta, z) \in U$. Además, para cada $(\theta', z') \in U$, se tiene que:

$$\|g_\theta(z) - g_{\theta'}(z')\| \leq L(\theta, z)(\|\theta - \theta'\| + \|z - z'\|).$$

Si tomamos esperanzas y $z = z'$:

$$\mathbb{E}_z \left[\|g_\theta(z) - g_{\theta'}(z)\|\right] \leq \|\theta - \theta'\| \, \mathbb{E}_z \left[L(\theta, z)\right],$$

siempre que $(\theta', z) \in U$. De este modo, podemos definir $U_\theta = \{\theta' : (\theta', z) \in U\}$, el cual es un conjunto abierto por serlo U.

Si además consideramos las condiciones de regularidad 5.1, entonces se puede definir $L(\theta) = \mathbb{E}_z \left[L(\theta, z)\right]$, lo que nos lleva a:

$$|W(P_r, P_\theta) - W(P_r, P_{\theta'})| \leq W(P_\theta, P_{\theta'}) \leq L(\theta) \, \|\theta - \theta'\|,$$

para cada $\theta \in U_\theta$.

Esto significa que $W(P_r, P_\theta)$ es localmente lipschitziana, lo que nos lleva a la continuidad en todo punto, y además a su diferenciabilidad en casi todo punto (teorema de *Radamacher*).

Para finalizar la prueba, veamos un contraejemplo para los casos de D_{KL} y D_{JS}. Sea $Z \sim \mathsf{U}[0, 1]$, donde $\mathsf{U}[0, 1]$ es una distribución uniforme unidimensional en el intervalo $[0, 1]$. Sea P_0 la distribución de $(0, Z) \in \mathbb{R}^2$, uniforme en una línea vertical que pasa por el origen. Ahora, sea $g_\theta(z) = (\theta, z)$, siendo $\theta \in \mathbb{R}$ un parámetro. Entonces, bajo estas suposiciones, se tiene que:

- $D_{JS}(P_0, P_\theta) = \begin{cases} \log(2) & \text{si} \quad \theta \neq 0, \\ 0 & \text{si} \quad \theta = 0. \end{cases}$

- $D_{KL}(P_0, P_\theta) = \begin{cases} +\infty & \text{si} \quad \theta \neq 0, \\ 0 & \text{si} \quad \theta = 0. \end{cases}$

Lo cual demuestra que el teorema no se cumple para ni para la divergencia de *Jensen-Shannon*, ni para la divergencia de *Kullback-Leibler* (las funciones no son continuas).

A.2. Dualidad de *Kantorovich-Rubinstein*

Sean p, q densidades de probabilidad, $\mathcal{P}(p,q)$ el conjunto de probabilidades con marginales p y q y h una función 1-Lipschitziana en $(\mathcal{X}, \|\cdot\|)$. Se tiene que:

$$W(p,q) = \inf_{P\in\mathcal{P}(p,q)} \mathbb{E}\left[\|x-y\|\right] = \sup_{\|h\|_L\leq 1} \left[\mathbb{E}_p(h(x)) - \mathbb{E}_q(h(y))\right]. \tag{A.1}$$

Veámoslo. Seguiremos la demostración de [38].

Comenzaremos utilizando los multiplicadores de *Lagrange* para dos funciones medibles y acotadas $f, g : \mathcal{X} \to \mathbb{R}$:

$$L(P,f,g) = \int_{\mathcal{X}\times\mathcal{X}} \|x-y\| P(x,y)dydx + \int_{\mathcal{X}} \left(p(x) - \int_{\mathcal{X}} P(x,y)dy\right) f(x)dx + \int_{\mathcal{X}} \left(q(y) - \int_{\mathcal{X}} P(x,y)dx\right) g(y)dy.$$

Juntando términos de manera adecuada, se puede reescribir la Lagrangiana:

$$L(P,f,g) = \mathbb{E}_x\left[f(x)\right] + \mathbb{E}_y\left[g(y)\right] + \int_{\mathcal{X}\times\mathcal{X}} \left(\|x-y\| - p(x) - q(y)\right) P(x,y)dydx.$$

A partir de esto, aplicando la dualidad fuerte:

$$W(p,q) = \inf_P \sup_{f,g} L(P,f,g) = \sup_{f,g} \inf_P L(P,f,g).$$

Si $\|x-y\| < f(x)+g(y)$, para algunos $x, y \in \mathcal{X}$, entonces podemos concentrar la masa de P en (x,y), t por lo tanto $L(P,f,g) \to -\infty$. Por lo tanto, ha de ser que:

$$f(x) + g(y) \leq \|x-y\|.$$

Con ello conseguimos que, minimizando sobre P, lo mejor será que $P = 0$:

$$\sup_{f,g} \inf_P L(P,f,g) = \sup_{f,g,f(x)+g(y)\leq\|x-y\|} \left[\mathbb{E}_x(f(x)) + \mathbb{E}_y(g(y))\right] = W(p,q)$$

Con lo anterior, podemos ver que, optimizando sobre la clase de funciones Lipschitzianas de constante 1, obtenemos una cota inferior para la distancia de *Wasserstein*. Esto nos permite escribir:

$$\mathbb{E}_x(h(x)) + \mathbb{E}_y(h(y)) = \int_{\mathcal{X}\times\mathcal{X}} \left(h(x) - h(y)\right) P(x,y)dxdy \leq \int_{\mathcal{X}\times\mathcal{X}} \|x-y\| P(x,y)dxdy \leq W(p,q).$$

Lo que nos permite obtener la desigualdad:

$$\sup_{\|h\|_L\leq 1} \left[\mathbb{E}_x(h(x)) - \mathbb{E}_y(h(y))\right] \leq W(p,q),$$

Así que será suficiente con ver que esta desigualdad se alcanza.

Consideramos la función definida por:

$$x \mapsto \inf_u \left[\|x-u\| - g(u)\right],$$

y que llamaremos κ. Como g es acotada, el ínfimo es finito y la función está bien definida. Además, κ es 1-Lipschitziana:

$$\kappa(x) \leq \|x-u\| - g(y) \leq \|x-y\| + \|y-u\| - g(u),$$

siendo $x, y \in \mathcal{X}$, $u \in \mathcal{X}$ arbitrario. Como u es arbitrario:

$$\kappa(x) \leq \|x-y\| + \inf\left[\|x-u\| - g(u)\right] g(y) = \|x-y\| + \kappa(y),$$

lo que equivale a:

$$\kappa(x) - \kappa(y) \leq \|x-y\|.$$

Intercambiando los papeles de x e y se obtiene que:

$$\kappa(y) - \kappa(x) \leq \|x-y\|,$$

lo que prueba que κ es una función 1-Lipschitziana.

Ahora, para cada par f, g que cumpla $f(x) + g(y) \leq \|x-y\|$, se tendrá que:

$$f(x) \leq \kappa(x) \leq \|x-x\| - g(x) = -g(x),$$

de lo que se deduce:

$$\mathbb{E}_x(f(x)) + \mathbb{E}_y(g(y)) \leq \mathbb{E}_x(\kappa(x)) - \mathbb{E}_y(\kappa(y)).$$

Esto nos permite concluir que:

$$\begin{aligned} W(p,q) &= \sup_{f,g,f(x)+g(y)\leq\|x-y\|} \left[\mathbb{E}_x(f(x)) + \mathbb{E}_y(g(y))\right] \\ &\leq \sup_{\|h\|_L\leq 1} \left[\mathbb{E}_x(h(x)) - \mathbb{E}_y(h(y))\right] \leq W(p,q). \end{aligned}$$

Y con ello termina la demostración de la dualidad.

Bibliografía

[1] Microsoft (Marzo 2023). *Microsoft 365 Copilot*. Última vez visitado: 2023, 20 de Abril. URL: https://blogs.microsoft.com/blog/2023/03/16/introducing-microsoft-365-copilot-your-copilot-for-work/.

[2] Cem Akkus et al. «Multimodal Deep Learning». En: (2023). arXiv: 2301.04856 [cs.CL].

[3] Martin Arjovsky, Soumith Chintala y Léon Bottou. «Wasserstein GAN». En: (2017). arXiv: 1701.07875 [stat.ML].

[4] Rowel Atienza. *Advanced Deep Learning with Keras*. https://www.packtpub.com/product/advanced-deep-learning-with-keras/9781788629416. Packt Publishing, Octubre 2018.

[5] Dzmitry Bahdanau, Kyunghyun Cho y Yoshua Bengio. «Neural Machine Translation by Jointly Learning to Align and Translate». En: (2016). arXiv: 1409.0473 [cs.CL].

[6] Tadas Baltrušaitis, Chaitanya Ahuja y Louis-Philippe Morency. «Multimodal Machine Learning: A Survey and Taxonomy». En: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 41.2 (2019), págs. 423-443. DOI: 10.1109/TPAMI.2018.2798607.

[7] Álvaro Baños Izquierdo et al. «Modelos generativos profundos: autocodificadores variacionales». En: (2022). URL: https://uvadoc.uva.es/handle/10324/57976.

[8] P. Billingsley. *Probability and Measure (third ed.)*. John Wiley y Sons, 1995.

[9] L. Bottou et al. «Comparison of classifier methods: a case study in handwritten digit recognition». En: 2 (1994), 77-82 vol.2. DOI: 10.1109/ICPR.1994.576879.

[10] Olivier Bousquet et al. «From optimal transport to generative modeling: the VEGAN cookbook». En: (2017). arXiv: 1705.07642 [stat.ML].

[11] Sebastién Bubeck. *Convex optimization: Algorithms and Complexity*. Foundations y trends in Machine Learning, 2015.

[12] Kyunghyun Cho et al. «Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation». En: (2014). arXiv: 1406.1078 [cs.CL].

[13] T.M. Cover y J.A. Thomas. *Elements of Information Theory*. https://books.google.es/books?id=VWq5GG6ycxMC. Wiley, 2012. ISBN: 9781118585771.

[14] Nello Cristianini y John Shawe-Taylor. *An Introduction to Support Vector Machines and Other Kernel-based Learning Methods*. Cambridge University Press, 2000. DOI: 10.1017/CBO9780511801389.

[15] Jacob Devlin et al. «BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding». En: (2019). arXiv: 1810.04805 [cs.CL].

[16] Kevin Swersky Geoffrey Hinton Nitish Srivastava. *Lecture slides from the 2012 Coursera course: Neural Networks for Machine Learning*. Última vez visitado: 2023, 24 de Abril. URL: https://www.cs.toronto.edu/~hinton/coursera_slides.html.

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

[18] Ian J. Goodfellow et al. «Generative Adversarial Networks». En: (2014). arXiv: 1406.2661 [stat.ML].

[19] Matt Hoffman et al. «Stochastic Variational Inference». En: (2013). arXiv: 1206.7051 [stat.ML].

[20] Yacine Jernite, Samuel R. Bowman y David Sontag. «Discourse-Based Objectives for Fast Unsupervised Sentence Representation Learning». En: (2017). arXiv: 1705.00557 [cs.CL].

[21] Oleksii Kuchaiev y Boris Ginsburg. «Factorization tricks for LSTM networks». En: (2018). arXiv: 1703.10722 [cs.CL].

[22] Y. LeCun y C. Cortes. *The MNIST Database*. Última vez visitado: 2023, 27 de Mayo. URL: http://yann.lecun.com/exdb/mnist/.

[23] Y. Lecun et al. «Gradient-based learning applied to document recognition». En: *Proceedings of the IEEE* 86.11 (1998), págs. 2278-2324. DOI: 10.1109/5.726791.

[24] Alireza Makhzani et al. «Adversarial Autoencoders». En: (2015). DOI: 10.48550/ARXIV.1511.05644. URL: https://arxiv.org/abs/1511.05644.

[25] Bryan McCann et al. «Learned in Translation: Contextualized Word Vectors». En: (2018). arXiv: 1708.00107 [cs.CL].

[26] OpenAI. «GPT-4 Technical Report». En: (2023). arXiv: 2303.08774 [cs.CL].

[27] OpenAI. «Improving Language Understanding by Generative Pre-Training». En: (2018). URL: https://cdn.openai.com/research-covers/language-unsupervised/language_understanding_paper.pdf.

[28] OpenAI. *Página web de ChatGPT*. Última vez visitado: 2023, 18 de Agosto. URL: https://openai.com/blog/chatgpt.

[29] OpenAI. *Página web de DALL-E 2*. Última vez visitado: 2023, 18 de Agosto. URL: https://openai.com/product/dall-e-2.

[30] Daniel W. Otter, Julian R. Medina y Jugal K. Kalita. «A Survey of the Usages of Deep Learning in Natural Language Processing». En: (2019). arXiv: 1807.10854 [cs.CL].

[31] Matthew E. Peters et al. «Deep contextualized word representations». En: (2018). arXiv: 1802.05365 [cs.CL].

[32] Alec Radford et al. «Learning Transferable Visual Models From Natural Language Supervision». En: (2021). arXiv: 2103.00020 [cs.CV].

[33] Sebastian Ruder. «An overview of gradient descent optimization algorithms». En: (2017). arXiv: 1609.04747 [cs.LG].

[34] Shai Shalev-Shwartz y Shai Ben-David. «Dimensionality Reduction». En: *Understanding Machine Learning: From Theory to Algorithms*. Cambridge University Press, 2014.

[35] Weaver W Shannon CE. *A mathematical theory of communication*. https://pure.mpg.de/rest/items/item_2383164/component/file_2383163/content. University of Illinois Press, Champaign, IL, 1963.

[36] Noam Shazeer et al. «Outrageously Large Neural Networks: The Sparsely-Gated Mixture-of-Experts Layer». En: (2017). arXiv: 1701.06538 [cs.LG].

[37] Ilya Sutskever, Oriol Vinyals y Quoc V. Le. «Sequence to Sequence Learning with Neural Networks». En: (2014). arXiv: 1409.3215 [cs.CL].

[38] John Thickstun. *Kantorovich-Rubinstein Duality*. Última vez visitado: 2023, 17 de Agosto. URL: https://courses.cs.washington.edu/courses/cse599i/20au/resources/L12_duality.pdf.

[39] Ilya Tolstikhin et al. «Wasserstein Auto-Encoders». En: (2017). DOI: 10.48550/ARXIV.1711.01558. URL: https://arxiv.org/abs/1711.01558.

[40] *Variational Methods for Machine Learning with Applications to Deep Networks*. Springer International Publishing, 2021.

[41] Ashish Vaswani et al. «Attention Is All You Need». En: (2017). arXiv: 1706.03762 [cs.CL].