UNIVERSIDAD
DE GRANADA

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

Doble Grado en Ingeniería Informática y Matemáticas

TRABAJO DE FIN DE GRADO

Fundamentos de Deep Learning y desarrollo de un modelo de Aprendizaje Federado para la diagnosis de COVID-19 a partir de radiografías de tórax

Presentado por:
Francisco Miguel Castro Macías

Tutor:
Francisco Herrera Triguero
Departamento de Ciencias de la Computación e Inteligencia Artificial

Siham Tabik
Departamento de Ciencias de la Computación e Inteligencia Artificial

Curso académico 2020-2021

Fundamentos de Deep Learning y desarrollo de un modelo de Aprendizaje Federado para la diagnosis de COVID-19 a partir de radiografías de tórax

Francisco Miguel Castro Macías

Francisco Miguel Castro Macías *Fundamentos de Deep Learning y desarrollo de un modelo de Aprendizaje Federado para la diagnosis de COVID-19 a partir de radiografías de tórax*.
Trabajo de fin de Grado. Curso académico 2020-2021.

Responsable de tutorización

Francisco Herrera Triguero
Departamento de Ciencias de la Computación e Inteligencia Artificial

Siham Tabik
Departamento de Ciencias de la Computación e Inteligencia Artificial

Doble Grado en Ingeniería Informática y Matemáticas

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

Universidad de Granada

Declaración de originalidad

D./Dña. Francisco Miguel Castro Macías

Declaro explícitamente que el trabajo presentado como Trabajo de Fin de Grado (TFG), correspondiente al curso académico 2020-2021, es original, entendida esta, en el sentido de que no ha utilizado para la elaboración del trabajo fuentes sin citarlas debidamente.

En Granada a 8 de julio de 2021

Fdo: Francisco Miguel Castro Macías

A mis padres, por creer siempre en mí.

Índice general

Resumen

En los dos últimos años el mundo ha sido testigo de la aparición de una de las enfermedades más contagiosas del siglo XXI. La llamada COVID-19 ha ocasionado una pandemia que ha llegado a todos los rincones del planeta. Uno de los aspectos cruciales para poder tratar con éxito esta enfermedad es que el paciente sea diagnosticado a tiempo. Debido a la alta tasa de contagio, un gran número de pacientes han de ser atendidos en muy poco tiempo. Por ello es fundamental desarrollar técnicas de diagnóstico rápidas y precisas. Uno de estos métodos es el diagnóstico mediante radiografías de tórax, que se presenta como el más efectivo en cuanto al tiempo que requiere y al coste que conlleva.

En la actualidad, los modelos de aprendizaje profundo constituyen el estado del arte en todas las tareas de visión por computador. Las redes neuronales se han aplicado con éxito a multitud de problemas, entre los que se encuentran clasificación de imágenes y segmentación. El tipo concreto de red neuronal convolucional ha obtenido resultados superiores a las del experto humano en numerosas situaciones.

Una de las aplicaciones más exitosas de estas redes es el diagnóstico médico. Es indiscutible que un modelo robusto y preciso puede servir como un método de triaje y de apoyo a la toma de decisiones de los expertos en medicina. Sabemos, sin embargo, que para que estos modelos puedan obtener resultados competitivos debemos contar con una gran cantidad de imágenes en nuestro conjunto de datos. Esto supone un problema debido a las grandes limitaciones que tenemos para construir conjuntos de datos grandes con información procedente de hospitales. Por motivos legales, los centros médicos no pueden compartir los datos de sus pacientes libremente. Al tratar con ellos es necesario seguir un riguroso proceso para mantener la privacidad y la seguridad de la información. Ante esta necesidad surge el paradigma del aprendizaje federado. En contraposición al aprendizaje clásico, la totalidad de los datos no está disponible para el sujeto que lleva a cabo el entrenamiento, sino que estos están repartidos en distintos nodos.

En este trabajo diseñamos una nueva metodología para predecir la enfermedad COVID-19 a partir de radiografías de tórax que residen en diversos hospitales. En la vertiente más práctica, desarrollamos la librería SDNETLearning que permite entrenar modelos e inferir el diagnóstico de una radiografía en plataformas distribuidas de hospitales a nivel nacional.

En la vertiente teórica del trabajo realizamos un estudio exhaustivo de los fundamentos del problema del aprendizaje. De esta forma se estudiará el paradigma de aprendizaje estadístico clásico y el nuevo paradigma de aprendizaje federado y los retos que plantea. Además, formalizaremos la teoría que respalda a los métodos de aprendizaje profundo que componen el estado del arte para problemas de clasificación.

Los resultados que obtenemos como fruto de este trabajo son muy prometedores y establecen el camino a seguir para obtener una solución general a este problema.

Palabras clave: probabilidad, teoría de la información, optimización, aprendizaje automático, teoría de la generalización, clasificación, aprendizaje profundo, redes neuronales, aprendizaje federado.

Extended abstract

In the last years, the world has witnessed the appearance of COVID-19, one of the most infectious diseases in the 21st century. As a consequence, the planet is in a pandemic situation whose end is not clear. An early diagnosis of COVID-19 is the most critical step to treat the infection so a large number of patients will need to be assessed in short time intervals. The COVID-19 is diagnosed using RT-PCR testing. The needed material and equipment are not easily accessible and it takes from 24 to 48 hours. This is why an alternative and complementary method is needed. The use of chest X-Ray (CXR) images is the most time/cost effective tool for assisting clinicians in making decisions.

Deep learning models are now the state of the art in all computer vision tasks. Neural networks have been successfully applied to a wide variety of problems, including image classification and segmentation. Convolutional neural networks have outperformed human experts in many situations. One of the most successful applications of these networks is medical diagnosis. It is undeniable that a robust and accurate model can serve as a method of triage and decision support for medical experts. We know, however, that in order for these models to obtain competitive results we must have a large number of images in our dataset.

This is a problem which emerges due to the severe limitations we have in building large datasets with information from hospitals. For legal reasons, medical centres cannot share their patients' data freely. When dealing with them, it is necessary to follow a rigorous process to maintain the privacy and security of the information. In response to this need, the federated learning paradigm emerges. In contrast to classical learning, the entire data is not available for the training subject, but is distributed among different nodes.

In this work, we design a new methodology for predicting COVID-19 using CXR images in a distributed setting under privacy and security constraints. On the more practical side, we developed the SDNETLearning library that allows to train models and infer the diagnosis of a CXR on distributed platforms from nationwide hospitals.

In the theoretical part of the work, we carry out an exhaustive study of the foundations of the learning problem. This way, we will study the classic statistical learning paradigm and the new federated learning paradigm and the challenges it poses. In addition, we will formalise the theory behind the state-of-the-art deep learning methods for problems of classification.

The results we obtain are very promising and set the way forward for a general solution to this problem.

Our approach

To face the problem we need to understand the theory of machine learning and deep learning. This is the goal of the theoretical approach of this work, whose basis is formed by the probability theory, information theory and convex optimization. Once we have learned about them, we can move forward with the of study the generalization theory and its consequences in machine learning, as well as the most important deep learning models for our work: feedforward networks, convolutional networks and generative adversarial networks.

In the final part of the theoretical framework, we introduce the concept of federated learning and the federated optimization problem that implies. We give a brief guideline of the current challenges in this field of learning.

The second part of this work is a much more practical one. We describe the COVIDGR-FL dataset, pointing its most important features. Also, we show the insights about the Python library SDNETLearning, which implements the SDNET methodology for a general binary classification problem in a federated scenario. In the last part, we analyze and compare the results of the experiments that have been conducted while this software.

Mathematical background

The mathematical framework for the aforementioned approach rests on three pillars. Each one is contained in a different chapter. The first one is probability. It is closely related to the measure theory, so we first review the concepts of measure and integral. We assume that the reader knows the construction of the abstract integral of a measurable function f with respect to a measurement μ, so that the expression

$$\int f \, d\mu$$

is well defined. All we need to do is to point some useful results about it. One of these allows us to write the equality

$$\int_{\Omega} (f \circ X) \; d\mu = \int_{\Omega'} f \, d(\mu \circ X^{-1}).$$

As a consequence, we can express the expectation of a random variable with respect to its probability distribution. Therefore, we review the concepts of random variable, probability distribution, conditional probability, and the definition and calculus of expectation.

The second pillar is information theory. This chapter intends to formulate the concepts of *entropy* and *divergence*, and to prove the relations between them. The most important result is the *information inequality* ([16]). It was first proved by Solomon Kullback and Richard Leibler and it allows us to prove that *Kullback-Leibler divergence* is indeed a divergence. It will be used to prove that generative adversarial networks behave as expected in an ideal setting.

Convex optimization is the name of the last pillar. We begin by introducing the problem of convex optimization. Afterwards, we prove the Jensen's inequality, an important result used to prove the information inequality. To finish we justify the *gradient descent* rule:

$$x_{t+1} = x_t - \nabla f(x_t) \tag{1}$$

To do so, we analyze the convergence of the algorithm that (1) defines under the assumptions that f is convex and differentiable and ∇f is Lipschitz continuous.

Learning theory and models

When we talk about learning theory in this work we refer to three fields: machine learning, deep learning and federated learning. Each of these fields is studied in a separate chapter and it provides the key elements to understand the problem of COVID-19 prediction (in both classical and federated scenarios) and SDNET methodology.

We follow Vladimir Vapnik's original ideas ([33]) to introduce the learning problem and the generalization theory. One of the most important results is the *Vapnik-Chervonenkis inequality*. It is based on the *Vapnik-Chervonenkis dimension*, an integer that enables us to measure the size of the hypothesis set defined by a model.

We also point important consequences that are present in every machine learning problem. One of these is named *approximation-generalization tradeoff* and is closely related with the bias-variance decomposition of the mean squared error.

Overfitting is another phenomenon that can be explained using the Vapnik-Chervonenkis inequality. To assess this problem it is common to modify the loss function by adding a penalty term, all of which is a form of *regularization*.

To understand the meaning of deep learning, we formally formulate the class of functions implemented by a *feedforward neuronal network*, as well as its training and inference process: *forward propagation* and *backward propagation*.

Discrete convolution operation leads us to define *convolutional networks (CNN)* as an extension of these networks. We can express the convolution between two tensors as a product of two matrixes. The convolution layer is an important component in every deep learning model, and is usually followed by a process of *pooling* and the *activation function*. The output of a CNN (or a neural network in general) can be used to reproduce the input classifying each pixel. This is known as *segmentation*.

Generative adversarial networks (GANs) deal with the problem of generative modeling by trying to solve a minimax problem between a generator and a discriminator. Following the original paper ([10]), we study the behavior of these networks in an idealised scenario.

FUCITNET ([25]) is a model for classification problems inspired by GANs method, in which we have a classifier and a generator for each class of the problem. This classifier focuses on minimizing the cross-entropy loss of each generator of belonging to its class. Each generator G will try to minimize the sum of a similarity term and the classification loss of $G(x)$ to its particular class. By doing this, each generator learns class-inherent transformations that bring a sample into the class domain.

COVID-SDNET ([32]) is a new methodology for predicting COVID-19 disease from CXR images. We denote the labels of a binary classification problem as $\{P, N\}$. It is composed of three stages:

1. The first one removes unnecessary information by cropping the image using a segmentation mask.

2. In the second one, for each sample x, the generators of FUCITNET (G_P, G_N) are used to create two new samples: $G_P(x), G_N(x)$. If the original sample belongs to class C, the transformation can belong to the classes CTN and CTP. In this way, we obtain four new classes: $\{PTP, PTN, NTP, NTN\}$.

3. In the last stage a CNN classification model is trained to predict the new classes. To predict the label $\{P, N\}$ for each original sample, SDNET proposes a new inference process.

As we can see, the first two are associated to the pre-processing to produce quality data (smart data stages), while the last corresponds to the learning and inference process.

To conclude this section, we introduce the problem of federated learning. In the classical scenario, the whole data is available to the subject in charge of training the model. In the federated scenario, the data is spread across several nodes, and each one of them cannot access any data other than its own.

To train a model in this new setting a federated optimization algorithm is proposed. Every node trains the same model using by its own data. Afterwards, the weights are fused using an aggregation operator. Two of the most popular operators are called Federated Averaging (which simply averages the parameters of each model) and Weighted Federated Averaging (which averages the parameters weighting each one according to some criteria).

SDNETLearning and experimentation

SDNETLearning is a new Python library built on top of Tensorflow, Pytorch and Sherpa.ai-FL ([26]). It aims to provide the necessary tools to train the FUCITNET model, as well as the SDNET pipeline, in both classical and federated scenarios. To accomplish this, we have developed the following classes:

- CIT, to encapsulate the training and inference processes of FUCITNET. It has methods to train and evaluate the generators and the classifier of the model in an arbitrary classification problem
- SDNET and FederatedSDNET, to encapsulate the training and inference processes of SDNET in its centralized and federated version. Using FederatedSDNET we can train FUCITNET in a federated scenario as well. These classes also provide methods to read the data from a predefined format.

Using this library, we have conducted several experiments to evaluate the components of SDNET in various partitions of COVIDGR-FL. Centralized experiments show similar results to [32], although FUCITNET's classifier shows superior performance.

Federated experimental setup is made by partitioning the original data according to different features of the images in COVIDGR-FL. These experiments simulate a much more realistic scenario and, as we expected, the performance of the model decreases. However, the results are still promising given the short amount of data we are working with.

We have also observed interesting phenomena in the convergence of the federated model when changing the aggregation operator. The model always converges to a state of equilibrium, but this convergence is highly dependent on the partition and the operator.

In the future, the goal is to obtain a sufficiently large database to conduct experiments in which every node is well represented. These results we have obtained leave open questions about the behavior of the federated models. To answer them it is required more empirical and theoretical study.

Keywords: probability, information theory, optimization, machine learning, generalization theory, clasification, deep learning, neural networks, federated learning.

Parte I.

Introducción y objetivos

1. Introducción

El primer capítulo de este trabajo pretende ofrecer una visión general del mismo. Para ello vamos a contextualizarlo dentro de las ramas de la ciencia con las que está relacionado. A continuación, describiremos el problema que aborda y la estructura que presenta.

1.1. Contextualización

En los dos últimos años el mundo ha sido testigo de la aparición de una de las enfermedades más contagiosas del siglo XXI. Esta enfermedad lleva el nombre de COVID-19 y ha ocasionado una pandemia que ha llegado a todos los rincones del planeta.

Sin duda alguna, este acontecimiento supone un punto de inflexión para nuestra comunidad que nos ha llevado a replantear los cimientos sobre los que estamos construyendo nuestra forma de vida. Además, aunque la tendencia aún no es clara, la aparición de nuevas variantes presenta la posibilidad de que tengamos que aprender a convivir con la enfermedad.

Uno de los aspectos cruciales para poder tratar con éxito la COVID-19 es que el paciente sea diagnosticado a tiempo. Debido a la alta tasa de contagio, un gran número de pacientes han de ser atendidos en muy poco tiempo. Por ello es fundamental desarrollar técnicas de diagnóstico rápidas y efectivas ([29]).

En la actualidad, la forma más habitual de establecer tal diagnóstico es el test RT-PCR ([6]). Sin embargo, debido a los materiales que requiere y el tiempo que conlleva, existe la necesidad de explorar nuevos métodos.

Uno de estos métodos es el diagnóstico mediante radiografías de tórax. Este se presenta como el más efectivo en cuanto al tiempo que requiere y al coste que conlleva.

El problema de determinar si una imagen médica presenta o no una enfermedad ha sido ampliamente estudiado en el ámbito del aprendizaje automático y numerosas variantes de técnicas de aprendizaje profundo han sido aplicadas con éxito a problemas de clasificación parecidos.

Este enfoque ha sido aplicado recientemente en numerosos trabajos de investigación. De hecho, el número de artículos científicos centrados en diagnosticar esta enfermedad utilizando técnicas de aprendizaje se ha disparado en el último año[1,2].

Las bases teóricas de las redes neuronales se asentaron hace varias décadas ([18]). Sin embargo, su utilización era prácticamente inviable debido al enorme coste computacional que acarreaba. En los últimos años, el aprendizaje profundo, o *Deep Learning* en inglés, ha experimentado un resurgimiento debido a la creciente capacidad de cálculo y de almacenamiento disponible. Esto se debe al desarrollo de dispositivos GPU y CPU mucho más potentes, junto a técnicas dedicadas a este campo ([24]).

El paradigma de aprendizaje automático en el que nos estamos moviendo asume que el proceso, técnica o algoritmo de aprendizaje tiene todos los datos del conjunto de entrenamiento a su disposición. Sin embargo, este no tiene por qué ser el caso que nos ocupa. Es

[1] https://covid19primer.com/dashboard
[2] https://covid19primer.com/categories/8/papers

muy normal que distintos hospitales alberguen información sobre pacientes que, o bien no pueden compartir por motivos legales, o bien requiere un proceso largo y tedioso para ser compartida.

Es por ello que, para permitir la colaboración de tales clientes, debemos tratar los datos con un gran nivel de seguridad y privacidad. Surge así el paradigma del aprendizaje federado ([15]), que se ha hecho muy popular en los últimos años.

1.2. Descripción del problema

Este trabajo se enmarca en el contexto descrito en la sección anterior. Buscamos, por una parte, desarrollar un modelo robusto para predecir el COVID-19 usando técnicas de aprendizaje profundo. Por otra parte, pretendemos que ese modelo se adapte con buenos resultados al escenario federado.

Los modelos que perseguimos estudiar e implementar son FUCITNET ([25]) y SDNET ([32]). Ellos son el eje central del trabajo y su tratamiento nos puede servir para dividirlo en dos bloques siguiendo la clásica fragmentación entre teoría y práctica.

El primer bloque, que es fundamentalmente teórico, persigue que el lector comprenda las bases en las que se asientan:

- El paradigma clásico de aprendizaje automático.
- El nuevo paradigma de aprendizaje federado y los retos que plantea.
- La formalización y formulación de las técnicas de aprendizaje profundo. Dentro de estas técnicas estudiamos FUCITNET y SDNET.

En el segundo bloque, que es eminentemente práctico, implementamos los modelos elegidos y realizamos experimentos para comprobar la validez de los mismos. Esta segunda parte se realizará a su vez en dos fases:

1. La primera fase estudia el problema concreto de la diagnosis de COVID-19 en el escenario clásico. Se pretende comprender e imitar los resultados de los trabajos originales ([25], [32]), desarrollando el software necesario.
2. En la segunda fase, extenderemos el problema al escenario federado. Nos moveremos en un escenario inexplorado hasta el momento, intentando comprender los resultados que obtengamos.

1.3. Estructura del trabajo

La división que hemos presentado anteriormente nos da una idea de la estructura que va a seguir este trabajo. Este, como se puede observar en el índice, se divide en partes, las cuales a su vez se componen de capítulos. Esta organización respeta las directrices de los trabajos de la titulación:

1. La primera parte está compuesta por dos capítulos. El primero es el capítulo en que nos encontramos. En el segundo se presentan los objetivos planteados al inicio del trabajo.

2. La segunda parte trata los conceptos propios de la titulación de Matemáticas que subyacen a todo el trabajo. Son esenciales para comprender todos los razonamientos que les siguen. Está compuesta por los capítulos 3, 4 y 5.

   En el capítulo 3 realizamos un breve repaso a conceptos y resultados fundamentales sobre probabilidad y estadística. En el cuarto capítulo estudiamos la teoría de la información, ofreciendo resultados importantes sobre entropía y divergencia. En el quinto se estudia una pequeña parte del problema de optimización de funciones convexas.

3. La tercera parte versa sobre aquellos conceptos de ciencias de la computación que perseguimos comprender. Se compone de los capítulos 6, 7 y 8.

   En el capítulo 6 hablamos de aprendizaje automático. Describimos el problema clásico del aprendizaje, analizamos uno de los resultados más importantes dentro de la teoría de la generalización y sus consecuencias en el aprendizaje.

   El séptimo capítulo es el más extenso y está dedicado al aprendizaje profundo. Definimos formalmente la estructura de red neuronal y sus procesos de entrenamiento; para después estudiar las redes neuronales convolucionales y las redes generativas antagónicas. Terminamos el capítulo formulando los modelos FUCITNET y SDNET.

   En el capítulo octavo presentamos finalmente el aprendizaje federado. Analizamos su formulación, las técnicas de optimización que requiere emplear y los retos que conllevan.

4. La cuarta parte del trabajo es la más práctica. Se compone de los capítulos 9 y 10.

   En el capítulo 9 comparamos el método de diagnóstico mediante radiografías con otras técnicas. También describimos las métricas y el conjunto de datos que vamos a usar.

   El capítulo 10 se corresponde con la experimentación. En él describimos el software que hemos desarrollado para llevar a cabo los experimentos. Terminamos analizando los resultados que hemos obtenido en los mismos.

5. La última parte se compone sólo del capítulo 11. En él recogemos todas las conclusiones que hemos obtenido y el trabajo que en un futuro podría llevarse a cabo.

1.4. Bibliografía fundamental

El conjunto de escritos utilizados para presentar este texto es muy variado. Señalamos a continuación aquellos que han sido más importantes a la hora de llevarlo a cabo:

- *Probability Theory: A Comprehensive Course* ([14]), escrito por A. Kenkle, ha sido una referencia constante a lo largo de todo el trabajo, especialmente en los capítulos sobre probabilidad y teoría de la información.
- *Convex Optimization* ([5]), de Stephen Boyd y Lieven Vandenberghe, se ha empleado en el capítulo de optimización convexa.
- *The Nature of Statistical Learning Theory* ([33]), escrito por Vladimir N. Vapnik. En él se recoge la teoría del aprendizaje estadístico y ha servido para plantear la mayoría de los elementos del capítulo 6.

- *Deep Learning* ([12]), escrito por Goodfellow, Bengio y Courville. Es la referencia principal sobre aprendizaje profundo, el cual ha servido para estructurar el capítulo 7.
- Los artículos *COVIDGR Dataset and COVID-SDNet Methodology for Predicting COVID-19 Based on Chest X-Ray Images* ([32]) y *FuCiTNet: Improving the generalization of deep learning networks by the fusion of learned class-inherent transformations* ([25]), en los que participan los tutores del trabajo, son la motivación, el punto de partida del mismo y donde se describen los modelos implementados.
- El artículo *Federated Learning and Differential Privacy: Software tools analysis, the Sherpa.ai FL framework and methodological guidelines for preserving data privacy* ([26]) de la cual es autora, entre otros, Nuria Rodríguez Barroso, y que ha servido para motivar el problema de aprendizaje federado. Además en él se presenta y describe una de las librerías empleadas en la experimentación.

2. Objetivos

Los objetivos planteados al inicio del trabajo son los siguientes:

A. Matemáticas. El objetivo principal es entender el formalismo de las redes profundas para lo cual se busca:
   - A1. Estudiar los conceptos matemáticos en que se fundamentan.
   - A2. Estudiar la definición y fundamentos teóricos de las redes neuronales prealimentadas.

B. Informática. El objetivo principal es, usando radiografías de tórax de pacientes, plantear un modelo de aprendizaje federado para diagnósticar la enfermedad COVID-19 y simular diversos escenarios con varios clientes en que implementar dicho modelo. Para ello se pretende:
   - B1. Implementar la metodología SDNET y FUCITNET y adaptarla al escenario federado.
   - B2. Imitar los resultados conseguidos en [32] sobre un nuevo conjunto de datos.
   - B3. Simular diversas configuraciones con varios clientes para obtener datos sobre la validez del modelo federado.

Los objetivos de la parte de matemáticas han sido totalmente satisfechos:

- A1 queda cubierto por los capítulos 3, 4 y 5 en los que se estudian la teoría de la probabilidad, la teoría de la información y optimización convexa. El Capítulo 6 también está relacionado con este objetivo ya que la teoría del aprendizaje es un pilar muy importante en el desarrollo de redes neuronales.
- El objetivo A2 se cumple a través del Capítulo 7 ya que en él se presentan formalmente las redes neuronales prealimentadas y la clase de funciones que implementan. Además, en este capítulo estudiamos formalmente dos de las variantes más importantes: las redes neuronales convolucionales y las redes generativas antagónicas.

Los objetivos de la titulación de informática también podemos considerarlos cumplidos:

- El objetivo de implementación B1 se lleva a cabo a través del software SDNETLearning, que describimos en la Subsección 10.1.2.
- El objetivo B2 también se ha cumplido, como se puede comprobar en la Sección 10.2, en la cual analizamos los resultados de los experimentos en el escenario centralizado.
- Por último, el objetivo B3 se ha llevado también a cabo, y prueba de ello es la Sección 10.3, en la que analizamos los resultados de los diversos experimentos que hemos efectuado en el escenario federado.

En el Tabla 2.1 recogemos los aspectos formativos de la titulación que tienen más relación con el trabajo.

Matemáticas	Informática
Probabilidad	Aprendizaje automático
Inferencia estadística	Visión por computador
Análisis matemático	Ingeniería del conocimiento
Álgebra lineal	Estructuras de datos
Topología	Programación y diseño orientado a objetos
Métodos numéricos	Inteligencia de negocio

Tabla 2.1.: Aspectos formativos relacionados con este trabajo

Parte II.

Matemáticas

3. Probabilidad

En este capítulo presentamos los resultados fundamentales sobre probabilidad en los que se apoya el resto del trabajo. La probabilidad surge de forma natural con el concepto de medida, y junto a este se encuentra el concepto de integral respecto a una medida:

$$\int_\Omega f \, d\mu$$

En este trabajo se supone conocido el significado de la anterior expresión y por ello, no vamos a repetir aquí la construcción de dicha integral. Nos limitaremos a recordar brevemente algunos conceptos y resultados sobre teoría de la medida que tendremos que usar. Las referencias relacionadas son [7], [14] y [27].

3.1. Algunos resultados sobre integración

Esta sección está orientada a fijar la notación y recordar algunos conceptos estudiados en teoría de la medida. La prueba de todos los resultados que aquí se presentan puede encontrarse en las referencias citadas anteriormente.

Dado un conjunto Ω vamos a denotar por $\mathcal{P}(\Omega)$ a las partes de Ω.

Definición 3.1 (σ-álgebra)**.** Sea Ω un conjunto. Denotamos por . Decimos que $\mathcal{A} \subset \mathcal{P}(\Omega)$ es una *σ-álgebra* si cumple:

1. $\emptyset \in \mathcal{A}$.
2. Si $A \in \mathcal{A}$, entonces $\Omega \setminus A \in \mathcal{A}$.
3. Si $\{A_n\}_{n\in\mathbb{N}} \subset \mathcal{A}$ es una colección numerable de subconjuntos, entonces $\cup_{n=1}^{\infty} A_n \in \mathcal{A}$

Si Ω es un subconjunto y $\mathcal{A}$ es una σ-álgebra en Ω, a la dupla $(\Omega, \mathcal{A})$ se le llama *espacio medible*.

Definición 3.2 (Medida)**.** Sea $(\Omega, \mathcal{A})$ un espacio medible. Una *medida* en $(\Omega, \mathcal{A})$, o simplemente en Ω, es una aplicación $\mu \colon \mathcal{A} \to \mathbb{R}$ que cumple

1. $\mu(A) \geq 0$ para cada $A \in \mathcal{A}$.
2. Si $\{A_n\}_{n\in\mathbb{N}} \subset \mathcal{A}$ es una colección numerable de subconjuntos disjuntos dos a dos, entonces $\mu(\cup_{n=1}^{\infty} A_n) = \sum_{n=1}^{\infty} \mu(A_n)$.

Para hacer referencia a un espacio medible $(\Omega, \mathcal{A})$ donde hay definida una medida μ escribiremos $(\Omega, \mathcal{A}, \mu)$ y lo llamaremos *espacio de medida*. Si $\mu(\Omega) = 1$, diremos que μ es una *medida de probabilidad* y que $(\Omega, \mathcal{A}, \mu)$ es un *espacio de probabilidad*. En un espacio de probabilidad, los elementos de $\mathcal{A}$ reciben el nombre de *eventos*.

Ejemplo 3.1 (σ-álgebra y medidas de Borel y de Lebesgue)**.**

1. Todo espacio topológico se considerará un espacio medible con la σ-álgebra generada por la familia de abiertos de la topología. La *σ-álgebra de Borel* en $\mathbb{R}^n$ es la σ-álgebra generada por la familia de abiertos de $\mathbb{R}^n$. La denotaremos por $\mathcal{B}(\mathbb{R}^n)$ y sus conjuntos recibirán el nombre de *borelianos*. Es un resultado conocido que $\mathcal{B}(\mathbb{R}^n)$ es la σ-álgebra conocida por los intervalos acotados de $\mathbb{R}^n$.

2. La *σ-álgebra de Lebesgue* es la mayor σ-álgebra en $\mathbb{R}^n$ que contiene a los intervalos acotados. Sobre ella se define la medida de Lebesgue, que en este trabajo notaremos por λ^n.

Definición 3.3 (Aplicación medible)**.** Sean $(\Omega, \mathcal{A})$ y $(\Omega', \mathcal{A}')$ dos espacios medibles. Una aplicación $X\colon \Omega \to \Omega'$ se dice *medible* si $X^{-1}(A') \in \mathcal{A}$ para cada $A' \in \mathcal{A}'$.

Ejemplo 3.2.

1. La composición de funciones medibles es medible.

2. La restricción de una función medible a un subconjunto medible es medible (con la σ-álgebra inducida).

3. La función característica de un subconjunto $A \subset \Omega$ es la función definida por

$$\mathbb{1}_A(x) = \begin{cases} 1 & \text{si } x \in A \\ 0 & \text{si } x \notin A \end{cases}$$

   Se cumple que $\mathbb{1}_A$ es medible en $(\Omega, \mathcal{A})$ si y sólo si $A \in \mathcal{A}$.

4. Las proyecciones coordenadas son funciones medibles. Estas son las funciones $r_i\colon \mathbb{R}^n \to \mathbb{R}$ definidas por $r(x_1, \ldots, x_n) = x_i$.

En un espacio de medida $(\Omega, \mathcal{A}, \mu)$ se define el concepto de integral de una función medible real respecto de la medida μ, que da sentido a la expresión:

$$\int_\Omega f \, d\mu$$

La construcción de tal integral sucede primero para funciones simples positivas, después para funciones medibles positivas como límite de tales funciones y posteriormente para funciones medibles. Si la integral de f respecto de una medida μ existe, esto es,

$$\int_\Omega f \, d\mu < \infty$$

diremos que f es μ-integrable. Suponemos conocida la noción de integral para presentar los siguientes resultados.

Proposición 3.1. *Sean $(\Omega, \mathcal{A})$ y $(\Omega', \mathcal{A}')$ dos espacios medibles. Sea μ una medida en Ω y sea $X\colon \Omega \to \Omega'$ medible. Definimos $\mu_X = \mu \circ X^{-1}$, la cual es una medida en Ω'. Sea $f\colon \Omega' \to \mathbb{R}$ μ_X-integrable en Ω'. Entonces $f \circ X$ es μ-integrable en Ω y se cumple*

$$\int_\Omega (f \circ X) \; d\mu = \int_{\Omega'} f \, d(\mu \circ X^{-1})$$

Para acabar esta breve sección vamos a recordar el concepto de densidad, que nos ayudará a calcular la esperanza de una variable aleatoria usando la integral de Lebesgue.

Proposición 3.2. *Sean $(\Omega, \mathcal{A}, \mu)$ un espacio de medida y $f\colon \Omega \to \mathbb{R}_0^+$ una función medible. Definimos $\nu\colon \mathcal{A} \to \mathbb{R}$ como*

$$\nu(A) = \int_A f \, d\mu = \int_\Omega \mathbb{1}_A f \, d\mu, \quad \forall A \in \mathcal{A}$$

Entonces ν es una medida en Ω.

En las condiciones de la proposición anterior, escribiremos $f\mu = \nu$ y diremos que ν tiene *función de densidad* f respecto a μ. Cuando la medida μ se corresponda con la medida de Lebesgue, diremos directamente que ν tiene *función de densidad* f.

Proposición 3.3. *Sea $(\Omega, \mathcal{A}, \mu)$ un espacio de medida y sean $f\colon \Omega \to \mathbb{R}_0^+$ y $g\colon \Omega \to \mathbb{R}$ funciones medibles. Entonces se cumple que g es $f\mu$-integrable en Ω si y sólamente si gf es μ-integrable en Ω, en cuyo caso:*

$$\int_\Omega g \, d(f\mu) = \int_\Omega gf \, d\mu$$

3.2. Variables aleatorias

Definición 3.4 (Variable aleatoria)**.** Sea $(\Omega, \mathcal{A}, P)$ un espacio de probabilidad y $(\Omega', \mathcal{A}')$ un espacio medible. Decimos que $X\colon \Omega \to \Omega'$ es una variable aleatoria con valores en Ω' si X es una aplicación medible.

Observación 3.1.

1. En la definición de variable aleatoria, si $\Omega' = \mathbb{R}$ decimos que X es una variable aleatoria real y si $\Omega' = \mathbb{R}^n$ decimos que X es un vector aleatorio real.

2. Para $A' \in \mathcal{A}'$, escribiremos $P[X \in A'] = P\left[X^{-1}(A')\right]$.

Dado un vector aleatorio real X que toma valores en $\mathbb{R}^n$ la aplicación $X_i = r_i \circ X$ es medible, y por tanto es una variable aleatoria real. De la misma forma, dadas $X_1, \ldots, X_n$ variables aleatorias, la aplicación $X = (X_1, \ldots X_n)$ que aplica X_i en cada componente es un vector aleatorio real.

Definición 3.5. Sea $X\colon \Omega \to \Omega'$ una variable aleatoria. Si $X(\Omega)$ es un conjunto numerable, entonces decimos que X es una variable aleatoria *discreta*. En otro caso diremos que X es una variable aleatoria *continua*.

Dada una variable aleatoria X se puede comprobar que $P \circ X^{-1}$ es una medida de probabilidad en $(\Omega', \mathcal{A}')$. Así surge el concepto de distribución, que presentamos a continuación y que será especialmente importante en este trabajo.

Definición 3.6 (Distribución de probabilidad)**.** Sea $(\Omega, \mathcal{A}, P)$ un espacio de probabilidad y $(\Omega', \mathcal{A}')$ un espacio medible. Sea $X\colon \Omega \to \Omega'$ una variable aleatoria con valores en Ω'.

1. La medida de probabilidad $P_X = P \circ X^{-1}$ se llama la *distribución* de X. Escribiremos $X \sim \mu$ si $\mu = P_X$ y diremos que X tiene distribución μ.

2. Cuando $\Omega' = \mathbb{R}^n$, se define la aplicación $F_X \colon X(\Omega) \to \mathbb{R}$ como

$$F_X(x) = P\left[X \leq x\right] = P_X\left(]-\infty, x]^n\right)$$

y se le llama la *función de distribución* de P_X.

Diremos que una familia de variables aleatorias $\{X_1, \ldots, X_n\}$ son o están *idénticamente distribuidas* si $P_{X_i} = P_{X_j}$ para cualesquiera i, j.

Dado un vector aleatorio real $X = (X_1, \ldots, X_n)$ podemos hablar de las distribuciones de cada una de las variables aleatorias que lo componen: P_{X_i}. A estas distribuciones les llamaremos *distribuciones marginales*. Las *funciones de distribución marginales* vendrán dadas por:

$$F_{X_i}(x_i) = \lim_{x_j \to +\infty, j \neq i} F_X(x_1, \ldots, x_i, \ldots, x_n)$$

Definición 3.7 (Función masa de probabilidad)**.** Sea $(\Omega, \mathcal{A}, P)$ un espacio de probabilidad. Sea $X \colon \Omega \to \mathbb{R}^n$ una variable aleatoria discreta. La función $p_X \colon X(\Omega) \to [0, 1]$ definida por

$$p_X(x) = P_X(x) = P(X = x), \quad \forall x \in X(\Omega)$$

se denomina la *función masa de probabilidad de X o de P_X*.

Definición 3.8 (Función de densidad)**.** Sea $(\Omega, \mathcal{A}, P)$ un espacio de probabilidad. Sea $X \colon \Omega \to \mathbb{R}^n$ una variable aleatoria continua tal que P_X tiene densidad p_X respecto de la medida de Lebesgue en R^n. A la función p_X se denomina la *función de densidad de X o de P_X*.

Obsérvese que si una variable aleatoria real continua tiene función de densidad ocurre que:

$$P_X(x) = \lim_{\epsilon \to 0} \int_{x-\epsilon}^{x+\epsilon} p_X(t)dt = 0$$

por lo que no tiene sentido hablar de función masa de probabilidad en tal caso.

Observación 3.2. Nos interesan especialmente aquellas distribuciones de probabilidad P_X en $\mathbb{R}^n$ que son *absolutamente continuas* respecto a la medida de Lebesgue. En ese caso, el teorema de Radon-Nikodym ([14]) asegura que existe una función $f \colon \mathbb{R}^n \to \mathbb{R}_0^+$ tal que $P_X = f\lambda^n$.

3.3. Independencia de variables aleatorias

Definición 3.9. Sea $(\Omega, \mathcal{A}, P)$ un espacio de probabilidad y sea $\{(\Omega_i, \mathcal{A}_i) \colon i \in \{1, \ldots, n\}\}$ una familia finita de espacios medibles. Para cada i sea $X_i \colon \Omega \mapsto \Omega_i$ una variable aleatoria. Diremos que $\{X_1, \ldots, X_n\}$ es una familia de variables aleatorias independientes si se cumple:

$$P\left(X_1 \in A_1, \ldots, X_n \in A_n\right) = \prod_{i=1}^{n} P\left(X_i \in A_i\right)$$

para cualesquiera $A_1 \in \mathcal{A}_1, \ldots, A_n \in \mathcal{A}_n$

Supongamos que $\{X_1, \ldots, X_n\}$ son variables aleatorias reales y llamemos $X = (X_1, \ldots, X_n)$. De la definición de independencia se deduce que $\{X_1, \ldots, X_n\}$ son independientes si y sólo si se cumple:

$$F_X(x_1, \ldots, x_n) = F_{X_1}(x_1) \cdots F_{X_n}(x_n)$$

para todo $(x_1, \ldots, x_n) \in \mathbb{R}^n$.

Si cada P_{X_i} tiene función de densidad f_i, el teorema de Fubini nos permite asegurar que P_X tiene función de densidad dada por:

$$f(x_1, \ldots, x_n) = f_1(x_1) \cdots f_n(x_n)$$

3.4. Momentos de una variable aleatoria

Definición 3.10 (Momentos)**.** Sea $(\Omega, \mathcal{A}, P)$ un espacio de probabilidad y sea X una variable aleatoria real.

1. Si X es P-integrable en Ω, definimos la *esperanza* de X como

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

2. Si $k \in \mathbb{N}$ y X^k es P-integrable en Ω, definimos el *momento centrado de orden k de X* como

$$m_k = \mathbb{E}\left[X^k\right]$$

Para un vector aleatorio real $X = (X_1, \ldots, X_n)$ se define la esperanza y los momentos centrados de orden k componente a componente:

$$\mathbb{E}[X] = (\mathbb{E}[X_1], \ldots, \mathbb{E}[X_n]), \quad m_k = \left(\mathbb{E}[X_1^k], \ldots, \mathbb{E}[X_n^k]\right).$$

A continuación definimos la varianza de una variable aleatoria.

Definición 3.11 (Varianza)**.** Sea $(\Omega, \mathcal{A}, P)$ un espacio de probabilidad y sea X una variable aleatoria real. Si X y X^2 son P-integrable en Ω, se define la *varianza* de X como

$$\operatorname{var}(X) = \mathbb{E}\left[(X - \mathbb{E}[X])^2\right]$$

La siguiente igualdad justifica las condiciones exigidas en la definición:

$$\operatorname{var}(X) = \mathbb{E}\left[X^2\right] - (\mathbb{E}[X])^2.$$

Para un vector aleatorio real $X = (X_1, \ldots, X_n)$ se define la varianza como:

$$\operatorname{var}(X) = \mathbb{E}\left[\|X - \mathbb{E}[X]\|^2\right] = \mathbb{E}\left[\|X\|^2\right] - \|\mathbb{E}[X]\|^2.$$

Definición 3.12 (Covarianza)**.** Sea $(\Omega, \mathcal{A}, P)$ un espacio de probabilidad y sean X, Y variables aleatorias reales. Si X, Y y XY son P-integrable en Ω, se define la *covarianza* de X como

$$\operatorname{cov}(X, Y) = \mathbb{E}[(X - \mathbb{E}[X])(Y - \mathbb{E}[Y])]$$

Se cumple que

$$\operatorname{cov}(X, Y) = \mathbb{E}[XY] - \mathbb{E}[X]\,\mathbb{E}[Y]$$

Si $\operatorname{cov}(X, Y) = 0$, decimos que X y Y no están correladas. En particular, cuando X e Y son independientes, entonces $\operatorname{cov}(X, Y) = 0$.

Cálculo de la esperanza

Fijemos un espacio de probabilidad $(\Omega, \mathcal{A}, P)$. A efectos de poder calcular la esperanza de una variable aleatoria, es especialmente interesante expresar la esperanza respecto a la distribución de X. La proposición 3.1 nos permite escribir:

$$\int_{\Omega} (f \circ X)\, dP = \int_{X(\Omega)} f\, dP_X.$$

La igualdad anterior motiva el siguiente resultado, cuya prueba se deriva de la proposición 3.1 y de la proposición 3.3, que nos permitirá calcular la esperanza de funciones de variables aleatorias.

Proposición 3.4. *Sea $(\Omega, \mathcal{A}, P)$ un espacio de probabilidad y $(\Omega', \mathcal{A}')$ un espacio medible. Sea $X\colon \Omega \to \Omega'$ una variable aleatoria y $f\colon X(\Omega) \to \mathbb{R}$ una función tal que $f \circ X$ es P-integrable. Entonces*

$$\mathbb{E}\left[f(X)\right] = \int_{X(\Omega)} f\, dP_X$$

Además:

- *Si $\Omega' = \mathbb{R}^n$ y P_X tiene función de densidad p_X respecto de la medida de Lebesgue en R^n se cumple:*

$$\mathbb{E}\left[f(X)\right] = \int_{X(\Omega)} p_X f\, d\lambda^n = \int_{X(\Omega)} p_X(x) f(x)\, dx$$

- *Si $\Omega' = \mathbb{R}^n$ y X es una variable aleatoria discreta, entonces*

$$\mathbb{E}\left[f(X)\right] = \sum_{x \in X(\Omega)} p_X(x) f(x)$$

La anterior expresión tiene la ventaja de que en ella no interviene la medida de probabilidad sobre el espacio de partida, sólo la distribución de la variable aleatoria. Es muy común escribir

$$\mathbb{E}_{X \sim P_X}\left[X\right] = \int_{X(\Omega)} x\, dP_X(x)$$

haciendo de esta forma referencia explícitamente a la distribución de la variable aleatoria y obviando la medida de probabilidad sobre el espacio de partida. Cuando quede claro la distribución a la que nos referimos, escribiremos simplemente $\mathbb{E}_X[X]$.

3.5. Probabilidad condicionada

Terminamos este capítulo con un breve repaso al concepto de la probabilidad condicionada. Recordemos en primer lugar la definición clásica de probabilidad condicionada para eventos de un espacio de probabilidad. Sea B un evento en un espacio de probabilidad cumpliendo $P(B) > 0$. Se definía la probabilidad del evento A condicionada a la ocurrencia del evento B

como:

$$P(A \mid B) = \frac{P(A \cup B)}{P(B)}$$

Traslademos este concepto ahora a las variables aleatorias. Sea $(\Omega, \mathcal{A}, P)$ un espacio de probabilidad y sean $X, Y \colon \Omega \to \mathbb{R}$ dos variables aleatorias reales. Consideremos, para cada $x \in X(\Omega)$ y para cada intervalo $]y-\delta, y+\delta] \subset Y(\Omega)$ tal que $P(y-\delta < Y \leq y+\delta) > 0$, la expresión:

$$P(X \leq x \mid y-\delta < Y \leq y+\delta) = \frac{P(X \leq x, y-\delta < Y \leq y+\delta)}{P(y-\delta < Y \leq y+\delta)}$$

La expresión anterior representa la distribución de probabilidad de X dado que $Y \in]y-\delta, y+\delta]$. El interés de tal probabilidad reside en el caso en que el límite cuando δ tiende a cero por la derecha existe.

Definición 3.13. Sea $(\Omega, \mathcal{A}, P)$ un espacio de probabilidad y sean $X, Y \colon \Omega \to \mathbb{R}$ dos variables aleatorias reales. En el caso en que el límite anterior exista, se define la *función de distribución de X condicionada a que $Y = y$* como:

$$F_{X|Y}(x|y) = \lim_{\delta \mapsto 0+} P(X \leq x \mid y-\delta < Y \leq y+\delta)$$

Nótese que conocer la función de distribución es equivalente a conocer la distribución de probabilidad de una variable aleatoria. Por tanto, podemos hablar de la *distribución de X condicionada a que $Y = y$* y denotarla por $P_{X|Y}$.

Sea (X, Y) un vector aleatorio real. Supongamos que la función de densidad de $P_{(X,Y)}$ es f y que la función de densidad de P_Y es f_Y y cumple $f_Y > 0$. Supongamos también que f y f_Y son continuas. Tenemos:

$$\begin{aligned} F_{X|Y}(x|y) &= \lim_{\delta \mapsto 0+} \frac{P(X \leq x, y-\delta < Y \leq y+\delta)}{P(y-\delta < Y \leq y+\delta)} = \\ &= \lim_{\delta \mapsto 0+} \frac{\int_{-\infty}^{x} \int_{y-\delta}^{y+\delta} f(u,v)\, dudv}{\int_{y-\delta}^{y+\delta} f_Y(v)\, dv} \end{aligned}$$

Diviendo numerador y denominador por 2δ y efectuando el límite, por ser f y f_Y continuas obtenemos que:

$$F_{X|Y}(x|y) = \frac{\int_{-\infty}^{x} f(u,y)\, du}{f_Y(y)} = \int_{-\infty}^{x} \frac{f(u,y)}{f_Y(y)}\, du$$

Hemos probado el siguiente resultado:

Proposición 3.5. *Sea $(\Omega, \mathcal{A}, P)$ un espacio de probabilidad y sea $(X, Y) \colon \Omega \to \mathbb{R}^2$ un vector aleatorio real. Supongamos que las funciones de densidad de $P_{(X,Y)}$ y P_Y, que denotamos por f y f_Y, son continuas y $f_Y > 0$. Entonces la función de densidad de $P_{X|Y}$ existe y viene dada por*

$$f_{X|Y}(x|y) = \frac{f(x,y)}{f_Y(y)}$$

para cada $x \in X(\Omega), y \in Y(\Omega)$.

Los resultados, definiciones y comentarios anteriores se puede extender sistemáticamente a vectores aleatorios reales de n componentes.

4. Teoría de la información

En este capítulo pretendemos introducir brevemente las ideas principales de la teoría de la información. Esta teoría se sitúa en la intersección de la probabilidad, de la estadística, las ciencias de la computación y la ingeniería en general, ya que sus resultados son aplicables en una amplia gama de campos y situaciones.

Un concepto clave es el de entropía. Este concepto tiene numerosas interpretaciones. Por un lado la podemos entender como una medida de la cantidad de incertidumbre (o desorden) presente en una muestra aleatoria o en un proceso aleatorio. También puede entenderse como una medida de la cantidad de información presente en una señal.

En el contexto del aprendizaje automático sólo usamos algunas ideas clave para caracterizar distribuciones de probabilidad y medir la similaridad entre ellas.

4.1. Entropía

Claude Elwood Shannon presentó el concepto de entropía en [30] para estudiar la longitud de mensajes construidos a partir de un alfabeto discreto y que se mandaban por un canal ruidoso. En el contexto de este trabajo vamos a definir la entropía para distribuciones arbitrarias que provengan de una función de densidad.

Definición 4.1 (Entropía)**.** Sea X un vector aleatorio. Supongamos que la función de distribución de X tiene densidad o función masa de probabilidad p. Definimos la *entropía* de X como

$$H(X) = -\mathbb{E}_{X\sim P_X}\left[\log(p(X))\right]$$

siempre que exista la esperanza de la derecha. De forma más general, podemos hablar de la *entropía* de una distribución de probabilidad P en $\mathbb{R}^n$ que tiene densidad p:

$$H(P) = -\mathbb{E}_{X\sim P}\left[\log(p(X))\right]$$

En la definición de entropía es importante la elección de la base del logaritmo. Si en la definición anterior tomamos el logaritmo en base 2, entenderemos que la entropía es una medida que se expresa en bits. Veamos un ejemplo:

Ejemplo 4.1. Sea X una variable aleatoria discreta con imagen $\{a, b, c, d\}$. Supongamos que:

$$P[X = x] = \begin{cases} \frac{1}{2}, & \text{si } x = a, \\ \frac{1}{4}, & \text{si } x = b, \\ \frac{1}{8}, & \text{si } x = c, \\ \frac{1}{8}, & \text{si } x = d, \end{cases}$$

La entropía de X tomando el logaritmo en base 2 es:

$$H(X) = -\frac{1}{2}\log\left(\frac{1}{2}\right) - \frac{1}{4}\log\left(\frac{1}{4}\right) - \frac{1}{8}\log\left(\frac{1}{8}\right) - \frac{1}{8}\log\left(\frac{1}{8}\right) = \frac{7}{4}$$

Si quisiéramos determinar el valor de X con el mínimo número de preguntas binarias (del tipo ¿es $X = x$?) sería una buena idea empezar preguntando si $X = a$ ya que es el valor más probable. A continuación podemos preguntar por b, y luego por c o d. Podemos ver que el número esperado de preguntas binarias para descubrir el valor de X es $\frac{7}{4}$.

4.2. Divergencia

El concepto de divergencia nos permite medir cuánto se parecen dos distribuciones de probabilidad. Hay distintos conceptos de divergencia, pero todos se encuadran dentro de la siguiente definición.

Definición 4.2 (Divergencia)**.** Sea X un conjunto. Una *divergencia* es una aplicación $D\colon X \times X \to \mathbb{R}$ que cumple

1. Para cualesquiera $x, y \in X$ se cumple $D(x, y) \geq 0$.
2. $D(x, y) = 0$ si y sólo si $x = y$.

La divergencia de Kullback-Leibler también se conoce como entropía relativa. Fue presentada en [16], donde se prueba la llamado *desigualdad de la información*.

Definición 4.3 (Divergencia Kullback-Leibler)**.** Sean P y Q dos medidas de probabilidad en $\mathbb{R}^n$ de las que se conocen funciones de densidad o funciones masa de probabilidad, p y q respectivamente. Se define la *divergencia Kullback-Leibler* de P y Q como

$$D_{KL}(P, Q) = \mathbb{E}_{X \sim P}\left[\log\left(\frac{p(X)}{q(X)}\right)\right]$$

La divergencia de Kullback-Leibler tiene una propiedad que no parece deseable. A priori, si la divergencia es una medida del parecido entre dos distribuciones de probabilidad, cabría esperar que si "P se parece a Q", entonces "Q se parece a P". Sin embargo $D_{KL}(P, Q) = -D_{KL}(Q, P)$. Por otra parte, tampoco cumple la desigualdad triangular, por lo que no podemos considerarla una distancia.

La divergencia de Jensen-Shannon, que definimos a continuación a partir de la de Kullback-Leibler, sí que presenta esta propiedad "deseable"de simetría:

Definición 4.4 (Divergencia Jensen-Shannon)**.** Sean P y Q dos medidas de probabilidad en $\mathbb{R}^n$ de las que se conocen funciones de densidad o funciones masa de probabilidad, p y q respectivamente. Se define la *divergencia Jensen-Shannon* como

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

donde $M = \frac{1}{2}(P + Q)$.

Observación 4.1. Nótese que M vuelve a ser una distribución de probabilidad que tiene densidad o función masa de probabilidad $\frac{p+q}{2}$.

Ya estamos en condiciones de presentar la desigualdad de la información, la cual nos confirma que D_{KL} es una divergencia. Usaremos este resultado cuando desarrollemos la teoría de las redes generativas antagónicas. Para probarlo usamos un resultado que vamos a demostrar en el siguiente capítulo.

Teorema 4.1 (Desigualdad de la información)**.** *Sean P y Q dos medidas de probabilidad en $\Omega \subset \mathbb{R}^n$ de las que se conocen funciones de densidad, p y q respectivamente. Entonces*

1. $D_{KL}(P, Q) \geq 0$
2. $D_{KL}(P, Q) = 0$ *si y sólamente si* $p = q$ *en casi todo punto.*

Demostración. Recordemos que $-\log$ es una función convexa. Aplicando la desigualdad de Jensen (Corolario 5.1) obtenemos:

$$D_{KL}(P, Q) = \int_\Omega p(x) \left(-\log \left(\frac{q(x)}{p(x)} \right) \right) dx \geq -\log \left(\int_\Omega p(x) \frac{q(x)}{p(x)} dx \right) = -\log(1) = 0$$

Si $D_{KL}(P, Q) = 0$, entonces se da la igualdad en la desigualdad de Jensen, lo cual en este caso conduce a que $x \mapsto \frac{q(x)}{p(x)}$ sea constante e igual a 1 en casi todo punto. Recíprocamente, si $p(x) = q(x)$ para casi todo $x \in \Omega$ entonces es claro que $D_{KL}(P, Q) = 0$. □

Corolario 4.1. *Sean P y Q dos medidas de probabilidad en $\Omega\mathbb{R}^n$ de las que se conocen funciones de densidad, p y q respectivamente. Entonces*

1. $D_{JS}(P, Q) \geq 0$
2. $D_{JS}(P, Q) = 0$ *si y sólamente si* $p = q$ *en casi todo punto.*

Observación 4.2. Los anteriores resultados también son ciertos si p y q son funciones masa de probabilidad. Además, se cumple que $p = q$ en casi todo punto si y sólamente si $p = q$ en todo punto.

4.3. Entropía cruzada

Definición 4.5 (Entropía cruzada)**.** Sean P y Q dos medidas de probabilidad en $\mathbb{R}^n$ de las que se conocen sus funciones de densidad o funciones masa de probabilidad p y q respectivamente. Se define la *entropía cruzada* de P y Q como

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

Denotando por p y q a las funciones de densidad de P y Q observamos que

$$\begin{aligned} H(P, Q) &= -\int_{R^n} p(x) \log(p(x))\, dx + \int_{R^n} p(x) \log \left(\frac{p(x)}{q(x)} \right) dx = \\ &= -\int_{R^n} p(x) \log(q(x))\, dx = \\ &= -\mathbb{E}_{X \sim P} \left[\log \left(q(X) \right) \right] \end{aligned}$$

Podemos obtener una igualdad análoga si p y q son funciones masa de probabilidad. A la vista de ella, podemos generalizar la definición de $H(P, Q)$ al caso en que sólo conocemos la función de densidad o función masa de probabilidad de Q.

Definición 4.6 (Entropía cruzada)**.** Sean P y Q dos medidas de probabilidad en $\mathbb{R}^n$. Supongamos que q es la función de densidad o función masa de probabilidad de Q. Se define la *entropía cruzada* de P y Q como

$$H(P, Q) = -\mathbb{E}_{X \sim P} \left[\log \left(q(X) \right) \right]$$

Minimizar la entropía cruzada es equivalente a minimizar la divergencia Kullback-Leibler. Es común fijar una de las distribuciones, P, y buscar la distribución Q que más se parece a P minimizando la entropía cruzada.

5. Optimización basada en gradiente descendente

El problema del aprendizaje es formulado de forma natural como un problema de optimización. Dedicamos este capítulo a estudiar estos problemas y el algoritmo del gradiente descendente, que es una de las técnicas que mejores resultados proporcionan en la práctica.

En este capítulo vamos a definir lo que es un problema de optimización. Nos centraremos en problemas de optimización convexos, por lo que será necesario estudiar algunas propiedades útiles de las funciones convexas.

Por último, presentaremos el algoritmo de gradiente descendente. Estudiaremos la convergencia de este método para funciones convexas y señalaremos brevemente algunas de sus variantes.

5.1. Optimización

Definición 5.1. Un problema de *minimización* es un problema de la forma:

$$\min_{x \in \Omega} f(x),$$

donde $f \colon \Omega \to \mathbb{R}$ una función que llamaremos *función objetivo*. De forma análoga, un problema de *maximización* es un problema de la forma:

$$\max_{x \in \Omega} f(x)$$

Diremos que un problema es de *optimización* si es de minimización o de optimización.

Nótese que un problema de optimización para una función f se convierte en un problema de minimización para la función $-f$, y viceversa. Por tanto, no perdemos generalidad si nos centramos en problemas de minimización.

Decimos que un problema de optimización es *convexo* si la función objetivo es convexa. En la siguiente sección definiremos qué es una función convexa y estudiaremos algunas propiedades útiles de las mismas.

5.2. Funciones convexas

5.2.1. Definición y propiedades

Definición 5.2 (Conjunto convexo)**.** Un conjunto $K \subset R^d$ se dice que es convexo si para cualesquiera $x, y \in K$, $t \in [0,1]$ se cumple $tx + (1-t)y \in K$.

Definición 5.3 (Función convexa)**.** Sea $K \subset \mathbb{R}^d$ convexo y sea $f \colon K \to \mathbb{R}$. Decimos que f es convexa si para cada $x, y \in K$ y cada $t \in [0,1]$ se cumple

$$f\left(tx + (1-t)y\right) \leq tf(x) + (1-t)f(y)$$

Diremos que f es estrictamente convexa si la desigualdad anterior es estricta, es decir, si para cada $x, y \in K$ con $x \neq y$ y para cada $t \in [0,1]$ se cumple:

$$f\left(tx + (1-t)y\right) < tf(x) + (1-t)f(y)$$

Cuando las desigualdades anteriores se den en sentido contrario, diremos que f es *cóncava* o *estrictamente cóncava*. Es claro que f es convexa (respectivamente estrictamente convexa) si y sólo si $-f$ es cóncava (respectivamente estrictamente cóncava).

Es necesario exigir que el conjunto en que esté definido la función sea convexo, pues en otro caso las expresiones no tendrían sentido. Cuando $d = 1$, la interpretación geométrica de la definición nos sugiere que la gráfica de la función f se encuentra por debajo del segmento de extremos x e y, sea cuales sean estos extremos.

Comenzamos señalando dos propiedades importantes de las funciones convexas, las cuales serán importantes de cara a la optimización.

Proposición 5.1. *Sea $K \subset \mathbb{R}^d$ convexo y abierto y sea $f\colon K \to \mathbb{R}$ convexa. Entonces*

1. *Cualquier mínimo local de f es un mínimo global.*
2. *Si f es diferenciable, entonces para cualesquiera $x, y \in K$ se cumple*

$$\nabla f(y)^T (x - y) \leq f(x) - f(y) \tag{5.1}$$

Demostración.

1. Supongamos que x^* es un mínimo local. Entonces existe $r > 0$ tal que $f(x^*) \leq f(x)$ para cada $x \in B(x^*, r)$. Supongamos para llegar a contradicción que existe $z \in K$ tal que $f(z) < f(x^*)$. Entonces, para $t \in [0,1]$:

   $$f(tx^* + (1-t)z) \leq tf(x^*) + (1-t)f(z) < f(x^*)$$

   Evaluando en $t = 1$ se obtiene que $f(x^*) < f(x^*)$, lo cual no es posible.

2. De la definición de convexidad, se cumple que:

   $$f\left(y + t(x-y)\right) - f(y) \leq t(f(x) - f(y))$$

   Diviendo por $t > 0$ ambos términos obtenemos:

   $$\frac{f\left(y + t(x-y)\right) - f(y)}{t} \leq f(x) - f(y)$$

   Tomando límite cuando cuando t tiende a cero y aplicando la definición de diferencial obtenemos

   $$\nabla f(y)^T (x - y) = df_y(x - y) \leq f(x) - f(y)$$

□

Lema 5.1 (Lema de las tres secantes)**.** *Sea $f\colon]a,b[\to \mathbb{R}$ convexa. Si $a < s < t < u < b$ entonces*

$$\frac{f(t) - f(s)}{t - s} \leq \frac{f(u) - f(s)}{u - s} \leq \frac{f(u) - f(t)}{u - t}$$

.

Demostración. Tomamos $\lambda = \frac{t-s}{u-s}$ que claramente pertenece a $[0,1]$ y cumple $1-\lambda = \frac{u-t}{u-s}$. Aplicando la definición de convexidad:

$$f(t) = f\left(\frac{u-t}{u-s}s + \frac{t-s}{u-s}u\right) \leq \frac{u-t}{u-s}f(s) + \frac{t-s}{u-s}f(u)$$

De aquí deducimos la primera desigualdad:

$$\begin{aligned}\frac{f(t)-f(s)}{t-s} &\leq \frac{1}{t-s}\left(\frac{u-t}{u-s}f(s) + \frac{t-s}{u-s}f(u) - f(s)\right)\\ &= \frac{1}{t-s}\left(\frac{s-t}{u-s}f(s) + \frac{t-s}{u-s}f(u)\right)\\ &= \frac{f(u)-f(s)}{u-s}\end{aligned}$$

Análogamente se obtiene la segunda desigualdad, lo cual concluye la prueba. □

Proposición 5.2. *Sea $f\colon]a,b[\to\mathbb{R}$ convexa.*

1. *Existen las derivadas laterales de f en cada punto del intervalo $]a,b[$. Además, se cumple $f'_-(x) \leq f'_+(x)$ para todo $x \in]a,b[$.*

2. *Sean $x,y \in]a,b[$ tales que $x > y$. Entonces*

$$\begin{aligned}f'_+(y)(x-y) &\leq f(x)-f(y)\\ f'_-(x)(x-y) &\geq f(x)-f(y)\end{aligned}$$

Demostración.

1. Fijamos $x \in]a,b[$. En primer lugar, si $\epsilon, h > 0$ son tales que $h \leq \epsilon$ y $a < x < x+h \leq x+\epsilon < b$, podemos aplicar el lema de las tres secantes para obtener

$$\frac{f(x+h)-f(x)}{h} \leq \frac{f(x+\epsilon)-f(x)}{\epsilon}$$

   Por otra parte, si $h_1, h_2 > 0$ son tales que $a < x-h_1 < x < x+h_2 < b$ aplicamos de nuevo el lema anterior para obtener

$$\frac{f(x)-f(x-h_1)}{h_1} \leq \frac{f(x+h_2)-f(x-h_1)}{h_2+h_1} \leq \frac{f(x+h_2)-f(x)}{h_2}$$

   Las desigualdades que hemos obtenido nos permiten asegurar que la función definida por $h \mapsto \frac{f(x+h)-f(x)}{h}$ para cada $h \in]x, x+\epsilon[$ es creciente y acotada inferiormente, por lo que existe el límite cuando h tiende a cero. Esto prueba la existencia de $f'_+(x)$ y análogamente puede probarse la existencia de $f'_-(x)$. Tomando límite en la última desigualdad concluimos que $f'_-(x) \leq f'_+(x)$.

2. De la definición de convexidad, se cumple que:

$$f\left(y + t(x-y)\right) - f(y) \leq t(f(x)-f(y))$$

Diviendo por $t > 0$ ambos términos obtenemos:

$$\frac{f\left(y + t(x-y)\right) - f(y)}{t} \leq f(x) - f(y)$$

Tomamos límite cuando $t \to 0$ para obtener $f'_+(y)(x-y) \leq f(x) - f(y)$. Para obtener la otra desigualdad, intercambiamos los papeles de x e y:

$$\frac{f\left(x + t(y-x)\right) - f(x)}{t} \leq f(y) - f(x)$$

Reescribimos para poder tomar límite:

$$\frac{f\left(x - t(x-y)\right) - f(x)}{t} \leq f(y) - f(x)$$

y tomamos límite cuando $t \to 0$ obteniendo $-f'_-(x)(x-y) \leq f(y) - f(x)$, de donde se deduce la desigualdad buscada.

□

5.2.2. Desigualdad de Jensen

Si una función f es convexa se puede probar fácilmente por inducción que si $p_1, \ldots, p_n \geq 0$ y cumplen $\sum_{i=1}^{n} p_i = 1$ entonces

$$f\left(p_1 x_1 + \ldots + p_n x_n\right) \leq p_1 f(x_1) + \ldots + p_n f(x_n)$$

La desigualdad de Jensen supone una generalización del hecho anterior a espacios de probabilidad, donde la sumatoria se sustituye por una medida de probabilidad. Esta desigualdad será especialmente importante para probar la conocida como *desigualdad de la información*.

La prueba que exponemos aquí es la que Rick Durrett proporciona en [7]. La misma gira en torno al siguiente lema, que se deduce fácilmente de las propiedades que hemos obtenido para funciones convexas.

Lema 5.2. *Sea $f\colon]a,b[\to \mathbb{R}$ convexa. Sea $c \in]a,b[$. Existe $r\colon \mathbb{R} \to \mathbb{R}$ lineal cumpliendo $r(c) = f(c)$ y $r(x) \leq f(x)$ para cada $x \in]a,b[$.*

Demostración. Sea $\alpha \in \mathbb{R}$ tal que $f'_-(c) \leq \alpha \leq f'_+(c)$. Definimos $r(x) = \alpha(x-c) + f(c)$. Claramente se cumple que $r(c) = f(c)$. Sea $x \in]a,b[$. Si $x > c$ entonces

$$\alpha(x-c) \leq f'_+(c)(x-c) \leq f(x) - f(c)$$

Si $x < c$ entonces

$$\alpha(c-x) \geq f'_-(c)(c-x) \geq f(c) - f(x)$$

de donde $\alpha(x-c) \leq f'_-(c)(x-c) \leq f(x) - f(c)$. En cualquier caso hemos probado que se cumple $r(x) \leq f(x)$. □

Teorema 5.1 (Desigualdad de Jensen)**.** *Sea $(\Omega, \mathcal{A}, P)$ un espacio de probabilidad, $\varphi\colon]a,b[\to \mathbb{R}$ una función convexa y $f\colon \Omega \to \mathbb{R}$ es una función medible con $f(\Omega) \subset]a,b[$. Entonces:*

$$\varphi\left(\int_\Omega f \, dP\right) \leq \int_\Omega (\varphi \circ f) \, dP$$

Además, la igualdad se da, si y sólo si, o bien f es constante, o bien φ coincide con una función lineal en casi todo punto de $f(\Omega)$.

Demostración. Llamamos $c = \int_\Omega f\, dP$. Tomamos r como en el lema anterior. Como r es lineal podemos permutarla con la integral, obteniendo la desigualdad buscada:

$$\int_\Omega (\varphi \circ f)\, dP \geq \int_\Omega (r \circ f)\, dP = r\left(\int_\Omega f\, dP\right) = \varphi\left(\int_\Omega f\, dP\right)$$

Si se da la igualdad y f no es constante, entonces $(\varphi \circ f)(x) = (r \circ f)(x)$ para casi todo $x \in \Omega$. □

Corolario 5.1. *Sea $\Omega \subset \mathbb{R}^n$ y sea $p\colon \Omega \to \mathbb{R}_0^+$ tal que*

$$\int_\Omega p(x)\, dx = 1$$

Sea $\varphi\colon]a,b[\to \mathbb{R}$ una función convexa y $f\colon \Omega \to \mathbb{R}$ es una función medible con $f(\Omega) \subset]a,b[$. Entonces:

$$\varphi\left(\int_\Omega p(x) f(x)\, dx\right) \leq \int_\Omega p(x)\varphi(f(x))\, dx$$

Además, la igualdad se da, si y sólo si, o bien f es constante, o bien φ coincide con una función lineal en casi todo punto de $f(\Omega)$.

Demostración. Basta aplicar el resultado anterior al espacio de medida $(\Omega, \mathcal{B}^n, p\lambda^n)$. □

5.3. Gradiente descendente

Sabemos que el gradiente de una función diferenciable "apunta" o "señala" en la dirección de la máxima pendiente del grafo de la función. Es decir, nos informa de en qué dirección la función crece. Por tanto, si avanzamos en la dirección opuesta al gradiente deberíamos obtener valores más pequeños de la función.

El algoritmo de gradiente descendente se basa en, partiendo de un punto x_0, ir realizando actualizaciones del mismo en dirección opuesta al gradiente:

$$x_{t+1} = x_t - \eta \nabla f(x_t)$$

donde $\eta \in \mathbb{R}$ se denomina *tasa de aprendizaje*. Esta tasa puede ser constante o adaptarse al número de iteraciones y el valor del gradiente en cada momento. El algoritmo de gradiente descendente, tal y como lo hemos relatado, figura en algoritmo 1.

Algoritmo 1: Gradiente descendente.

Entrada: Función f, tasa de aprendizaje η
Inicializar $w(0)$
para $t = 0, 1, 2, \ldots$ **hacer**
  $w(t+1) = w(t) - \eta \nabla f(w(t))$
fin

5.3.1. Convergencia

Vamos a probar que, bajo ciertas hipótesis no muy restrictivas, el algoritmo de gradiente descendente llega a obtener un mínimo global. Supondremos que la función objetivo es convexa lo cual nos permite asegurar que cualquier mínimo local es un mínimo local. Además, exigiremos que la aplicación gradiente sea lipschitziana.

Definición 5.4 (Gradiente Lipschitziano)**.** Sea $K \subset \mathbb{R}^d$ y $f\colon K \to \mathbb{R}$ diferenciable. Decimos que la aplicación gradiente de f es *lipschitziano* con constante $L \geq 0$ si

$$\|\nabla f(x) - \nabla f(y)\| \leq L\|x - y\| \quad \forall x, y \in K$$

Lema 5.3. *Sea $K \subset \mathbb{R}^d$ convexo y $f\colon K \to \mathbb{R}$ diferenciable. Si el gradiente de f es lipschitziano con constante $L > 0$, entonces*

$$f(y) \leq f(x) + \nabla f(x)^T(y - x) + \frac{L}{2}\|y - x\|^2 \tag{5.2}$$

para cualesquiera $x, y \in K$.

Demostración. Sea $x, y \in K$. Definimos la función $g(t) = f(x + t(y - x))$ para cada $t \in [0, 1]$. Se cumple que g es derivable con

$$g'(t) = \nabla f(x + t(y - x))^T(y - x)$$

para cada $t \in [0, 1]$. Aplicando la desigualdad de Cauchy-Schwartz y la definición de gradiente lipschiztiano:

$$\begin{aligned} g'(t) - g'(0) &= \left[\nabla f(x + t(y - x)) - \nabla f(x)\right]^T (y - x) \leq \\ &\leq \|\nabla f(x + t(y - x)) - \nabla f(x)\|\|y - x\| \leq \\ &\leq Lt\|y - x\|^2 \end{aligned}$$

Aplicando el teorema fundamental del cálculo:

$$\begin{aligned} f(y) = g(1) &= g(0) + \int_0^1 g'(t)dt \leq g(0) + g'(0) + L\|y - x\|^2 \int_0^1 tdt = \\ &= f(x) + \nabla f(x)^T(y - x) + \frac{L}{2}\|y - x\|^2 \end{aligned}$$

□

Podemos probar ya la convergencia del método del gradiente descendente. Supondremos la existencia de un punto donde se alcanza el mínimo de la función f.

Proposición 5.3. *Sea $f\colon \mathbb{R}^d \to \mathbb{R}$ tal que $f \in C^1(\mathbb{R}^d)$, es convexa y su gradiente es lipschitziano con constante $L \geq 0$. Supongamos que x^* es un mínimo local de f. Sea $\eta \leq \frac{1}{L}$ y sea $x_0 \in \mathbb{R}^d$. Definimos*

$$x_k = x_{k-1} - \eta \nabla f(x_{k-1})$$

para cada $k \in \mathbb{N}$. Entonces:

$$f(x_k) - f(x^*) \leq \frac{\|x_0 - x^*\|^2}{2\eta k}$$

Demostración. La convexidad de f nos permite asegurar que x^* es un mínimo global. fijamos $x \in \mathbb{R}^d$ y llamamos $y = x - \eta \nabla f(x)$. De la desigualdad (5.2) tenemos que:

$$f(y) \leq f(x) - \eta \|\nabla f(x)\|^2 + \frac{1}{2} L \eta^2 \|\nabla f(x)\|^2$$

Usando que $\eta \leq \frac{1}{L}$, llegamos a:

$$f(y) \leq f(x) - \frac{1}{2} \eta \|\nabla f(x)\|^2 \tag{5.3}$$

En particular, $f(y) \leq f(x)$, y por tanto:

$$f(x_{k+1}) \leq f(x_k)$$

lo cual prueba que la actualización del gradiente descendente conlleva el decrecimiento de la función objetivo en cada iteración.

Usando la desigualdad (5.1), obtenemos

$$f(x) \leq f(x^*) + \nabla f(x)^T (x - x^*) \tag{5.4}$$

Combinando (5.3) y (5.4) obtenemos

$$f(y) - f(x^*) \leq \frac{1}{2\eta} \left(2\eta \nabla f(x)^T (x - x^*) - \eta^2 \|\nabla f(x)\|^2 \right)$$

Sumando y restando $\|x - x^*\|^2$ y observando:

$$\|y - x^*\|^2 = \|x - \eta \nabla f(x) - x^*\|^2 = \|x - x^*\|^2 + \eta^2 \|\nabla f(x)\|^2 - 2\eta \nabla f(x)^T (x - x^*),$$

llegamos a:

$$f(y) - f(x^*) \leq \frac{1}{2\eta} \left(\|x - x^*\|^2 - \|y - x^*\|^2 \right)$$

Para concluir la prueba aplicar la desigualdad obtenida eligiendo $x = x_{i-1}$, $y = x_{i-1} - \eta \nabla f(x_{i-1})$ para cada i de la siguiente forma:

$$\begin{aligned} k \left(f(x_k) - f(x^*) \right) &\leq \sum_{i=1}^{k} \left(f(x_i) - f(x^*) \right) \leq \sum_{i=1}^{k} \frac{1}{2\eta} \left(\|x_{i-1} - x^*\|^2 - \|x_i - x^*\|^2 \right) = \\ &= \frac{1}{2\eta} \left(\|x_0 - x^*\|^2 - \|x_k - x^*\|^2 \right) \leq \\ &\leq \frac{1}{2\eta} \left(\|x_0 - x^*\|^2 \right) \end{aligned}$$

□

El resultado que acabamos de probar nos garantiza que, si existe un punto en el que la función objetivo alcanza su valor mínimo, el algoritmo del gradiente descendente converge.

5.3.2. Variantes

Para usar el gradiente descendente hemos de elegir el criterio de parada y la tasa de aprendizaje. La proposición 5.3 pone de manifiesto la importancia de elegir un valor correcto para η. Por otra parte, elegir un criterio de parada erróneo puede hacer que el algoritmo oscile alrededor de un mínimo, obteniendo soluciones cada vez peores.

Existen algunas variantes del gradiente descendente que adaptan estos parámetros conforme avanzan las iteraciones. Se ha comprobado empíricamente que en muchos casos su aplicación conduce a mejores resultados. A continuación, describimos brevemente la idea en que se basan algunos de ellos.

- **Gradiente descendente con momento.** Es una técnica para acelerar la convergencia del gradiente descendente. Se basa en acumular en cada iteración t un vector de velocidades $v(t)$ en las direcciones en las que más está descendiendo el valor de la función objetivo:

$$v(t+1) = \mu v(t) - \eta \nabla F(w(t))$$
$$w(t+1) = w(t) + v(t+1)$$

  siendo $\eta > 0$ la tasa de aprendizaje y $\mu \in [0,1]$ el coeficiente de momento.

- **Adagrad y RMSprop.** Adagrad adapta la tasa de aprendizaje a los parámetros w. En lugar de mantener la misma para todos, a aquellos que tienen mayores derivadas parciales les corresponde una tasa más baja. Vamos a llamar $g(t) = \nabla F(w(t+1))$. Si $w \in \mathbb{R}^d$, definimos:

$$G(0) = 0$$
$$G(t+1)_i = G(t)_i + [g(t)_i]^2$$

  La regla de actualización queda:

$$w(t+1)_i = w(t)_i - \frac{\eta}{\sqrt{G(t)_i + \epsilon}} g(t)_i$$

  siendo $\eta > 0$ la tasa de aprendizaje y $\epsilon > 0$ un término pequeño para evitar la división por cero.

  El mayor inconveniente de Adagrad es que las componentes de G en cada iteración pueden crecer indefinidamente, haciendo que las actualizaciones de los parámetros no tengan efecto. RMSProp intenta solucionar esto definiendo:

$$G(t+1)_i = \alpha G(t)_i + (1-\alpha)[g(t)_i]^2$$

  para $\alpha \in [0,1]$.

- **Adam [13].** Adam es una abreviación de Adaptive Moment Estimation. Esta variante también adapta las tasas de aprendizaje para cada parámetro. Combina las ideas del gradiente descendente con momento y de RMSprop. Llamando $g(t) = \nabla F(w(t))$:

$$v(t+1) = \alpha_1 v(t) + (1-\alpha_1) g(t)$$
$$G(t+1)_i = \alpha_2 G(t)_i + (1-\alpha_2)[g(t+1)_i]^2$$

para $\alpha_1, \alpha_2 \in [0, 1]$. Podemos considerar que v y G son estimaciones del primer y segundo momento no centrado del gradiente en cada iteración. Los autores sugieren inicializarlos como $v(0) = G(0) = 0$, lo cual hace que estén sesgados entorno al cero. Proponen corregir el sesgo definiendo:

$$\hat{v}(t) = \frac{1}{1 - (\alpha_1)^t} v(t)$$
$$\hat{G}(t) = \frac{1}{1 - (\alpha_2)^t} G(t)$$

La regla de actualización queda:

$$w(t+1)_i = w(t)_i - \frac{\eta}{\sqrt{\hat{G}(t)_i} + \epsilon} \hat{v}(t)_i$$

Parte III.

Informática

6. Aprendizaje automático

En este capítulo vamos a presentar las ideas subyacentes al problema del aprendizaje automático. Pretendemos definir rigurosamente el problema del aprendizaje y presentar los resultados más importantes de la teoría de la generalización ([33], [4]). Basándonos en tales resultados explicaremos los desafíos subyacentes a todo problema de aprendizaje ([12]).

6.1. El problema del aprendizaje

Cuando decimos que un algoritmo aprende, ¿qué entendemos por aprender? Tom Mitchell proporciona la siguiente definición en [21]: "se dice que un algoritmo o programa A aprende de la experiencia E respecto a alguna tarea T y una medida de rendimiento P, si su rendimiento en la tarea T de acuerdo a P mejora con la experiencia E".

Por su parte, Vapnik-Chervonenkis expone en [33] que el problema del aprendizaje se puede plantear a través de tres componentes:

1. Un generador G de vectores aleatorios $x \in \mathcal{X}$, obtenidos de forma independiente de una distribución de probabilidad P que está fija pero es desconocida.

2. Un supervisor S que devuelve un valor $y \in \mathcal{Y}$ para cada vector de entrada x, de acuerdo a una distribución de probabilidad condicionada $P(y|x)$.

3. Una máquina de aprendizaje LM que es capaz de implementar un conjunto de funciones $h(x, \alpha)$ con $\alpha \in \Lambda$, siendo Λ un conjunto de parámetros abstractos.

El problema del aprendizaje consiste en elegir la función que mejor aproxima la respuesta del supervisor para cada vector x. Para elegir esta función, nos basamos en un conjunto de observaciones independientes e idénticamente distribuidas obtenidas de acuerdo a la distribución $P(x, y) = P(x)P(y|x)$, definida en el espacio $\mathcal{X} \times \mathcal{Y}$. Este conjunto de observaciones se llama *conjunto de entrenamiento*.

Para elegir la mejor aproximación medimos la *pérdida*, el error, o la discrepancia entre la respuesta del supervisor y la respuesta de nuestra aproximación ante un valor de entrada. Para ello recurrimos a una función ℓ y consideramos el siguiente funcional:

$$L(\alpha) = \mathbb{E}_{(X,Y)\sim P}\left[\ell(Y, f(X, \alpha))\right] = \int_{\mathcal{X}\times\mathcal{Y}} \ell(y, f(x, \alpha))\, dP(x, y) \tag{6.1}$$

Dado que la distribución de probabilidad P es desconocida y tenemos a nuestra disposición un conjunto de entrenamiento, nuestro objetivo es encontrar la función $f(x, \alpha_0)$ que minimiza el funcional anterior sobre la clase de funciones $\{f(x, \alpha) : \alpha \in \Lambda\}$.

En función del conjunto de datos del que aprende nuestro algoritmo encontramos dos grandes clases de problemas de aprendizaje:

- **Aprendizaje supervisado:** cada ejemplo está etiquetado con el resultado que nuestra función o aproximación debería de producir. Es decir, cada ejemplo de entrenamiento

se puede entender como un par (x, y) donde x es el valor que nuestro algoritmo recibe como entrada e y es el valor que debería producir.

- **Aprendizaje no supervisado:** los datos no están etiquetados. Ahora cada ejemplo de entrenamiento se compone sólo de un valor z y no conocemos la respuesta del supervisor. Este tipo de problemas no siguen el esquema descrito líneas más arriba. Ahora la distribución de probabilidad la consideramos definida en un espacio $\mathcal{Z}$ y el objetivo es minimizar:

$$L(\alpha) = \mathbb{E}_{Z\sim P}\left[\ell(f(Z,\alpha))\right] = \int_{\mathcal{Z}} \ell(f(z,\alpha))\, dP(z)$$

Podemos definir un problema de aprendizaje de forma general como sigue:

Definición 6.1 (Problema de aprendizaje)**.** Sea $\mathcal{Z} \subset \mathbb{R}^n$ un conjunto y P una medida de probabilidad sobre $\mathcal{Z}$. Sea Λ un conjunto. Sea $\ell\colon \mathcal{Z} \times \Lambda \to \mathbb{R}^+$ tal que para cada $\alpha \in \Lambda$ la función $z \mapsto \ell(z, \alpha)$ es P-integrable en $\mathcal{Z}$. Consideremos el funcional:

$$L(\alpha) = \mathbb{E}_{Z\sim P}\left[\ell(Z,\alpha)\right] = \int_{\mathcal{Z}} \ell(z,\alpha)\, dP(z), \quad \alpha \in \Lambda \tag{6.2}$$

El problema de aprendizaje consiste en minimizar el funcional (6.2) en el caso en que P es desconocida, pero se conoce una realización de una muestra aleatoria independiente e idénticamente distribuida de acuerdo a P.

6.2. Tres problemas clásicos de aprendizaje

El planteamiento que hemos hecho es muy amplio y comprende multitud de problemas. Vamos a describir a continuación los tres problemas principales del aprendizaje supervisado: clasificación binaria, regresión y estimación de una función de densidad.

6.2.1. Clasificación binaria

La clasificación binaria es un problema supervisado en el que la respuesta del supervisor y toma valores binarios, es decir, $y \in \{0, 1\}$ y para cada entrada x y cada $\alpha \in \Lambda$ se tiene que $f(x, \alpha) \in \{0, 1\}$. La función de error considerada es:

$$\ell(y, y') = \begin{cases} 0 & \text{si } y = y' \\ 1 & \text{si } y \neq y' \end{cases}$$

a la que se le suele llamar *error de clasificación*. El funcional (6.1) determina entonces la probabilidad de que la respuesta del supervisor y de nuestra solución sea diferente.

6.2.2. Regresión

Los problemas de regresión son problemas supervisados en que la respuesta del supervisor y es un valor real y $f(x, \alpha)$ es también un valor real para cada x y cada $\alpha \in \Lambda$. La función de error considerada ahora es:

$$\ell(y, y') = (y - y')^2$$

a la que se le suele llamar *error de regresión*. Minimizar (6.1) nos conduce a un problema de regresión.

6.2.3. Estimación de densidad

Ahora las funciones consideradas son funciones de densidad, cuya medida asociada determina una distribución de probabilidad. En este problema intentamos encontrar una función de densidad (x, α) cuya distribución de probabilidad asociada aproxime la distribución desconocida P. Para ello, consideramos como medida del error:

$$\ell(x, \alpha) = -\log p(x, \alpha),$$

de forma que (6.2) se convierte en la entropía cruzada. Es decir, denotando por P_α a la medida de probabilidad dada por:

$$P_\alpha(A) = \int_A p(x, \alpha)\, dP(x),$$

tenemos que $L(\alpha) = H(P, P_\alpha)$. Al minimizar el funcional L estamos minimizando la entropía cruzada, eligiendo la distribución de probabilidad P_α que mejor aproxima a P.

6.3. Minimización del riesgo empírico

La distribución P que genera el conjunto de ejemplos es desconocida. Sólo conocemos al conjunto de ejemplos, que vamos a denotar por:

$$\mathcal{D} = \{z_1, \ldots, z_N\}$$

Para minimizar el funcional (6.1) se emplea el principio inductivo llamado *minimización del riesgo empírico (ERM)*:

- Consideramos el funcional

$$L_{emp}(\alpha) = \frac{1}{N} \sum_{n=1}^{N} \ell\left(z_n, \alpha\right)$$

  construido a partir de los ejemplos de entrenamiento. A este funcional se le llama *error empírico*, *error de entrenamiento* o *error dentro de la muestra*.

- En lugar de minimizar (6.2) minimizamos el error de entrenamiento.

Sea cual sea el método o algoritmo de entrenamiento que se elija, el objetivo será minimizar el error de entrenamiento.

6.4. Teoría de la generalización

El planteamiento que hemos presentado del problema del aprendizaje usa el conjunto de entrenamiento, que es un conjunto finito de ejemplos, para aprender la función objetivo o respuesta del supervisor. Estos ejemplos de entrenamiento son sólo una pequeña parte de todos los ejemplos que pueden existir.

El lector, de forma natural, puede preguntarse por qué un conjunto limitado de ejemplos revela suficiente información para cumplir nuestro objetivo. En esta sección pretendemos responder a esta pregunta. Nuestro objetivo es demostrar que una disminución en el error de entrenamiento conlleva una disminución de la función de pérdida.

Los resultados que vamos a presentar son para el problema de clasificación binaria. Una generalización de los mismos al problema de aprendizaje general puede encontrarse en [33].

Supongamos que $\mathcal{X} \subset \mathbb{R}^n$ y $\mathcal{Y} = \{0, 1\}$. Sea $(\Omega, \mathcal{A}, \mathbb{P})$ un espacio de probabilidad y sea $\mathcal{D} = \{(x_1, y_1), \ldots, (x_N, y_N)\} \subset \mathcal{X} \times \mathcal{Y}$ el conjunto de entrenamiento. Suponemos que $\mathcal{D}$ es una realización de una muestra aleatoria simple de la variable aleatoria $(X, Y) \colon \Omega \to \mathcal{X} \times \mathcal{Y}$, cuya distribución denotamos por P. Consideramos el conjunto de funciones de hipótesis $\mathcal{H} = \{f(x, \alpha) \colon \alpha \in \Lambda\}$.

Estamos interesados en acotar la siguiente cantidad

$$|L_{emp}(\alpha) - L(\alpha)|,$$

para cualquier $\alpha \in \Lambda$, en términos del conjunto de entrenamiento, que es lo único que conocemos. A esta cantidad se le llama *error de generalización*. Si el error de generalización es bajo para un modelo determinado por α, se suele decir que $L_{emp}(\alpha)$ *generaliza bien* a $L(\alpha)$ o que el modelo *generaliza* correctamente a nuevos ejemplos.

6.4.1. Conjunto de funciones finito

En el caso en que Λ es finito podemos ofrecer una prueba sencilla del resultado. Necesitaremos la desigualdad de Hoeffding, que presentamos a continuación:

Lema 6.1 (Desigualdad de Hoeffding). *Sea $(\Omega, \mathcal{A}, \mu)$ un espacio de probabilidad. Sea $Z_1, \ldots, Z_N$ una muestra aleatoria simple de una variable aleatoria Z definida en tal espacio y que sigue una distribución de Bernouilli con media $\mathbb{E}[Z]$. Sea $\overline{Z}$ la media muestral. Sea $\epsilon \in \mathbb{R}^+$. Entonces se cumple que:*

$$\mu\left(|\mathbb{E}[Z] - \overline{Z}| > \epsilon\right) \leq 2e^{-2\epsilon^2 N} \tag{6.3}$$

Proposición 6.1. *Supongamos que $\Lambda = \{\alpha_1, \ldots, \alpha_M\}$ es finito. Sea $\epsilon > 0$. Entonces*

$$\mathbb{P}\left[\max_{\alpha \in \Lambda} |L_{emp}(\alpha) - L(\alpha)| > \epsilon\right] \leq 2M\epsilon^{-2\epsilon^2 N} \tag{6.4}$$

Demostración. Fijemos $\alpha_k \in \Lambda$ y definamos $I : \mathcal{X} \times \mathcal{Y} \to \{0, 1\}$ como

$$I(x, y) = \begin{cases} 1, & \text{si } f(x, \alpha_k) = y, \\ 0, & \text{si } f(x, \alpha_k) \neq y \end{cases}$$

Si $(X, Y) \sim P$, entonces $I(X, Y)$ es una variable aleatoria definida en $(\Omega, \mathcal{A}, \mathbb{P})$ que sigue una distribución de Bernouilli. Escribiendo $I_n = I(x_n, y_n)$ se tiene que $I_1, \ldots, I_n$ es una muestra aleatoria simple de (X, Y). Fijamos $\epsilon > 0$ y podemos aplicar (6.3) para obtener

$$\mathbb{P}[|L(\alpha_k) - L_{emp}(\alpha_k)| > \epsilon] \leq 2\epsilon^{-2\epsilon^2 N} \tag{6.5}$$

Denotemos:

$$B = \left\{ \max_{\alpha \in \Lambda} |L(\alpha) - L_{emp}(\alpha)| > \epsilon \right\}$$
$$B_m = \left\{ |L(\alpha_m) - L_{emp}(\alpha_m)| > \epsilon \right\}$$

Se cumple que $B \subset \bigcup_{m=1}^{M} B_m$, por lo que:

$$\mathbb{P}[B] \leq \mathbb{P}[\bigcup_{m=1}^{M} B_m] \leq \sum_{m=1}^{M} \mathbb{P}[B_m]$$

Podemos aplicar la desigualdad (6.5) a cada término para obtener:

$$\mathbb{P}[|L(\alpha) - L_{emp}(\alpha)| > \epsilon] \leq 2M\epsilon^{-2\epsilon^2 N}$$

□

La desigualdad (6.4) nos dice que, fijado un margen de error ϵ, la diferencia entre L_{emp} y L queda controlada por el tamaño de la muestra. Aumentando el tamaño de N, la probabilidad de exceder el margen de error tiende a cero.

Reescribimos la última desigualdad. Fijamos $\delta \in]0,1[$ y tomamos $\epsilon = \sqrt{\frac{1}{2N} log\left(\frac{2M}{\delta}\right)}$. Aplicando el resultado anterior obtenemos que con probabilidad al menos $1 - \delta$,

$$L(\alpha) \leq L_{emp}(\alpha) + \sqrt{\frac{1}{2N} log\left(\frac{2M}{\delta}\right)}, \quad \forall \alpha \in \Lambda$$

Esta última desigualdad nos informa de que fijada una tolerancia δ el error L queda dominado por el error que se comete en el conjunto de entrenamiento, L_{emp}, más un término positivo. Lo importante es que este término positivo converge a cero cuando el tamaño de la muestra aumenta.

6.4.2. Dimensión de Vapnik-Chervonenkis

En la práctica, el conjunto Λ será infito. Para dar una respuesta en este caso presentamos las ideas básicas de la teoría de la *dimensión de Vapnik-Chervonenkis*. Llamamos $\mathcal{H} = \{f(x,\alpha) \colon \alpha \in \Lambda\}$.

Definición 6.2 (Dicotomías generadas por $\mathcal{H}$)**.** Sean $x_1, \ldots, x_N \in \mathcal{X}$. Llamamos *dicotomías generadas por* $\mathcal{H}$ *en* $x_1, \ldots, x_N$ al conjunto:

$$\mathcal{H}(x_1, \ldots, x_N) = \{(f(x_1,\alpha), \ldots, f(x_N,\alpha)) \colon \alpha \in \Lambda\} \subset \{0,1\}^N$$

Definición 6.3 (Función de crecimiento)**.** Definimos la función de crecimiento de $\mathcal{H}$ como

$$m_{\mathcal{H}} \colon \mathbb{N} \to \mathbb{R}$$
$$N \mapsto \max_{x_1,\ldots,x_N \in \mathcal{X}} |\mathcal{H}(x_1, \ldots, x_N)|$$

donde $|A|$ denota el cardinal de A.

Como $\mathcal{H}(x_1, \ldots, x_N) \subset \{0,1\}^N$, entonces $m_{\mathcal{H}}(N) \leq 2^N$ para cada $N \in \mathbb{N}$. Si para algún natural k ocurre que $m_{\mathcal{H}}(k) < 2^k$, podemos acotar la función de crecimiento por un polinomio de grado $k-1$:

Teorema 6.1. *Si $m_{\mathcal{H}}(k) < 2^k$ para algún número natural k, entonces*

$$m_{\mathcal{H}}(N) \leq \sum_{i=0}^{k-1} \binom{N}{i} \quad \forall N \in \mathbb{N}$$

Los valores de k para los cuales ocurre lo anterior reciben un nombre especial. Si $m_{\mathcal{H}}(k) < 2^k$ decimos que k es un *punto de ruptura*. En el momento en que identifiquemos un punto de ruptura, conseguimos acotar $m_{\mathcal{H}}(N)$ por un polinomio de grado $k-1$. Cuanto más pequeño sea el punto de ruptura, mejor será la cota que obtengamos.

Así surge el concepto que definimos a continuación: la *dimensión de Vapnik-Chervonenkis* nos permite medir cómo de grande es el conjunto de hipótesis. Intuitivamente, es el máximo número d de ejemplos que pueden ser separados en dos clases de todas las 2^d formas posibles, usando sólo funciones del conjunto de hipótesis.

Definición 6.4 (Dimensión de Vapnik-Chervonenkis)**.** La dimensión de Vapnik-Chervonenkis o dimensión VC de $\mathcal{H}$, denotada por $d_{VC}(\mathcal{H})$, es:

$$d_{VC}(\mathcal{H}) = \max\{N \in \mathbb{N} \colon m_{\mathcal{H}}(N) = 2^N\}$$

entendiendo que si $m_{\mathcal{H}}(N) = 2^N$ para cada $N \in \mathbb{N}$, entonces $d_{VC}(\mathcal{H}) = \infty$.

Si $d_{VC} < \infty$ entonces $d_{VC} + 1$ es un punto de ruptura. Del Teorema 6.1 deducimos que si $d_{VC} < \infty$, entonces

$$m_{\mathcal{H}}(N) \leq N^{d_{VC}} + 1.$$

El resultado central de esta teoría se enuncia como sigue:

Teorema 6.2 (Vapnik, Chervonenkis, 1971)**.** *Para cada $\epsilon > 0$ se cumple:*

$$\mathbb{P}\left[\sup_{\alpha \in \Lambda} |L_{emp}(\alpha) - L(\alpha)| > \epsilon\right] \leq 4 m_{\mathcal{H}}(2N) \exp\left(-\frac{1}{8}\epsilon^2 N\right)$$

Si $d_{VC}(\mathcal{H})$ es finita, entonces podemos acotar la función de crecimiento por un polinomio. Al tomar límite cuando N tiende a infinito, la diferencia entre L y L_{emp} se hace cero, para cualquier función del conjunto de hipótesis. Esto significa que, con un tamaño de muestra lo suficiente grande, cualquier función $f(x, \alpha)$ en un conjunto con dimensión VC finita obtendrá un error de generalización bajo.

Este resultado es uno de los más importantes en la teoría del aprendizaje estadístico. Establece que el aprendizaje es factible cuando el conjunto de hipótesis es infinito pero su dimensión VC es finita.

Corolario 6.1. *Sea $\delta > 0$. Entonces para cada $\alpha \in \Lambda$*

$$L(\alpha) \leq L_{emp}(\alpha) + \sqrt{\frac{8}{N} log\left(\frac{4 m_{\mathcal{H}}(2N)}{\delta}\right)} \tag{6.6}$$

con probabilidad al menos $1 - \delta$.

6.5. Equilibrio entre complejidad y generalización

6.5.1. Complejidad de $\mathcal{H}$

Escribamos

$$\Gamma(N, \mathcal{H}, \delta) = \sqrt{\frac{8}{N} log\left(\frac{4m_{\mathcal{H}}(2N)}{\delta}\right)},$$

por lo que la desigualdad (6.6) se reescribe como

$$L(\alpha) \leq L_{emp}(\alpha) + \Gamma(N, \mathcal{H}, \delta)$$

Recordemos que, en la práctica, el conjunto de entrenamiento es fijo, y por tanto también lo es N. Cuanto más grande es d_{VC} más grande se hace $\Gamma(N, \mathcal{H}, \delta)$. Si entendemos que d_{VC} es una medida de la complejidad o de la dimensión de $\mathcal{H}$, podemos entender que $\Gamma(N, \mathcal{H}, \delta)$ es una penalización por la complejidad del modelo. Obsérvese, sin embargo, que cuanto más compleja sea la clase de funciones, más posibilidades tendremos de encontrar una que obtenga un error de entrenamiento bajo. Este es el primer compromiso que presenta la teoría de la generalización:

1. La clase $\mathcal{H}$ debe ser lo suficientemente compleja para que $L_{emp} \approx 0$.
2. Si la clase $\mathcal{H}$ es demasiado compleja, el error de generalización será alto, por lo que L no estará cerca de cero.

En la Figura 6.1 se representan estas ideas. La curva azul se corresponde con L, la roja con la idea de complejidad de la clase de hipótesis y la verde con L_{emp}. Obsérvese que al aumentar la complejidad de $\mathcal{H}$, disminuye el error de entrenamiento, pero el error fuera de la muestra L aumenta.

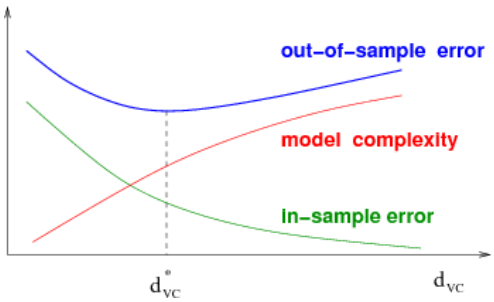


Figura 6.1.: Compromiso entre complejidad y generalización ([4]).

6.5.2. Sesgo y varianza

Basándonos en la teoría de Vapnik-Chervonenkis hemos detectado un fenómeno que en inglés de denomina *approximation-generalization tradeoff*. A continuación vamos a detectar este mismo fenómeno desde un punto de vista totalmente distinto y sin restringirnos al problema de clasificación binaria, usando un razonamiento típico de la inferencia estadística. Estudiamos qué ocurre si, en lugar de considerar a $\mathcal{D}$ como un conjunto fijo de ejemplos, pensamos en él como una muestra aleatoria simple que puede variar.

Sean $\mathcal{X} \subset \mathbb{R}^n$, $\mathcal{Y} \subset \mathbb{R}^m$. Consideremos un vector aleatorio (X, Y) en $\mathcal{X} \times \mathcal{Y}$ y denotemos por P a su distribución de probabilidad. El conjunto de entrenamiento, $\mathcal{D}$, que en la práctica está formado por las observaciones de una muestra aleatoria simple, a continuación será considerado como un vector aleatorio de vectores aleatorios independientes. Es decir:

$$\mathcal{D} = [(X_1, Y_1), \ldots, (X_N, Y_N)]$$

donde cada $(X_i, Y_i) \sim P$ y (X_i, Y_i) y (X_j, Y_j) son independientes para cada i, j. Por tanto $\mathcal{D} \sim P^N$, siendo P^N la medida de probabilidad producto en el espacio $(\mathcal{X} \times \mathcal{Y})^N$.

Vamos a suponer que existe una relación funcional entre X e Y, es decir, existe $S\colon \mathcal{X} \to \mathcal{Y}$ sobreyectiva, por lo que siempre podemos escribir $Y = S(X)$.

Como resultado del proceso de entrenamiento sobre una realización del conjunto de entrenamiento, nuestro método de entrenamiento obtiene una función f. Esta será dependiente de la realización de $\mathcal{D}$ y asignará a cada ejemplo x un valor que intenta aproximar a $S(x)$:

$$\begin{aligned} f\colon (\mathcal{X} \times \mathcal{Y})^N \times \mathcal{X} &\to \mathcal{Y} \\ (d, x) &\mapsto f(d, x) \end{aligned}$$

Entonces $f(\mathcal{D}, x)$ es un vector aleatorio para cada $x \in \mathcal{X}$, por lo que podemos calcular su esperanza $\overline{f}(x) = \mathbb{E}_{\mathcal{D}}[f(\mathcal{D}, x)]$. Se puede entender que $\overline{f}$ es la función "media" que se obtiene tras promediar el resultado del entrenamiento sobre infinitos conjuntos de datos generados de acuerdo a P.

Fijado $\mathcal{D}$, el error esperado al aproximar S por f usando el conjunto $\mathcal{D}$ viene dado por:

$$ECM(\mathcal{D}) = \mathbb{E}_X\left[\|f(\mathcal{D}, X) - S(X)\|^2\right]$$

Esta medida depende de la realización de $\mathcal{D}$ que dispongamos. El error esperado de nuestro modelo de entrenamiento, independiente de cualquier realización de $\mathcal{D}$, viene dada por:

$$\mathbb{E}_{\mathcal{D}}[ECM(\mathcal{D})]$$

Obsérvese que $\mathbb{E}_{\mathcal{D}}[ECM(\mathcal{D})]$ se puede corresponder a la medida de riesgo L utilizada en un problema de regresión. Usando la linealidad de la esperanza, es fácil probar la siguiente igualdad:

$$\mathbb{E}_{\mathcal{D}}[ECM(\mathcal{D})] = \mathbb{E}_X\left[\mathbb{E}_{\mathcal{D}}\left[\left\|f(\mathcal{D}, X) - \overline{f}(X)\right\|^2\right] + \left\|\overline{f}(\mathcal{D}, X) - S(X)\right\|^2\right]$$

Analicemos los dos sumandos presentes en el miembro de la derecha

1. El primer sumando se corresponde con la varianza del vector aleatorio $f(\mathcal{D}, X)$:

   $$Var(x) = \mathbb{E}_{\mathcal{D}}\left[\left\|f(\mathcal{D}, x) - \overline{f}(x)\right\|^2\right]$$

   el cual mide la variación que sufre la hipótesis elegida cuando varía el conjunto de entrenamiento.

2. El segundo sumando, que se denomina sesgo, mide cuánto se desvía $\overline{f}$ de la función a

aproximar:

$$Bias(x) = \left\| \overline{f}(\mathcal{D}, x) - S(x) \right\|^2$$

De esta forma, podemos escribir

$$\mathbb{E}_{\mathcal{D}}\left[ECM(\mathcal{D})\right] = \mathbb{E}_X\left[Var(X) + Bias(X)\right] = Var + Bias$$

donde $Var = \mathbb{E}_X\left[Var(X)\right]$ y $Bias = \mathbb{E}_X\left[Bias(X)\right]$. Esta igualdad recibe el nombre de *descomposición en sesgo y varianza* del error cuadrático medio. Como advertíamos, este nombre proviene de la inferencia estadística, ya que es una igualdad análoga al error cuadrático medio de un estimador.

El mismo fenómeno que observábamos usando la desigualdad de Vapnik-Chervonenkis sobre la complejidad de $\mathcal{H}$ se refleja, de forma intuitiva, en la descomposición en sesgo y varianza. Fijemos un método de aprendizaje, es decir, una forma de elegir una función de $\mathcal{H}$ minimizando el riesgo empírico.

- Supongamos que $\mathcal{H} = \{g\}$. En tal caso, $f(\mathcal{D}, x) = \overline{f}(x) = g(x)$, por lo que $Var = 0$. Sin embargo, el sesgo será muy alto a no ser que "g se parezca mucho a S". Intuitivamente, una clase de hipótesis con una complejidad baja conlleva una varianza baja, pero un sesgo muy alto.

- Por otra parte, si $\mathcal{H}$ es muy compleja el sesgo será muy bajo, incluso cero si $S \in \mathcal{H}$. Pero en ese caso la variabilidad que se producirá será muy alta, pues pequeños cambios en el conjunto de entrenamiento alterarán el resultado del proceso de entrenamiento.

Sin embargo, hay un detalle en la descomposición en sesgo y varianza que no estaba presente en la dimensión VC. El método, técnica o algoritmo de aprendizaje es el que se encarga de elegir la función correcta de la clase de hipótesis. Puede ocurrir que, usando el mismo conjunto de hipótesis, dos algoritmos presenten varianzas y sesgos muy distintos.

Cabe resaltar que, en la práctica, nunca podremos tener acceso a tal descomposición del error, ya que sólo contamos con un conjunto finito de ejemplos de entrenamiento.

En la siguiente sección se presenta la regularización, la cual nos permitirá modificar el método de aprendizaje para encontrar la mejor aproximación a la respuesta del supervisor dentro de clases muy complejas.

6.6. Sobrejuste y regularización

En la sección anterior hemos relatado cómo influye la complejidad de una clase en el error de generalización y la necesidad de desarrollar una técnica para guiar el entrenamiento en clases de hipótesis complejas. Podemos decir que el objetivo de toda técnica de entrenamiento es doble:

1. Conseguir que el error de entrenamiento, L_{emp}, sea bajo.

2. Conseguir que el error de generalización, $|L_{emp} - L|$, sea bajo. Cuando L_{emp} ya es bajo, esto es equivalente a que L sea alto.

Decimos que se produce *sobreajuste* o *overfitting* cuando el error de generalización es alto. Obsérvese que si no controlamos la complejidad de la clase de hipótesis caeremos en

sobreajuste. Si somos demasiados restrictivos con esta complejidad, no conseguiremos ni siquiera que L_{emp} sea bajo.

Desarrollar técnicas para evitar el sobreajuste es uno de los problemas centrales en aprendizaje automático. Además, el propósito de minimizar tanto el error de generalización como el error de entrenamiento es lo que diferencia al aprendizaje automático de un problema de optimización.

La *regularización* es una de las herramientas más populares para lidiar con el sobreajuste. Intuitivamente, las técnicas de regularización restringen la capacidad o complejidad de la clase de hipótesis con el objetivo de reducir el error de regularización pero no el error de entrenamiento. La regularización actúa de forma directa sobre el método o algoritmo de aprendizaje.

Una forma de hacer esto es añadir un término de penalización a L_{emp} y minimizar la nueva función obtenida. Este término se denomina *regularizador*. De esta forma, la nueva función a minimizar es

$$J(\alpha) = L_{emp}(\alpha) + R(\alpha), \quad \forall \alpha \in \Lambda$$

donde $R\colon \Lambda \to \mathbb{R}$ es el término regularizador.

Las técnicas de regularización pretenden actuar sobre la dimensión VC del modelo. Se espera que, minimizando la función R en lugar de L_{emp} el modelo de funciones considerado tenga una dimensión menor, por lo que el error de generalización puede descender.

Ejemplo 6.1 (Weight decay)**.** Un método de regularización popular es *weight decay* o decaimiento de pesos en español. Cuando $\Lambda \subset \mathbb{R}^d$, se toma

$$R(\alpha) = \lambda \alpha^T \alpha$$

para cierto $\lambda \in \mathbb{R}^+$. Sean $C_1, C_2 \in \mathbb{R}^+$, y sea

$$\mathcal{H}_{C_k} = \left\{ f(x, \alpha) \colon \alpha \in \Lambda, \alpha^T \alpha \leq C_k \right\}$$

Si $C_1 < C_2$ entonces $\mathcal{H}_{C_1} \subset \mathcal{H}_{C_2}$, por lo que $d_{VC}(\mathcal{H}_{C_1}) \leq d_{VC}(\mathcal{H}_{C_2})$.

Supongamos ahora que $\Lambda = \mathbb{R}^d$ y que L_{emp} es diferenciable y alcanza un mínimo en un punto donde el gradiente no se anula. Una sencilla aplicación del teorema de los multiplicadores de Lagrange nos permite afirmar que, para cada $C \in \mathbb{R}^+$, existe $\lambda_C \in \mathbb{R}^+$ tal que el problema de optimización con restricciones

$$\min_{\alpha \in \Lambda} L_{emp}(\alpha) \quad \text{sujeto a} \quad \alpha^T \alpha \leq C$$

es equivalente a

$$\min_{\alpha \in \Lambda} L_{emp}(\alpha) + \lambda_C \alpha^T \alpha$$

6.7. Gradiente descendente: aplicación en el aprendizaje

El aprendizaje a partir de un conjunto de datos se efectúa minimizando una función J que será, en algunos casos, una variación del riesgo empírico.

$$\text{Encontrar } \alpha^* \in \Lambda \text{ tal que } J(\alpha^*) = \min_{\alpha \in \Lambda} J(\alpha)$$

Debemos tener en cuenta que en la mayoría de ocasiones el conjunto de parámetros es un subconjunto del espacio euclídeo $\Lambda \subset \mathbb{R}^d$. Además, la función $\alpha \mapsto J(\alpha)$ suele ser diferenciable, por lo que podemos aplicar el algoritmo de gradiente descendente para resolver nuestro problema.

Para poder entrenar cualquier modelo de aprendizaje automático, bastará con elegir una función de pérdida J conveniente y que seamos capaces de calcular los gradientes de tal función. Esto requerirá que la función que implemente nuestro modelo sea diferenciable respecto de los parámetros que la definen, con el fin de aplicar la regla de la cadena. Veremos un ejemplo de esto cuando calculemos los gradientes de una red neuronal en el capítulo 2.

La función a minimizar tendrá la siguiente forma:

$$J(\alpha) = L_{emp}(\alpha) + R(\alpha) = \frac{1}{N}\sum_{n=1}^{N} \ell\left(z_n, \alpha\right) + R(\alpha)$$

Si suponemos que R es diferenciable y que $\alpha \mapsto \ell(z, \alpha)$ es diferenciable para cada z, entonces el gradiente se calcula como:

$$\nabla L_{emp}(\alpha) = \frac{1}{N}\sum_{n=1}^{N} \nabla_\alpha \ell\left(z_n, \alpha\right) + \nabla J(\alpha)$$

Para que el modelo de aprendizaje obtenido generalice correctamente a nuevos ejemplos el conjunto de entrenamiento deberá ser grande. Sin embargo, cuanto más ejemplos entren en juego, más costoso será calcular el gradiente. Recordemos que, en la práctica, los ejemplos de entrenamiento pueden superar los cientos de miles y que la evaluación de los gradientes también tiene un coste para nada despreciable. Por tanto, no es admisible utilizar todos los ejemplos de entrenamiento en cada actualización del gradiente.

Así surge una variación del gradiente descendente original llamada *gradiente descendente estocástico (SGD)*. La idea fundamental es que ∇L_{emp} es un estimador de la esperanza de ∇L, que a su vez se puede aproximar con un número menor de ejemplos. En cada iteración se elige un subconjunto $\{z_1, \ldots, x_M\} \subset \mathcal{D}$ con M significativamente más pequeño que N. A este subconjunto se le llama *minibatch*. El cálculo del gradiente se efectúa usando sólo los ejemplos del minibatch.

Algoritmo 2: Gradiente descendente estocástico.

Entrada: Función ℓ, tasa de aprendizaje $\eta \in \mathbb{R}^+$, conjunto $\mathcal{D}$
Inicializar $\alpha(0)$
para $t = 0, 1, 2, \ldots$ **hacer**
  Escoger un minibatch $\{z_1, \ldots, z_M\}$
  Calcular $g(t) = \frac{1}{M}\sum_{m=1}^{M} \nabla_\alpha \ell(z_m, \alpha(t)) + \nabla J(\alpha)$
  $\alpha(t+1) = \alpha(t) - \eta g(t)$
fin

En la práctica, los modelos de aprendizaje automático obtienen muy buenos resultados cuando son entrenados usando gradiente descendente y sus variantes. Evidentemente, no siempre las funciones objetivo cumplen las hipótesis exigidas en el estudio de la convergencia que hemos realizado en el Capítulo 5, pero aún así se consiguen óptimos locales en un tiempo razonable.

7. Aprendizaje profundo

El aprendizaje profundo, o *deep learning* en inglés, es el área del aprendizaje que estudia modelos que emplean una representación jerárquica de la información. Estos modelos aprenden conceptos complejos a partir de otros más simples. Para ello se usan las llamadas *redes neuronales*, que se han hecho muy populares en la última década y es un área de investigación muy activa.

En este capítulo introduciremos formalmente el concepto de red neuronal y su proceso de entrenamiento. A continuación, presentaremos los modelos de aprendizaje profundo que sientan la base para el problema que enfrentamos en este trabajo.

7.1. Redes neuronales prealimentadas

Las redes neuronales prealimentadas implementan una composición de funciones. Podemos representar una red neuronal como un grafo dirigido acíclico con los nodos divididos en capas. Cada nodo se llama neurona, recibe información de las neuronas de la capa anterior y calcula un valor de salida.

Empecemos con la definición formal de neurona:

Definición 7.1 (Neurona)**.** Sea $w \in \mathbb{R}^d$ y $\theta\colon \mathbb{R}^d \to \mathbb{R}$ una función que llamaremos función de activación. Una neurona calcula la función $h\colon \mathbb{R}^d \to \mathbb{R}$ dada por

$$h(x) = \theta\left(\sum_{i=1}^{d} w_i x_i\right), \quad \forall x \in \mathbb{R}^d$$

Vemos que el funcionamiento de una neurona es muy simple. Como entrada toma valores reales y calcula una transformación lineal de los mismos. A continuación, aplica una función de activación para producir el valor de salida. Veremos que el poder de las redes neuronales radica, en parte, en la elección de esta función de activación.

En las redes neuronales prealimentadas las salidas de las neuronas de una capa sirven de entrada a las neuronas de la capa superior. Cuando construimos una red debemos elegir el número de capas, el número de neuronas de cada capa y la función de activación de cada neurona.

Definición 7.2 (Red neuronal prealimentada)**.** Sea $T \in \mathbb{N}$ y el vector $d = (d_0, \dots, d_T) \in \mathbb{N}^T$. Sea $W = (W_1, \dots, W_T)$ con $W_t \in \mathbb{R}^{d_t \times d_{t-1}}$. Sea el conjunto de funciones $\{\theta_t\colon t \in \{1, \dots, T\}\}$ con $\theta_t \in C^1(\mathbb{R}^{d_{t-1}}, \mathbb{R}^{d_t})$. Consideramos:

$$\begin{cases} x_0 = x \in \mathbb{R}^{d_0}, \\ x_t = \theta_t(W_t x_{t-1}), \quad t \in \{1, \dots, T\} \end{cases}$$

Diremos que la red neuronal $[L, d, W, \{\theta_t\}]$ implementa la función $h(x, W) = x_T$.

En la siguiente observación recogemos información importante sobre esta definición:

Observación 7.1.

1. Diremos que T es el número de capas de la red. El número de neuronas de la capa t se corresponde con d_t y también se suele llamar dimensión de la capa t.

2. La matriz W_t es la matriz de pesos de la capa t. Estos serán los parámetros del modelo que el algoritmo de gradiente descendente modificará para minimizar la función de pérdida. Hemos decidido recoger los parámetros en forma de matriz pero también podríamos haberlo hecho en forma de vector: si $W_t = \left[w_{ij}\right]$, entonces consideramos el vector $w_t \in$ que está formado por las filas de W_t concatenadas:

$$w_t = \left[w_{11}, \ldots, w_{1d_{t-1}}, \ldots, w_{d_t 1}, \ldots, w_{d_t d_{t-1}}\right]$$

   Se cumple:

$$W_t x_{t-1} = X_{t-1} w_t$$

   donde $X_{t-1} \in \mathbb{R}^{d_t \times d_{t-1} d_t}$ se define como sigue:

$$X_{t-1} = \begin{bmatrix} x_{t-1} & 0 & \cdots & 0 \\ 0 & x_{t-1} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \ldots & x_{t-1} \end{bmatrix}$$

3. θ_t es la función de activación de la capa t. Cada una se puede escribir como $\theta_t = \left[\theta_{t,1} \ldots, \theta_{t,d_t}\right]$, de forma que θ_t aglutina las funciones de activación de cada neurona de la capa t.

4. La función que la red implementa en cada capa es $f_t(u) = \theta_t(W_t u)$. Escribiremos $s_t(u) = W_t u$, y por tanto $f_t = \theta_t \circ s_t$. De esta forma, $x_T = (f_T \circ f_{T-1} \circ \cdots \circ f_1)(x)$, y se escribe $h = f_T \circ f_{T-1} \circ \cdots \circ f_1$. Vemos como, efectivamente, una red neuronal prealimentada implementa una composición de funciones.

Expresamos el espacio de hipótesis de las redes neuronales prealimentadas de la siguiente forma:

$$\mathcal{H}_{NN} = \{h(x, W) \colon [L, d, W, \{\theta_t\}] \text{ implementa la función } h(x, W) = x_T\}$$

Señalamos ahora dos ejemplos clásicos en aprendizaje automático que se engloban en la definición de red neuronal prealimentada.

Ejemplo 7.1 (Perceptrón)**.** Tomando $T = 1$, $d \in \mathbb{N}$, $w \in \mathbb{R}^d$, y la función de activación

$$\theta(s) = \begin{cases} 1, \text{ si } s \geq 0, \\ 0, \text{ si } s < 0 \end{cases} \qquad \forall s \in \mathbb{R}$$

obtenemos el llamado modelo perceptrón:

$$h(x, w) = \theta\left(\sum_{i=1}^{d} w_i x_i\right), \quad \forall x \in \mathbb{R}^d$$

El modelo perceptrón supone una aproximación clásica al problema de clasificación binario. Notemos que $P = \left\{ x \in \mathbb{R}^d \colon \sum_{i=1}^{d} w_i x_i = 0 \right\}$ es un hiperplano en $\mathbb{R}^d$, por lo que $\mathbb{R}^d \setminus P$ tiene dos componentes conexas. La función h asigna cada ejemplo a una componente conexa y bajo la hipótesis de que los ejemplos de entrenamiento sean linealmente separables, el algoritmo perceptrón siempre encuentra el hiperplano que los separa. Este algoritmo y la prueba de su convergencia pueden encontrarse en [23].

Ejemplo 7.2 (Regresión logística)**.** Tomando $T = 1$, $d \in \mathbb{N}$, $w \in \mathbb{R}^d$, y la función de activación

$$\sigma(s) = \frac{e^s}{1 + e^s}, \quad \forall s \in \mathbb{R}$$

obtenemos el modelo de regresión logística:

$$h(x, w) = \sigma\left(\sum_{i=1}^{d} w_i x_i \right), \quad \forall x \in \mathbb{R}^d$$

La función σ toma imagen en el intervalo $[0, 1]$. Este método se emplea en problemas de clasificación binaria ya que la función h puede interpretarse como la probabilidad de que la etiqueta de un ejemplo sea 1.

7.2. Entrenamiento de redes neuronales

Para entrenar cualquier modelo de aprendizaje automático, debemos de ser capaces de calcular la función que implementa el modelo y los gradientes de la función de pérdida respecto a los parámetros del mismo. En el caso de las redes neuronales prealimentadas, veremos que estos dos procesos están estrechamente relacionados.

7.2.1. Propagación hacia delante

Propagación hacia delante, o *forward propagation* en inglés, es el nombre que recibe el algoritmo que se encarga de calcular la función que implementa una red neuronal prealimentada. Su nombre se debe a que, en el proceso de inferencia, en una red neuronal la información fluye desde la primera capa hacia las capas superiores.

La forma en que hemos definido la red neuronal nos sugiere la forma en que debemos calcular la salida, por lo que sólo nos queda describirlo sistemáticamente.

Algoritmo 3: Propagación hacia delante (FP).

Datos: Red neuronal prealimentada $[L, d, W, \{\theta_t\}]$
Entrada: $x \in \mathbb{R}^{d_0}$
$x_0 = x$
para $t = 1, \ldots, T$ **hacer**
  $s_t = W_t x_{t-1}$
  $x_t = \theta_t(s_t)$
fin
Salida: $(x_1, \ldots, x_T), (s_1, \ldots s_T)$

Cuando calculamos la salida de la red debemos calcular los vectores s_t. Esta es una información que no debemos olvidar en el cálculo de los gradientes.

7.2.2. Propagación hacia atrás

Para que la red neuronal aprenda mediante el algoritmo de gradiente descendente debemos ser capaces de calcular los gradientes de la función de pérdida. Es decir, si consideramos la función $W \mapsto \ell(h(x, W), y)$ para $(x, y) \in \mathcal{D}$ fijo, buscamos calcular los gradientes de la misma respecto de las matrices de pesos, es decir:

$$\nabla_W \ell = \left[\nabla_{W_1} \ell, \ldots, \nabla_{W_T} \ell\right]$$

Como vemos, los gradientes se efectúan respecto a una matriz. Este es un concepto que debemos aclarar. Podemos ver a $\nabla_{W_t} \ell$ como una matriz o un vector. Escribamos $W_t = \left[w_{ij}\right]$.

- Si entendemos que es una matriz, entonces $\nabla_{W_t} \ell = \left[\frac{\partial \ell}{\partial w_{ij}^t}\right]$.
- Para poder entenderlo como un vector, debemos recordar que W_t se correspondía con el vector w_t. En ese caso, $\nabla_{W_t} \ell = \nabla_{w_t} \ell$, donde el último gradiente es en el sentido convencional.

Nuestro razonamiento se basa en aplicar la regla de la cadena *recursivamente* a funciones que supondremos diferenciables. Por ello, necesitamos recordar algunos conceptos sobre diferenciabilidad a la vez que fijamos la notación.

Definición 7.3 (Matriz jacobiana)**.** Si $F\colon \mathbb{R}^n \to \mathbb{R}^m$ es diferenciable en $p \in \mathbb{R}^n$, la diferencial de F en p es una aplicación lineal que denotaremos por dF_p. La matriz de esta aplicación en la base usual se denomina matriz jacobiana y la denotaremos en lo que sigue por $\mathrm{J}F_p$. Si $F = (F_1, \ldots, F_m)$

$$\mathrm{J}F_p = \left[\frac{\partial F_i}{\partial x_j}(p)\right] \in \mathbb{R}^{m \times n}$$

Cuando $m = 1$ tenemos que $\mathrm{J}F_p = \nabla F(p)$.

Teorema 7.1 (Regla de la cadena)**.** *Si $F\colon \mathbb{R}^n \to \mathbb{R}^m$ es diferenciable en $p \in \mathbb{R}^n$ y $G\colon \mathbb{R}^m \to \mathbb{R}^k$ es diferenciable en $F(p)$, entonces $H = G \circ F$ es diferenciable y se cumple:*

$$dH_p = dG_{F(p)} \circ dF_p$$

de donde se deduce que $\mathrm{J}H_p = \mathrm{J}G_{F(p)}\mathrm{J}F_p$.

Supongamos que $\ell\colon \mathbb{R}^{d_T} \to \mathbb{R}$ es una función diferenciable. Para cada t podemos considerar la subred formada por las capas $t, t+1, \ldots, T$ como una red neuronal y definir la función $\ell_t\colon \mathbb{R}^{d_t} \to \mathbb{R}$ asociada a la misma:

$$\ell_t(u) = \ell\left((f_T \circ \cdots \circ f_t)(u)\right)$$

A partir de estas funciones definimos

$$\delta_T = (\mathrm{J}\ell)_{x_T} = \nabla \ell(x_T)$$
$$\delta_t = (\mathrm{J}\ell_{t+1})_{x_t} = \nabla \ell_{t+1}(x_t)$$

El siguiente resultado pone de manifiesto la importancia de estos elementos en el cálculo del gradiente:

Proposición 7.1. *Sea $[L, d, W, \{\theta_t\}]$ una red neuronal. Sea $\ell\colon \mathbb{R}^{d_T} \to \mathbb{R}$ una función diferenciable. Supongamos que las funciones de activación $\{\theta_t\}$ son también diferenciables. Si $x \in \mathbb{R}^{d_0}$, entonces*

$$\nabla_{w_t} \ell(h(x, W)) = \delta_t (\mathrm{J}\theta_t)_{s_t} X_{t-1} \tag{7.1}$$

y además se cumple

$$\delta_t = \delta_{t+1} (\mathrm{J}\theta_t)_{s_t} W_t \tag{7.2}$$

Demostración. Empezamos observando que existe una relación útil entre las funciones asociadas a las subredes que se deduce de la propia definición:

$$\ell_t(u) = \ell_{t+1}(\theta_t(W_t u))$$

Aplicando la regla de la cadena a esta igualdad se obtiene:

$$(\mathrm{J}\ell_t)_{x_{t-1}} = (\mathrm{J}\ell_{t+1})_{x_t} (\mathrm{J}\theta_t)_{s_t} W_t$$

que prueba (7.2). Para obtener (7.1) consideramos, para cada t, la función $g_t\colon \mathbb{R}^{d_t d_{t-1}} \to \mathbb{R}$ dada por

$$g_t(w_t) = \ell_{t+1}(\theta_t(X_{t-1} w_t))$$

entendiendo que $\ell_{t+1} = \ell$. Nótese que $g_t(w_t) = \ell(h(x, W))$ y por tanto $(\mathrm{J}g_t)_{w_t} = \nabla g_t(w_t) = \nabla_{w_t} \ell(h(x, W))$. Aplicando la regla de la cadena en la definición de g_t obtenemos

$$(\mathrm{J}g_t)_{w_t} = (\mathrm{J}\ell_{t+1})_{x_t} (\mathrm{J}\theta_t)_{s_t} X_{t-1}$$

lo cual concluye la prueba. □

El resultado que acabamos de probar nos permite calcular el gradiente de cualquier función de pérdida que sea diferenciable, bajo la hipótesis de que existan las derivadas parciales de las funciones de activación. Es decir, debemos elegir funciones que sean diferenciables o bien continuas y diferenciables en casi todo punto. La clave está en que podemos calcular δ_t aplicando la ecuación (7.2) desde la última capa de la red hasta la primera. De ahí que el algoritmo se llame propagación hacia atrás.

Algoritmo 4: Propagación hacia atrás (BP).

Datos: Red neuronal prealimentada $[L, d, W, \{\theta_t\}]$
Entrada: $x \in \mathbb{R}^{d_0}$, ℓ función diferenciable
$(x_1, \ldots, x_T), (s_1, \ldots s_T) = FP(x)$
$\delta_T = \nabla \ell(x_T)$
para *t = T-1,..., 1* **hacer**
  $\delta_t = \delta_{t+1} (\mathrm{J}\theta_t)_{s_t} W_t$
fin
Para cada t: $\nabla_{w_t} \ell(x) = \delta_t (\mathrm{J}\theta_t)_{s_t} X_{t-1}$
Salida: $\nabla_W \ell(x) = \left[\nabla_{W_1} \ell, \ldots, \nabla_{W_T} \ell\right]$

7.3. Redes neuronales convolucionales

Las redes neuronales convolucionales son también conocidas como redes convolucionales o CNNs. Toman su nombre de la operación de convolución discreta, que presentaremos a continuación, y son usadas para modelar series temporales e imágenes. La inclusión de la operación de convolución, entre otras, en el tratamiento de este tipo de datos ha supuesto un salto de calidad en la capacidad predictiva de estos modelos.

Una red convolucional está formada por una secuencia de operaciones o funciones que se aplican sucesivamente, de forma similar a lo que ocurre en una red neuronal prealimentada. Entenderemos que una red de este tipo está formada por capas, y cada capa toma un tensor de entrada y lo transforma en otro tensor aplicando una operación.

En esta sección estudiamos las operaciones comunes que suelen emplearse en una red convolucional: convolución, funciones de activación y pooling.

7.3.1. Convolución y correlación

Si recordamos la definición de las redes neuronales prealimentadas, en cada capa aplicábamos una transformación lineal al vector de la entrada mediante una matriz de pesos. En las redes convolucionales la convolución hace el papel de transformación lineal. Presentamos a continuación dos operaciones que están íntimamente ligadas: convolución y correlación.

La convolución tiene su origen en el campo del Análisis Matemático como un producto entre dos funciones valuadas en los números reales. Dadas dos funciones reales f y g, la convolución en un punto x viene dada por

$$(f * g)(x) = \int f(x-t)g(t)dt$$

siempre que la integral del miembro de la derecha tenga sentido. La convolución discreta, que es en la se inspiran estas redes, puede definirse de forma natural para las sucesiones del espacio de Banach ℓ_1.

Definición 7.4 (Convolución en ℓ_1)**.** Sean x, y dos sucesiones de números reales indizadas en $\mathbb{Z}$ verificando

$$\sum_{j=-\infty}^{\infty} |x_j| < \infty, \sum_{j=-\infty}^{\infty} |y_j| < \infty,$$

Definimos la sucesión

$$(x * y)_k = \sum_{j=-\infty}^{\infty} x_{k-j} y_j$$

Es fácil ver que $(x * y)_k$ está bien definido y que $\sum_{k=-\infty}^{\infty} |(x * y)_k| < \infty$. A la sucesión $x * y$ le llamamos convolución de x con y.

Se puede comprobar que esta operación es lineal en ambas variables, conmutativa y asociativa. Esta definición se puede generalizar a funciones discretas definidas en $\mathbb{Z}^d$. Decimos que $f\colon \mathbb{Z}^d \to \mathbb{R}$ es de soporte acotado si el conjunto $sop(f) = \{x \in \mathbb{Z}^d \colon f(x) \neq 0\} \subset \mathbb{R}^d$ es acotado.

Definición 7.5. Sean dos funciones discretas $f, g : \mathbb{Z}^d \to \mathbb{R}$ de soporte acotado. Definimos la

convolución d dimensional discreta de f con g en el punto $x = (x_1, \dots, x_d)$ como

$$(f * g)(x) = (f * g)(x_1, \dots, x_d) = \sum_{y_1=-\infty}^{+\infty} \dots \sum_{y_d=-\infty}^{+\infty} f(x_1 - y_1, \dots, x_d - y_d) g(y_1, \dots, y_d)$$

Así obtenemos una nueva función discreta $f * g$ que llamamos convolución de f con g.

Una vez hechas las presentaciones, pretendemos trasladar esta operación al campo de tensores de cualquier dimensión. Empezamos definiéndola para matrices.

Definición 7.6 (Convolución 2D)**.** Sea $F \in \mathbb{R}^{a_1 \times a_2}, G \in \mathbb{R}^{p \times p}$ con $p = 2b + 1$ y $2b < a_1, a_2$. Definimos la convolución de F y G como la matriz $F * G \in \mathbb{R}^{a_1 - 2b \times a_2 - 2b}$ dada por

$$(F * G)_{i,j} = \sum_{n=-b}^{b} \sum_{m=-b}^{b} F_{i-n, j-m} G_{n,m} \tag{7.3}$$

para cada $1 + b \leq i \leq a_1 - b$ y $1 + b \leq j \leq a_2 - b$.

En (7.3) el lector se habrá percatado de que los índices no son correctos. En primer lugar, debemos entender que el elemento $G_{n,m}$ se corresponde en realidad con $G_{n+b+1, m+b+1}$. En segundo lugar, si tomamos

$$1 \leq l \leq a_1 - 2b, 1 \leq q \leq a_2 - 2b$$

entonces el elemento $(F * G)_{l,q}$ se corresponde en realidad con $(F * G)_{l+b, q+b}$. Si quisiéramos mantener la coherencia en los índices deberíamos de escribir:

$$(F * G)_{i,j} = \sum_{n=1}^{2b+1} \sum_{m=1}^{2b+1} F_{i+2b+1-n, j+2b+1-m} G_{n,m}$$

la cual es una expresión con la que trabajar es más tedioso. De ahora en adelante, nos referiremos a la expresión de (7.3) teniendo en cuenta estas observaciones.

Definición 7.7 (Correlación cruzada 2D)**.** Sea $F \in \mathbb{R}^{a_1 \times a_2}, \mathbb{R}^{p \times p}$ con $p = 2b + 1$ y $2b < a_1, a_2$. Definimos la correlación cruzada de F y G como la matriz $F \star G \in \mathbb{R}^{a_1 - 2b \times a_2 - 2b}$ dada por

$$(F \star G)_{i,j} = \sum_{n=-b}^{b} \sum_{m=-b}^{b} F_{i+n, j+m} G_{n,m}$$

para cada $1 + b \leq i \leq a_1 - b$ y $1 + b \leq j \leq a_2 - b$.

La relación entre correlación y convolución es evidente:

$$F * G = F \star \widehat{G}$$

donde $\widehat{G}_{n,m} = G_{-n,-m}$. Para definir la convolución en 3 dimensiones basta aplicar la convolución 2D recién definida a cada canal.

Definición 7.8 (Convolución y correlación 3D)**.** Sea $F \in \mathbb{R}^{a_1 \times a_2 \times c}, G \in \mathbb{R}^{p \times p \times c}$ con $p = 2b + 1$ y $2b < a_1, a_2$. Definimos:

- la convolución de F y G como la matriz $F * G \in \mathbb{R}^{a_1-2b \times a_2-2b}$ dado por

$$(F * G)_{i,j} = \sum_{k=1}^{c} (F(k) * G(k))_{i,j}$$

- la correlación de F y G como la matriz $F \star G \in \mathbb{R}^{a_1-2b \times a_2-2b}$ dado por

$$(F * G)_{i,j} = \sum_{k=1}^{c} (F(k) \star G(k))_{i,j}$$

donde $G(k)$ es la matriz dada por $G(k)_{i,j} = G_{i,j,k}$ y $F(k)$ es la matriz dada por $F(k)_{i,j} = F_{i,j,k}$.

Al tensor G se le denomina **filtro, kernel o núcleo**. Un banco de filtros será un conjunto de tensores de tres dimensiones (que expresaremos como un tensor de cuatro dimensiones). Podemos aplicar un banco de filtros a un tensor concatenando el resultado de aplicar estos filtros individualmente:

Definición 7.9 (Convolución y correlación 3D con varios filtros)**.** Sea $F \in \mathbb{R}^{a_1 \times a_2 \times c}, G \in \mathbb{R}^{p \times p \times c \times N}$ con $p = 2b + 1$ y $2b < a_1, a_2$. Definimos:

- la convolución de F y G como el tensor $F * G \in \mathbb{R}^{a_1-2b \times a_2-2b \times N}$ dado por

$$(F * G)_{i,j,n} = (F * G(n))_{i,j}$$

- la correlación de F y G como el tensor $F \star G \in \mathbb{R}^{a_1-2b \times a_2-2b \times N}$ dado por

$$(F \star G)_{i,j,n} = (F \star G(n))_{i,j}$$

donde ahora $G(n)$ es el tensor dado por $G(n)_{i,j,k} = G_{i,j,k,n}$.

Una capa convolucional aplica un banco de filtros al tensor de entrada, obteniendo otro tensor. Una red que use una capa convolucional deberá aprender los parámetros que definen a los filtros. El tamaño del tensor de salida quedará determinado por el de entrada y el del banco de filtro que usemos.

Cuando convolucionamos una imagen con un filtro, obtenemos un tensor de menor dimensión. De alguna forma, la convolución nos ayuda a *resumir* la información presente en la imagen en unas pocas características.

En las redes neuronales prealimentadas en cada capa se obtiene un vector y cada componente de este vector está afectado por todos los elementos del vector de entrada a esa capa. En contraposición, en una capa convolucional, sólo algunos elementos de la entrada intervienen en la construcción de cada elemento de la salida. Por ello, a las capas de las redes prealimentadas se les llaman *capas totalmente conectadas*.

De aquí se deriva una importante característica de las capas convolucionales: para simplificar, si la entrada tiene m componentes y la salida n componentes, una capa totalmente conectada necesita del orden de mn parámetros, mientras que una capa convolucional que usa un filtro de k parámetros necesitará kn parámetros. Esto tiene importantes implicaciones en la eficiencia de los algoritmos fordward propagation y backward propagation que hemos presentado en la anterior sección.

Por último, vamos a señalar una idea importante que relaciona a la convolución y a la multiplicación de matrices. La convolución es una operación lineal, luego podemos pensar en expresarla como el **producto de dos matrices**. Esto tiene dos consecuencias interesantes:

- La convolución se introduce de forma natural en el modelado de red neuronal que hemos presentado en la sección anterior, sin más que restringir algunos parámetros de la matriz de pesos de cada capa.
- Podemos aprovechar los numerosos algoritmos para multiplicación de matrices que conocemos para implementar la convolución de forma eficiente.

Vamos a ver esto primero para la convolución de una matriz con otra y a continuación para el caso general de la convolución de dos tensores.

Dada una matriz $A \in \mathbb{R}^{a,b}$, vamos a denotar por $r(A) \in \mathbb{R}^{ab}$ el vector formado por las filas de A concatenadas. Dado un tensor $B \in \mathbb{R}^{a,b,c}$ denotamos por $r(B)$ al vector formado por la concatenación de $r(B(1)), \ldots, r(B(c))$.

Empecemos suponiendo que $F \in \mathbb{R}^{a_1 \times a_2}$, $G \in \mathbb{R}^{p \times p}$ con $p = 2b + 1$. Para cada entrada $F_{i,j}$ consideramos la submatriz de F de dimensiones $p \times p$ cuyo elemento central es $F_{i,j}$, la cual vamos a denotar por $P_{i,j} \in \mathbb{R}^{p \times p}$. Construimos a continuación la matriz $\boldsymbol{F} \in \mathbb{R}^{(a_1-2b)(a_2-2b) \times p^2}$ definida por

$$\boldsymbol{F} = \begin{bmatrix} r(P_{1+b,1+b}) \\ \vdots \\ r(P_{1+b,a_2-b}) \\ \vdots \\ r(P_{a_1-b,1+b}) \\ \vdots \\ r(P_{a_1-b,a_2-b}) \end{bmatrix}$$

donde $r(P_{i,j}) \in \mathbb{R}^{p^2}$ es el vector formado por las filas de $P_{i,j}$ concatenadas. También consideramos $\boldsymbol{G} = r(G)$. Es fácil observar que

$$(F \star G)_{i,j} = r(P_{i,j})^T r(G)$$

y que por tanto el producto de matrices $\boldsymbol{FG}$ es un vector formado por las filas de $F \star G$ concatenadas.

Ejemplo 7.3. Consideramos $F \in \mathbb{R}^{3\times4}$, $G \in \mathbb{R}^{3\times3}$ dadas por

$$F = \begin{bmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \end{bmatrix}, \quad G = \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix}$$

Entonces:

$$P_{2,2} = \begin{bmatrix} 1 & 2 & 3 \\ 5 & 6 & 7 \\ 9 & 10 & 11 \end{bmatrix}, \quad P_{2,3} = \begin{bmatrix} 2 & 3 & 4 \\ 6 & 7 & 8 \\ 10 & 11 & 12 \end{bmatrix}$$

$$\boldsymbol{F} = \begin{bmatrix} 1 & 2 & 3 & 5 & 6 & 7 & 9 & 10 & 11 \\ 2 & 3 & 4 & 6 & 7 & 8 & 10 & 11 & 12 \end{bmatrix}, \quad \boldsymbol{G} = \begin{bmatrix} a \\ b \\ c \\ d \\ e \\ f \\ g \\ h \\ i \end{bmatrix}$$

Para el caso general, consideramos $F \in \mathbb{R}^{a_1 \times a_2 \times c}$, $G \in \mathbb{R}^{p \times p \times c \times N}$ con $p = 2b + 1$. Siguiendo la misma estrategia, para cada entrada (i, j, k) de F consideramos la submatriz de F de dimensiones $p \times p$ cuyo elemento central es $F_{i,j,k}$ y que denotamos por $P_{i,j,k} \in \mathbb{R}^{p \times p}$. Construimos las matrices

$$\boldsymbol{F} = \begin{bmatrix} r(P_{1+b,1+b,1}) & \cdots & r(P_{1+b,1+b,c}) \\ \vdots & \vdots & \vdots \\ r(P_{1+b,a_2-b,1}) & \cdots & r(P_{1+b,a_2-b,c}) \\ \vdots & \vdots & \vdots \\ r(P_{a_1-b,1+b,1}) & \cdots & r(P_{a_1-b,1+b,c}) \\ \vdots & \vdots & \vdots \\ r(P_{a_1-b,a_2-b,1}) & \cdots & r(P_{a_1-b,a_2-b,c}) \end{bmatrix}, \quad \boldsymbol{G} = \begin{bmatrix} r(G(1)) \cdots r(G(N)) \end{bmatrix}$$

donde, como antes, $G(n)$ es el tensor dado por $G(n)_{i,j,k} = G_{i,j,k,n}$. Es fácil observar que

$$(F \star G)_{i,j,n} = \begin{bmatrix} r(P_{i,j,1}) \cdots r(P_{i,j,c}) \end{bmatrix}^T r(G(n))$$

y por tanto $\boldsymbol{FG}$ es una matriz que tiene en la fila n los elementos de la matriz $(F \star G)(n)$.

Ejemplo 7.4. Consideramos $F \in \mathbb{R}^{3 \times 4 \times 2}$, $G \in \mathbb{R}^{3 \times 3 \times 2}$

$$F(1) = F(2) = \begin{bmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \end{bmatrix}, \quad G(1) = G(2) = \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix}$$

Entonces:

$$P_{2,2,1} = P_{2,2,2} = \begin{bmatrix} 1 & 2 & 3 \\ 5 & 6 & 7 \\ 9 & 10 & 11 \end{bmatrix}, \quad P_{2,3,1} = P_{2,3,2} = \begin{bmatrix} 2 & 3 & 4 \\ 6 & 7 & 8 \\ 10 & 11 & 12 \end{bmatrix}$$

$$\boldsymbol{F} = \begin{bmatrix} 1 & 2 & 3 & 5 & 6 & 7 & 9 & 10 & 11 & 1 & 2 & 3 & 5 & 6 & 7 & 9 & 10 & 11 \\ 2 & 3 & 4 & 6 & 7 & 8 & 10 & 11 & 12 & 2 & 3 & 4 & 6 & 7 & 8 & 10 & 11 & 12 \end{bmatrix}$$

$$\boldsymbol{G} = \begin{bmatrix} a & a \\ b & b \\ c & c \\ d & d \\ e & e \\ f & f \\ g & g \\ h & h \\ i & i \end{bmatrix}$$

7.3.2. Función de activación

La función de activación suele actuar sobre el tensor obtenido sobre la operación de convolución. La elección de esta función determina el tipo de unidades ocultas que estamos usando en nuestra red. La función de activación que se sitúe en la última capa determinará dónde toma valores la jerarquía de funciones que implementa la red

Estas funciones cumplen el mismo papel que las funciones de activación presentadas en las redes neuronales prealimentadas: es importante que sean diferenciables, o al menos, diferenciables en casi todo punto, para poder aplicar la propagación hacia atrás.

El diseño de funciones de activación y unidades ocultas es un campo muy activo, pero no se guía por principios teóricos, sino por procesos de *ensayo y error*. A continuación señalamos las unidades más populares. A veces bastará definir una función real de variable real y entender que las unidades ocultas correspondientes la aplican componente a componente.

- Las **unidades lineales rectificadas** ([22]), también llamadas **RELU**, son una de las más populares en redes convolucionales para problemas de clasificación. Su función de activación es:

  $$\theta(t) = \max\{0, t\}$$

  Leaky RELU es una variante de RELU que implementa:

  $$\theta(t) = \max\{0, t\} + \alpha \min\{0, t\}$$

  siendo $\alpha \in \mathbb{R}$.

- Unidades **maxout**: dividen un vector de entrada $x \in \mathbb{R}^{kd}$ en k subvectores y calculan el máximo de cada uno, obteniendo un vector de salida $z \in \mathbb{R}^d$

- Unidades con **activación sigmoidal**: usan la función sigmoidal ya presentada con anterioridad:

  $$\sigma\colon \mathbb{R} \to [0,1], \quad \sigma(t) = \frac{e^t}{1+e^t}$$

  Las unidades con **activación tangente hiperbólica** usan:

  $$\tanh\colon \mathbb{R} \to [0,1], \quad \tanh(t) = \frac{e^t - e^{-t}}{e^t + e^{-t}}$$

  En problemas de clasificación binarios, estas funciones de activación suelen usarse en la última capa, de forma que la salida de la red proporcione las probabilidades de

pertenecer o no a cada una de las clases.

- Unidades con **activación softmax**: la función softmax se define como sigue:

$$\text{softmax}\colon \mathbb{R}^N \to [0,1]^N, \quad \text{softmax}(x)_k = \frac{e^{x_k}}{\sum_{n=1}^{N} e^{x_n}}$$

  La función softmax suele usarse en problemas de clasificación con varias clases. Al colocarla en la salida de la red, se interpreta que la red proporciona las probabilidades de pertenecer a cada una de las clases consideradas.

7.3.3. Pooling

Dada una matriz $M \in \mathbb{R}^{sa \times sb}$ la operación de *pooling* con *stride* $s \in \mathbb{N}$ obtiene una nueva matriz $Pool(M) \in \mathbb{R}^{a \times b}$ definida por

$$Pool(M)_{i,j} = f_{Pool}(M_s(i,j))$$

donde $f_{Pool}\colon \mathbb{R}^{s \times s} \to \mathbb{R}$ es una función que actúa sobre las submatrices

$$M_s(i,j) = [M_{l,m}\colon s(i-1) < l \leq si,\ s(j-1) < m \leq sj]$$

Como vemos, $Pool(M)$ es una nueva matriz cuyos elementos resumen de acuerdo a f_{Pool} la información en celdas contiguas de la matriz original. Algunos ejemplos populares que se suelen usar en las redes actuales son:

- Max pooling, en la que $f_{Pool} = max$, es decir:

$$Pool(M)_{i,j} = \max\{M_{l,m}\colon s(i-1) < l \leq si,\ s(j-1) < m \leq sj\}$$

- Average pooling, en la que f_{Pool} es el promedio de los elementos de la submatriz:

$$Pool(M)_{i,j} = \frac{1}{s^2} \sum_{s(i-1)<l\leq si} \ \sum_{s(j-1)<m\leq sj} M_{l,m}$$

- $p-$norm, en la que f_{Pool} aplica la norma p:

$$Pool(M)_{i,j} = \left(\sum_{s(i-1)<l\leq si} \ \sum_{s(j-1)<m\leq sj} (M_{l,m})^p \right)^{\frac{1}{p}}$$

Es muy común usar max pooling o average pooling con stride 2. En la Figura 7.1 se muestra el funcionamiento de max pooling y average pooling sobre un ejemplo.

7.3.4. Salida estructurada

La salida de las redes neuronales, y por tanto también la de las redes convolucionales no tiene por qué limitarse a valores reales para tareas de clasificación ó regresión. De hecho, la salida de una red convolucional será un tensor cuyas dimensiones están determinadas por

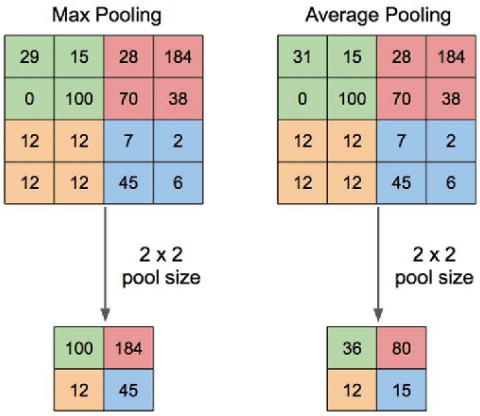


Figura 7.1.: Ejemplo de *max pooling* y *average pooling* ([39]).

las operaciones que definen a la red. Es por ello que las redes convolucionales se pueden utilizar para tareas muy diversas.

Una de estas tareas es la de *segmentación*. Un modelo convolucional que acepte imágenes puede emitir un tensor A de forma que $A_{i,j,k}$ es la probabilidad de que el píxel (i, j) de la imagen de entrada pertenezca a la clase k. Esto permite clasificar cada píxel y dibujar máscaras o contornos que delimiten a los objetos presentes en la imagen.

La aproximación al problema de segmentación requiere realizar consideraciones que escapan al propósito de este trabajo. Sólo debemos conocer que existe este problema y que existen modelos dedicados a resolverlo.

Un ejemplo de detección de objetos usando segmentación se observa en la Figura 7.2.

Figura 7.2.: Segmentación y detección de vehículos en una imagen ([17]).

7.4. Redes generativas antagónicas

Las redes generativas antagónicas, o *generative adversarial networks (GANs)* en inglés, es un *modelo generativo* propuesto por Goodfellow et al. [10]. Los modelos generativos tienen co-

mo objetivo aprender la distribución del conjunto de datos para poder generar de forma autónoma nuevos ejemplos del mismo.

El término *antagónicas* hace referencia a que este modelo usa dos redes neuronales que juegan un papel opuesto en el entrenamiento. Una de ellas es el *discriminador* o clasificador, que se entrena para distinguir ejemplos reales del conjunto de entrenamiento de ejemplos falsos. La otra es el *generador*, que intenta engañar al discriminador creando ejemplos falsos. En la Figura 7.3 podemos ver un ejemplo del uso de este modelo en la generación de nuevas imágenes.

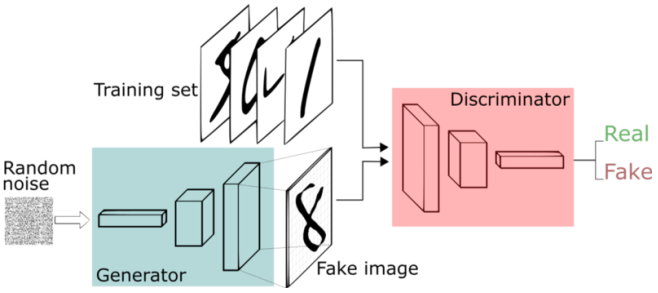


Figura 7.3.: GANs para la generación de nuevas imágenes ([1]).

Las GANs obtienen buenos resultados en la práctica, pero presentan ciertas dificultades derivadas en la forma en que se describen. Como consecuencia, no existe un entendimiento profundo de cómo convergen ni a qué convergen durante el proceso de entrenamiento.

En esta sección pretendemos señalar los resultados más importantes en la formulación original de las GANs y comentar cuáles son los problemas en que se centra la investigación de este campo.

7.4.1. Motivación

El modelado generativo, o *generative modeling* en inglés, tiene como objetivo aprender la distribución de probabilidad P_r que representa a los datos de entrenamiento. La aproximación de las redes generativas antagónicas consiste en aprender a generar nuevos ejemplos.

El modelo GANs, como ya hemos comentado, se puede entender como una competición entre generador y discriminador. El generador se puede representar como una función G que induce una distribución de probabilidad P_G. El discriminador también se puede representar como una función D que asigna a cada ejemplo x real o falso un número real $D(x) \in [0,1]$ que representa la probabilidad de que x sea un ejemplo real.

El generador debe producir imágenes que engañen al discriminador, por lo que debe aprender *algo* de la distribución P_r. Fijaremos una distribución de probabilidad P_z (que típicamente será una distribución normal). El generador tomará como entrada $z \sim P_z$ y generará un nuevo ejemplo $G(z) \sim P_G$. El generador pretenderá que $D(G(z))$ sea cercano a 1, es decir, que el discriminador asegure que con alta probabilidad un ejemplo falso es real.

7.4.2. Minimax GANs

Fijemos $\mathcal{X} \subset \mathbb{R}^n$, $\mathcal{Z} \subset \mathbb{R}^m$. Supongamos que $(\Omega, \mathcal{A}, \mathbb{P})$ es un espacio de probabilidad y sean $X\colon \Omega \to \mathcal{X}$, $Z\colon \Omega \to \mathcal{Z}$ dos variables aleatorias, cuyas distribuciones de probabilidad

son P_r y P_z respectivamente. Además, supondremos que ambas distribuciones son absolutamente continuas respecto a la medida de Lebesgue, con funciones de densidad p_r y p_z respectivamente.

Observación 7.2. Todos los resultados que siguen son ciertos si p_r y p_z son funciones masa de probabilidad y las demostraciones que aquí presentamos pueden obtenerse de forma análoga.

Definición 7.10 (Minimax GANs [10])**.** Consideramos la aplicación

$$V(G,D) = \mathbb{E}_{X\sim P_r}\left[\log(D(X))\right] + \mathbb{E}_{Z\sim P_z}\left[\log(1-D(G(Z)))\right]$$

donde $G\colon \mathcal{Z} \to \mathcal{X}$ y $D\colon \mathcal{X} \to [0,1]$. Nos referiremos al problema

$$\min_G \max_D V(G,D) \tag{7.4}$$

como el problema minimax de las redes generativas antagónicas, o minimax GANs para abreviar.

Observación 7.3. El primer sumando de V hace referencia a la probabilidad que D asigna a los ejemplos reales, mientras que el segundo a se refiere a la probabilidad que D asigna a los ejemplos falsos. D debe maximizar ambos sumandos (quiere distinguir correctamente tanto ejemplos reales como falsos), mientras que G sólo debe minimizar el segundo sumando (sólo quiere engañar al generador con ejemplos falsos).

Observación 7.4. El problema (7.4) busca su solución entre todas las posibles funciones D y G. Esto en la práctica no es posible, luego podemos decir que es un problema idealizado.

Llamemos $P_G = P_z \circ G^{-1}$. Si $Z \sim P_z$ entonces es claro que $Y = G(Z) \sim P_G$. Podemos escribir por tanto

$$V(G,D) = \mathbb{E}_{X\sim P_r}\left[\log(D(X))\right] + \mathbb{E}_{Y\sim P_G}\left[\log(1-D(Y))\right]$$

Diremos que $G\colon \mathcal{Z} \to \mathcal{X}$ es un *generador* si P_G admite una función de densidad, que notaremos por p_G. Así mismo, nos referiremos a $D\colon \mathcal{X} \to [0,1]$ como un *discriminador*.

Observación 7.5. Para el razonamiento que sigue, es necesario contar con una función de densidad de P_G. Para encontrar condiciones que aseguren esta existencia, podríamos recurrir al teorema de Radon-Nikodym ([14]): si $G^{-1}(B) = 0$ para cada $B \subset \mathbb{R}^n$ tal que $\lambda^n(B) = 0$, entonces G es un generador.

Teorema 7.2 ([10])**.** *Sea G un generador. Entonces el discriminador óptimo D^* de (7.4) viene dado por*

$$D^*(x) = \frac{p_r(x)}{p_r(x) + p_G(x)}, \quad \forall x \in sop(p_r) \cup sop(p_G)$$

Demostración. De la definición de esperanza

$$\begin{aligned} V(G,D) &= \int_{\mathcal{X}} p_r(x)\log(D(x))dx + \int_{\mathcal{X}} p_G(y)\log(1-D(y))dy = \\ &= \int_{\mathcal{X}} \left[p_r(x)\log(D(x)) + p_G(x)\log(1-D(x))\right]dx \end{aligned}$$

Sean $a, b \in \mathbb{R} \setminus \{0\}$. La función $g\colon [0,1] \to \mathbb{R}$ dada por

$$g(y) = a\log(y) + b\log(1-y), \quad \forall y \in [0,1],$$

es continua y está definida en un compacto, por lo que alcanza el máximo. De hecho, es fácil ver que g alcanza su máximo en el punto $\frac{a}{a+b}$. Es claro entonces que

$$V(G,D) \leq V(G,D^*)$$

□

Observación 7.6. Si tomamos $p_G = p_r$ se tiene que $D^*(x) = \frac{1}{2}$. El discriminador asigna la misma probabilidad a un ejemplo real y a un ejemplo falso porque no es capaz de diferenciarlos. A continuación buscamos probar que esta es la situación que se tiene cuando generador y discriminador son óptimos.

Como ya conocemos el discriminador óptimo, podemos dejarlo fijo y centrarnos en encontrar el generador óptimo. Esto es lo que pretende resaltar el siguiente corolario.

Corolario 7.1. *El problema Minimax GANs es equivalente a*

$$\min_G C(G)$$

donde $G\colon \mathcal{Z} \to \mathcal{X}$ *y*

$$C(G) = \mathbb{E}_{X\sim P_r}\left[\log\left(\frac{p_r(X)}{p_r(X)+p_G(X)}\right)\right] + \mathbb{E}_{X\sim P_G}\left[\log\left(\frac{p_G(X)}{p_r(X)+p_G(X)}\right)\right]$$

Demostración. Basta observar que $C(G) = V(G,D^*)$. □

Teorema 7.3. *Sea G un generador. Entonces $-\log(4) \leq C(G)$ y la igualdad se da si y sólo si $p_r = p_G$. Como consecuencia, fijado el discriminador óptimo D^*, el generador óptimo G^* para el problema (7.4) se alcanza sólo cuando $p_{G^*} = p_r$.*

Demostración. Sea G un generador. Entonces

$$C(G) + \log(4) = D_{KL}(P_r, \frac{P_r+P_G}{2}) + D_{KL}(P_G, \frac{P_r+P_G}{2} = 2D_{JS}(P_r, P_G)$$

El Corolario 4.1 nos dice que $C(G) + \log(4) \geq 0$, y la igualdad se da si y sólo si $p_r = p_G$. □

7.4.3. Implementación del modelo GANs

En la práctica, la búsqueda de del par (G, D) no se hace sobre todas las posibles funciones descritas en el apartado anterior. Como en todo problema de aprendizaje, consideramos una familia funciones parametrizadas y buscamos los mejores parámetros de acuerdo al problema que queremos resolver:

$$G_{W_G}(z) = G(z, W_G) \quad W_G \in \Theta_G$$

$$D_{W_D}(x) = D(x, W_D), \quad W_D \in \Theta_D$$

Consideraremos que G_{W_G} y D_{W_D} pertenecen a la clase de las redes neuronales presentada con anterioridad. Como hemos probado en secciones anteriores, estas son funciones diferenciables respecto a los parámetros que las definen y nos permiten entrenarlas aplicando el método del gradiente descendente.

Manteniendo la notación del apartado anterior, el nuevo problema se formula:

$$\min_{W_G \in \Theta_G} \max_{W_D \in \Theta_D} V(G_{W_G}, D_{W_D})$$

El proceso de entrenamiento, propuesto inicialmente en [10], se muestra en el algoritmo 5. Está inspirado en el estudio teórico que hemos presentado en la sección anterior: en primer lugar, se busca el mejor discriminador y, con él fijo, se realizan las actualizaciones en los pesos del generador.

Algoritmo 5: Entrenamiento de GANs mediante SGD.

Entrada: Tasa de aprendizaje η, conjunto $\mathcal{D}_r$ de datos reales, $\mathcal{D}_z$ datos generados de acuerdo a P_z

Inicializar $W_D(0), W_G(0)$

para $t = 0, 1, 2, \ldots$ **hacer**

  para $k = 0, 1, \ldots$ **hacer**

    Escoger $\{x_1, \ldots, x_M\} \subset \mathcal{D}_r$, $\{z_1, \ldots, z_M\} \subset \mathcal{D}_z$

    Calcular:

$$g_D(k) = \frac{1}{M} \nabla_{W_D} \sum_{m=1}^{M} \log\left(D(x_m, W_D(k))\right) + \log\left(1 - D(G(z_m, W_G(t)))\right)$$

    Actualizar: $W_D(k+1) = W_D(k) - \eta g_D(k)$

  fin

  Escoger $\{z_1, \ldots, z_M\} \subset \mathcal{D}_z$

  Calcular:

$$g_G(t) = \frac{1}{M} \nabla_{W_G} \sum_{m=1}^{M} \log\left(1 - D(G(z_m, W_G(t)))\right)$$

  Actualizar: $W_G(t+1) = W_G(t) - \eta g_G(t)$

fin

7.5. Transformaciones inherentes a la clase: FUCITNET

FUCITNET es un modelo propuesto es [25] para problemas de clasificación multiclase. Está inspirado en el modelo GANs, pero en vez de entrenar un sólo generador, entrena tantos generadores como clases haya en el problema. Cada generador, en vez de intentar engañar al discriminador, intenta "aprender transformaciones inherentes a su clase". Esto significa que cuando un generador recibe una imagen, el objetivo es que produzca otra donde las características de la clase del generador se encuentran potenciadas. De esta forma, se espera que el clasificador tendrá más éxito en su labor.

7.5.1. Entrenamiento e inferencia

Fijemos $\mathcal{X} \subset \mathbb{R}^d$, $\mathcal{Y} = \{1, \ldots, C\}$ donde C es el número de clases del problema. Supongamos que P es una distribución de probabilidad definida sobre el espacio $\mathcal{X} \times \mathcal{Y}$ y el conjunto de entrenamiento ha sido generado de acuerdo a P.

FUCITNET está compuesto por un conjunto de generadores

$$\{G_k \colon \mathcal{X} \to \mathcal{X} \colon k \in \mathcal{Y}\},$$

y un clasificador $D \colon \mathcal{X} \to [0,1]^C$, de forma que $D(x)_k$ pretende aproximar $P(y = k|x)$. Definimos $\widehat{D} \colon \mathcal{X} \times \mathcal{Y} \to [0,1]$ por $\widehat{D}(x,k) = D(x)_k$ para cada $k \in \mathcal{Y}$.

Consideramos las siguientes funciones de error:

- Se consideran dos versiones de la entropía cruzada. La primera de ellas, l_{CE}, es la entropía cruzada convencional aplicando a cada ejemplo la transformación de cada generador:

  $$l_{CE} = -\sum_{k=1}^{C} \mathbb{E}_{(X,Y)\sim P}\left[\log\left(\widehat{D}(G_k(X), Y)\right)\right].$$

  La segunda, que denotamos por $l_{CE}(k)$, es la entropía cruzada considerando sólo las imágenes de la clase k transformadas por el generador G_k:

  $$l_{CE}(k) = -\mathbb{E}_{(X,Y)\sim P}\left[\mathbb{1}_{\{k\}}(Y)\log\left(\widehat{D}(G_k(X), Y)\right)\right].$$

- Para cada generador, el error cuadrático medio entre cada $x \in \mathcal{X}$ y $G_k(x)$:

  $$l_{MSE}(k) = \mathbb{E}_X\left[\|X - G_k(X)\|^2\right].$$

- Consideramos una red VGG-16 preentrenada ([31]) y denotamos por Φ_n la transformación en la capa n-ésima de dicha red. Se define el error perceptual como:

  $$l_{perceptual}(k) = \sum_n \mathbb{E}_X\left[\|\Phi_n(X) - \Phi_n(G_k(X))\|^2\right].$$

El clasificador será entrenado para minimizar la versión empírica de l_{CE}. Cada generador G_k es entrenado para minimizar la versión empírica de la función

$$\underbrace{l_{MSE}(k) + 0.006 l_{perceptual}(k)}_{\text{Término de similaridad}} + \lambda l_{CE}(k), \tag{7.5}$$

donde $\lambda \in \mathbb{R}^+$ es un hiperparámetro a determinar en cada problema.

El término de similaridad se encarga de que cada generador no aprenda transformaciones que distorsionen demasiado la imagen original, mientras que el término de la entropía cruzada es el encargado de que cada generador se fije en las características que determinan su clase. Es claro que la minimización de $l_{CE}(k)$ favorece la clasificación de las imágenes transformadas por el generador G_k. La incorporación de este término al generador es la aportación más significativa de FUCITNET. En la Figura 7.4 se representa un esquema del proceso de entrenamiento.

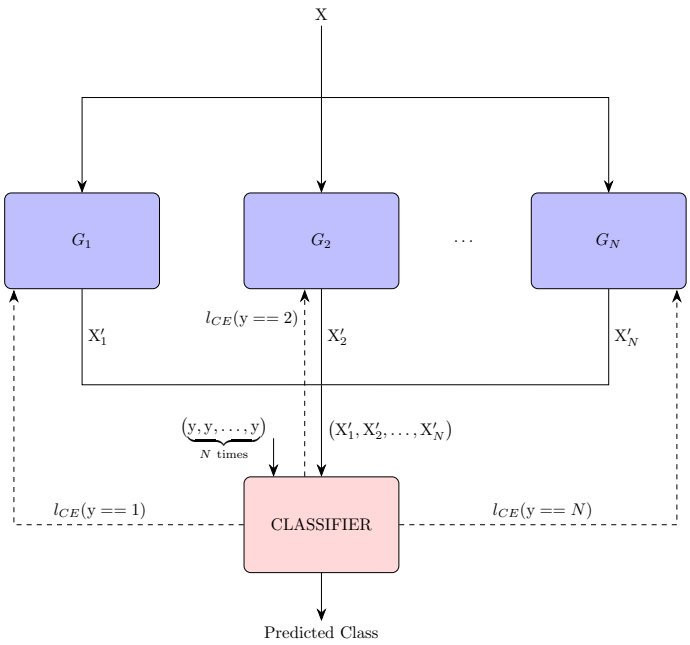


Figura 7.4.: Esquema del proceso de entrenamiento de FUCITNET ([25]). Dado $(X, y) \in \mathcal{X} \times \mathcal{Y}$, se calcula su imagen por cada generador obteniendo $G_k(x) = X'_k$. El clasificador calcula el error de entropía cruzada $l_{CE}(y == k)$ para cada clase, que es devuelto al generador correspondiente.

A la hora de inferir la clase de una nueva imagen (Figura 7.5), se clasifica cada transformación de la imagen y se asigna la etiqueta más probable. Esto es, dado $x \in \mathcal{X}$, se calcula:

$$[D(G_1(x)), \ldots, D(G_C(x))] \in [0,1]^{C^2}$$

y la etiqueta asignada al nuevo ejemplo x es

$$\arg\max [D(G_1(x)), \ldots, D(G_C(x))] \mod C$$

7.5.2. Arquitectura de los generadores y del clasificador

Todos los generadores usan la misma arquitectura, compuesta por 5 bloques residuales idénticos ([11]). Cada bloque residual está compuesto por dos capas convolucionales de dimensiones $3 \times 3 \times 64$ seguidos de una capa *batch-normalization* y de la activación RELU paramétrica.

El clasificador, por su parte, es una red Resnet18 ([11]) preentrenada en Imagenet ([28]) en la que se modifica la última capa para emitir un vector de probabilidades con tantos componentes como clases consideradas.

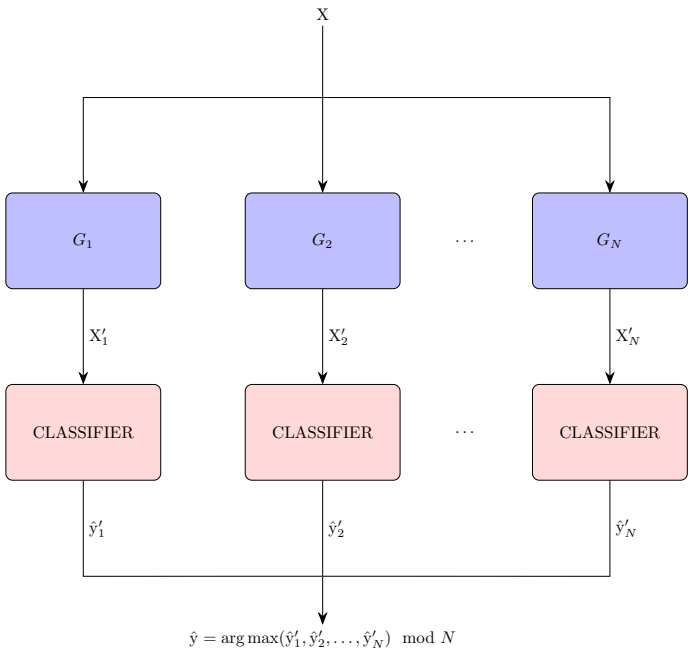


Figura 7.5.: Esquema del proceso de inferencia de FUCITNET ([25]).

7.6. La metodología SDNET

COVID-SDNET son las siglas de *COVID Smart Data based Network*, una infraestructura o metodología basada en técnicas de aprendizaje profundo para automatizar la diagnosis de la enfermedad COVID-19 a partir de radiografías de tórax ([32]). COVID-SDNET está formada por la unión de varios modelos y ha sido el punto de partida de este trabajo. Es por ello que en esta sección se recogen las ideas principales de tal metodología.

COVID-SDNET se organiza en tres etapas. La primera está dirigida a preprocesar los datos para eliminar la información innecesaria, mientras que la segunda pretende crear ejemplos a partir de los existentes que favorezcan la tarea de clasificación, que se lleva a cabo en la última etapa.

Para describir el proceso de entrenamiento y de inferencia denotaremos por $\mathcal{D}$ al conjunto de imágenes de entrenamiento:

$$\mathcal{D} = \{(x_1, y_1), \ldots, (x_N, y_N)\} \subset \mathcal{X} \times \{P, N\}$$

donde $y_n \in \{P, N\}$ es el diagnóstico de cada clase, que puede ser positivo (P) o negativo (N).

7.6.1. Recorte basado en segmentación

Dado que no todas las radiografías se hacen con los mismos dispositivos y la posición del paciente dentro de la misma puede variar, el primer paso es recortar la imagen original para obtener una imagen nueva en la que sólo figuren los pulmones del paciente.

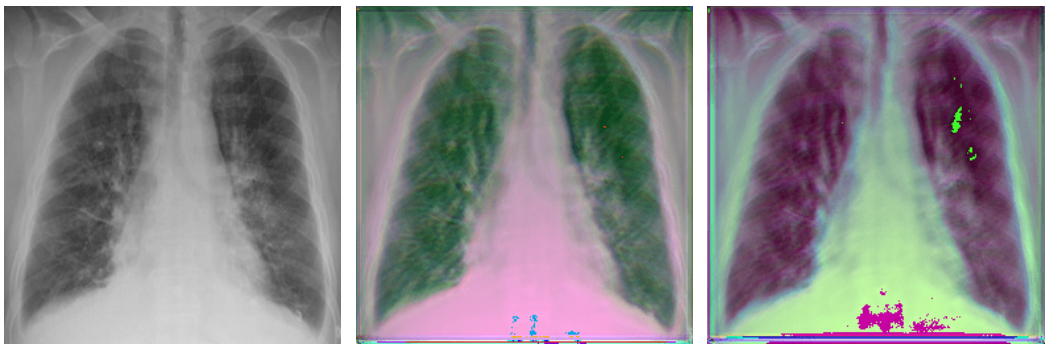
Para conseguir esto, primero se segmentan los pulmones usando una red U-Net preen-

trenada ([20]). A continuación se calcula el rectángulo más pequeño que contiene a ambos pulmones y, para evitar eliminar información útil, se amplía dicho rectángulo con el 2.5 % de píxeles en todas las direcciones (arriba, abajo, derecha, izquierda).

Para poder entrenar al modelo COVID-SDNET en primer lugar habrá que preprocesar el conjunto de entrenamiento aplicando la transformación que acabamos de describir, obteniendo un nuevo conjunto de entrenamiento $\mathcal{D}_s$.

7.6.2. Transformaciones inherentes a la clase

Se considera el modelo FUCITNET presentado en la sección anterior. Contaremos con dos generadores, G_N y G_P, y un clasificador D_F; los cuales serán entrenados usando el conjunto $\mathcal{D}_s$. A continuación cada imagen x es procesada para conseguir dos imágenes nuevas, $x_P = G_P(x)$ y $x_N = G_N(x)$, en las que se espera que las características de cada clase estén potenciadas. En la Figura 7.6 se puede observar un ejemplo del resultado de estas transformaciones.



(a) Imagen original x (N) (b) Transformación negativa $G_N(x)$ (c) Transformación positiva $G_P(x)$

Figura 7.6.: Transformaciones inherentes a la clase aplicadas a un ejemplo con diagnóstico negativo ([32]).

De esta forma, cuando el entrenamiento de FUCITNET haya finalizado podremos generar un nuevo conjunto de entrenamiento

$$\mathcal{D}_T = \{(G_P(x), y) \colon (x, y) \in \mathcal{D}_s\} \cup \{(G_N(x), y) \colon (x, y) \in \mathcal{D}_s\},$$

así como transformar el problema de clasificación binario original en un problema de clasificación de cuatro clases sobre el siguiente conjunto de entrenamiento:

$$\begin{aligned}\mathcal{D}_4 = \{(G_P(x), NTP) \colon (x, N) \in \mathcal{D}_s\} \cup \{(G_N(x), NTN) \colon (x, N) \in \mathcal{D}_s\} \cup \\ \{(G_P(x), PTP) \colon (x, P) \in \mathcal{D}_s\} \cup \{(G_N(x), PTN) \colon (x, P) \in \mathcal{D}_s\}.\end{aligned}$$

En este nuevo problema, las clases posibles son $\{NTP, NTN, PTP, PTN\}$. Veamos como resolviendo este problema de clasificación multiclase podemos dar una solución al problema original.

7.6.3. Clasificador SDNET

Consideramos la red convolucional Resnet-50 ([11]) preentrenada en el conjunto de datos Imagenet ([28]), de la cual se elimina la última capa y se le añade una nueva capa totalmente conectada de 512 neuronas con activación RELU, seguida de una capa totalmente conectada de 4 neuronas con activación softmax. Por tanto, esta red implementa un aplicación de $\mathcal{X}$ en $[0,1]^4$.

La red convolucional al completo es entrenada para minimizar la entropía cruzada sobre el conjunto $\mathcal{D}_4$. El objetivo es que la salida de la última capa proporcione la probabilidad de pertenecer a cada una de las 4 clases recientemente introducidas. Con estas probabilidades se aplica el siguiente proceso de inferencia para determinar si el diagnóstico es positivo o negativo:

Dado un ejemplo x, vamos a denotar por $\boldsymbol{\theta}(x) = [\theta_{PTP}(x), \theta_{PTN}(x), \theta_{NTP}(x), \theta_{NTN}(x)]$ a la salida del clasificador al ser evaluado en x. Para determinar la clase de x, evaluamos su transformación positiva y negativa:

$$\boldsymbol{\theta}(x_P) = \boldsymbol{\theta}(G_P(x)), \quad \boldsymbol{\theta}(x_N) = \boldsymbol{\theta}(G_N(x))$$

Denotemos por $y_P, y_N \in \{NTP, NTN, PTP, PTN\}$ la clase más probable de cada una de las transformaciones:

$$y_P = \arg\max \boldsymbol{\theta}(x_P), \quad y_N = \arg\max \boldsymbol{\theta}(x_N)$$

entendiendo que a cada una de las cuatro clases se les ha asignado el entero correspondiente. El diagnóstico de x estará determinado por la aplicación $\Psi\colon \mathcal{X} \to \{P, N\}$ que definimos de la siguiente forma:

1. Si $y_P = NTP,\ y_N = NTN$, entonces $\Psi(x) = N$.
2. Si $y_P = PTP,\ y_N = PTN$, entonces $\Psi(x) = P$.
3. Si no ocurre lo anterior:

$$\Psi(x) = \begin{cases} N & \text{si} \quad \max\{\theta_{NTk}(x_P), \theta_{NTk}(x_N)\} > \max\{\theta_{PTk}(x_P), \theta_{PTk}(x_N)\}, \forall k \in \{P, N\} \\ P & \text{en otro caso} \end{cases}$$

En la figura Figura 7.7 se representan cada una de las etapas que hemos descrito y cómo la información fluye de una a otra.

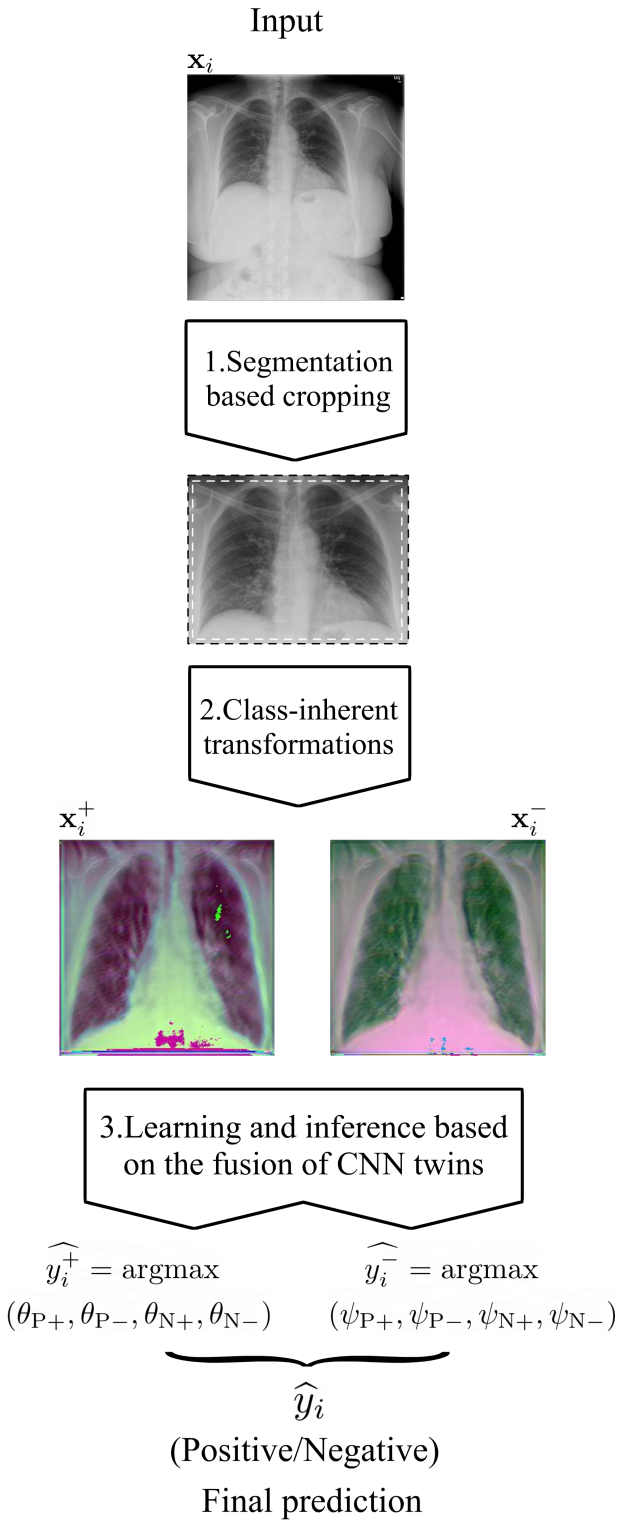


Figura 7.7.: Esquema de COVIDSDNET ([32]).

8. Aprendizaje federado

El paradigma de aprendizaje automático con el que hemos trabajado en los capítulos anteriores asume que el proceso, técnica o algoritmo de aprendizaje tiene todos los datos del conjunto de entrenamiento a su disposición. Solemos decir que esta es una configuración *centralizada*.

En la actualidad, los dispositivos móviles (smartphones y tablets) acompañan diariamente a sus usuarios, recogiendo grandes cantidades de datos a través de funcionalidades como la cámara, el micrófono y el GPS. Sin embargo, estos datos tienen una naturaleza sensible pues se debe preservar la información privada de los usuarios que llevan implícita. Esto imposibilita almacenar todos los datos de forma centralizada.

Este problema no se encuentra sólo en dispositivos móviles. Hay instituciones que poseen una gran cantidad de información de interés. Esta información debe ser tratada con un gran nivel de seguridad y privacidad, por lo que no puede salir de sus instalaciones. Por ejemplo, este es el caso de los datos que los hospitales albergan sobre sus pacientes.

Existe entonces la necesidad de reformular el problema del aprendizaje para poder entrenar modelos usando todos los datos disponibles en diversos dispositivos. Surge así el paradigma del *aprendizaje federado* ([19]), al cual nos referiremos como AF. El término federado se debe a que el proceso de aprendizaje se lleva a cabo a través de la federación o colaboración de diversos dispositivos que son coordinados por un servidor central.

8.1. Optimización federada

El paradigma de AFf, por ser una variación del problema de aprendizaje, lleva consigo un problema de optimización al que nos referiremos como *optimización federada*. Este problema debe ser resuelto para poder completar el proceso de entrenamiento.

Sean $\mathcal{Z} \subset \mathbb{R}^n, \Lambda \subset \mathbb{R}^d$. Sea $\ell\colon \mathcal{Z} \times \Lambda \to \mathbb{R}$ una medida de error. Suponemos la existencia de K *clientes* o *nodos* y una distribución de probabilidad P_k definida sobre $\mathcal{Z}$ asociada a cada nodo $k \in \{1, \ldots, K\}$. Definimos:

$$L_k(\alpha) = \mathbb{E}_{Z \sim P_k}\left[\ell(Z, \alpha)\right], \quad \forall \alpha \in \Lambda$$

El problema de optimización federada asociado consiste en resolver:

$$\min_{\alpha \in \Lambda} \frac{1}{K} \sum_{k=1}^{K} L_k(\alpha) \tag{8.1}$$

en el caso en que las distribuciones P_k son desconocidas pero, para cada $k \in \{1, \ldots, K\}$, conocemos una realización de una muestra aleatoria simple generada de acuerdo a P_k, que vamos a denotar por $\mathcal{D}_k \subset \mathcal{Z}$.

Seguimos el principio inductivo de minimización del riesgo empírico. Se define:

$$L_k^{emp}(\alpha) = \frac{1}{|\mathcal{D}_k|} \sum_{z \in \mathcal{D}_k} \ell(z, \alpha), \quad \forall \alpha \in \Lambda$$

Y consideramos el nuevo problema de optimización:

$$\min_{\alpha \in \Lambda} \frac{1}{K} \sum_{k=1}^{K} L_k^{emp}(\alpha)$$

Debemos tener en cuenta que cada nodo debe actuar sin conocer ninguna información sobre lo que ocurre en los otros nodos. La forma natural de proceder es que cada nodo k intente minimizar su función de error L_{emp}^k:

$$\alpha_k = \arg\min_{\alpha \in \Lambda} L_k^{emp}(\alpha),$$

y posteriormente, un nodo *central* se encargará de recoger toda esta información a través de un *operador de agregación*:

$$\Delta \colon \Lambda^K \to \Lambda$$
$$(\alpha_1, \ldots, \alpha_K) \mapsto \Delta(\alpha_1, \ldots, \alpha_K)$$

Este proceso de *aprendizaje-agregación* conforma una *ronda de aprendizaje*. En cada ronda de aprendizaje, el nodo k trata de minimizar la función L_k^{emp} usando un proceso o algoritmo A_k (que puede ser, por ejemplo, SGD). Posteriormente, el servidor central agrega los parámetros obtenidos por cada nodo y difunde el resultado entre los nodos. Al comienzo de la siguiente ronda de aprendizaje, cada nodo aplicará el proceso A_k partiendo de los pesos que acaba de recibir.

La iteración de rondas de aprendizaje usando los procesos de entrenamiento y el operador de agregación conforma el algoritmo general de AF, que presentamos en el algoritmo 6.

Cuando el número de clientes es muy grande se elige un subconjunto de clientes en cada ronda de aprendizaje y se realizan las actualizaciones sólo con ellos.

Algoritmo 6: Proceso de aprendizaje federado.

Entrada: $\mathcal{D}_1, \ldots, \mathcal{D}_K \subset \mathcal{Z}$, $\ell \colon \mathcal{Z} \times \Lambda \to \mathbb{R}$, procesos de aprendizaje A_k, operador de agregación Δ

inicio
- Inicializar $\alpha(0), \alpha_1(0), \ldots, \alpha_K(0)$
- **para** $r = 0, 1, 2, \ldots$ **hacer**
  - **para** $k \in \{1, \ldots, K\}$ **hacer**
    - $\alpha_k(r+1) = A_k(\alpha(r))$
  - **fin**
  - Aplicar el operador de agregación: $\alpha(r+1) = \Delta(\alpha_1(r+1), \ldots, \alpha_K(r+1))$.
  - Difundir $\alpha(r+1)$ a cada nodo.
- **fin**

fin

Cuando el objetivo es entrenar un modelo $\{f(x, \alpha) \colon \alpha \in \Lambda\}$, al final de cada ronda de

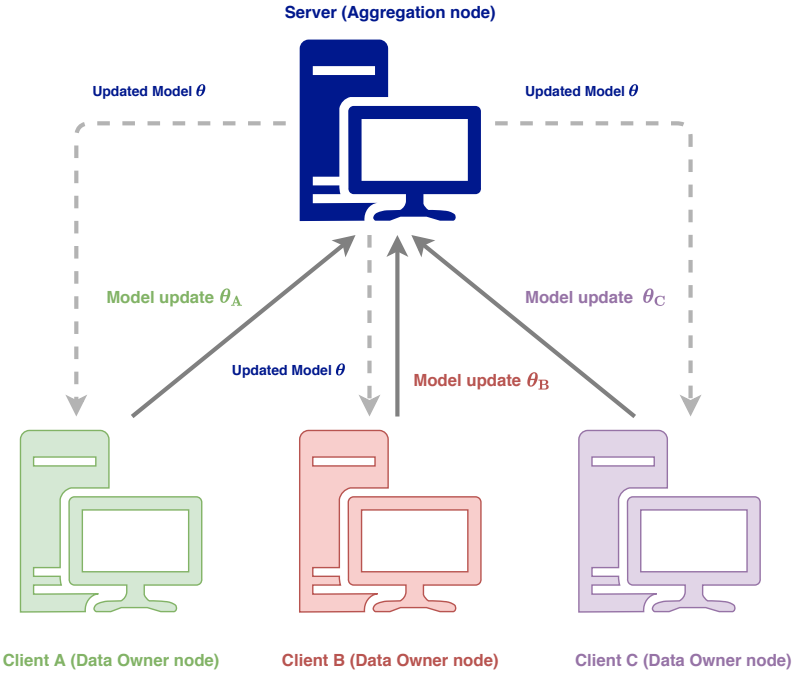


Figura 8.1.: Esquema del paradigma de aprendizaje federado ([26]).

aprendizaje podemos diferenciar dos situaciones. Por una parte, cada nodo k alberga la función determinada por sus parámetros: $f(x, \alpha_k)$. Por otra parte, el nodo central alberga el resultado de agregar esos parámetros: $f(x, \Delta(\alpha_1, \ldots, \alpha_K))$. El modelo central nunca es entrenado directamente, sino que sólo se actualiza a través del operador de agregación. En la Figura 8.1 se presenta un esquema del proceso que acabamos de presentar.

8.2. Federated Averaging

A continuación presentamos el algoritmo más aplicado en problemas de AF. Recibe el nombre de *federated averaging (FedAvg)* y consiste en tomar en el operador de agregación la media de los parámetros de cada cliente ([19]):

$$\Delta(\alpha_1, \ldots, \alpha_K) = \frac{1}{K} \sum_{k=1}^{K} \alpha_k$$

Una modificación popular consiste en ponderar cada parámetro usando $w_1, \ldots, w_K \in \mathbb{R}$ tales que $\sum_{k=1}^{K} w_k = 1$. Esta versión recibe el nombre de *Weighted Federated Averaging (WFedAvg)*:

$$\Delta(\alpha_1, \ldots, \alpha_K) = \sum_{k=1}^{K} w_k \alpha_k$$

Algoritmo 7: Federated averaging.

Entrada: $\mathcal{D}_1, \ldots, \mathcal{D}_K \subset \mathcal{Z}$, $w_1, \ldots, w_K \in \mathbb{R}$ tales que $\sum_{k=1}^{K} w_k = 1$, $\ell\colon \mathcal{Z} \times \Lambda \to \mathbb{R}$
inicio
  Inicializar $\alpha(0), \alpha_1(0), \ldots, \alpha_K(0)$
  para $r = 0, 1, 2, \ldots$ **hacer**
    para $k \in \{1, \ldots, K\}$ **hacer**
      Elegir un minibatch $B_k \subset \mathcal{D}_k$
      $\alpha_k(r+1) = \alpha_k(r) - \eta_k \sum_{z \in B_k} \nabla_\alpha \ell(z, \alpha_k(r))$
    fin
    Aplicar el operador de agregación: $\alpha(r+1) = \sum_{k=1}^{K} w_k \alpha_k(r+1)$.
    Difundir $\alpha(r+1)$ a cada nodo.
  fin
fin

8.3. Sherpa.ai Federated Learning

La última sección de este capítulo la dedicamos a *Sherpa.ai Federated Learning (Sherpa.ai FL)*, el cual es un framework libre[1] para AF. Fue presentado en 2020 en [26] y desarrollado en una colaboración entre el Instituto DaSCI[2] y Sherpa.ai[3]. Describimos brevemente los módulos de la librería señalando sus funcionalidades principales.

Cabe decir que los experimentos que se presentan se han desarrollado partiendo de este software. Por tal razón reservamos este espacio del trabajo para entender cómo está estructurado y cómo funciona.

8.3.1. Estructura

La librería usada se estructura en diversos módulos. Cada uno se encarga de encapsular una funcionalidad específica de la configuración federada que hemos presentado en este capítulo. En la figura Figura 8.2 se muestra un esquema con las relaciones existentes entre los módulos de Sherpa.ai FL, que pasamos a describir a continuación:

- Data base (`data_base`): este módulo se encarga de definir la clase `DataBase`, la cual provee las funcionalidades necesarias para trabajar con cualquier conjunto de datos. Además ofrece clases predefinidas para usar conjuntos de datos populares en la literatura como MNIST, Iris, ...
- Data distribution (`data_distribution`): se encarga de distribuir los ejemplos del conjunto de datos entre los clientes involucrados en el proceso de entrenamiento. Podemos definir la forma en que se realiza esta distribución.
- Private (`private`): provee diversas interfaces para definir la forma en que se accede a los datos de cada cliente, asegurando así la privacidad de los mismos.
- Federated Aggregator (`federated_aggregator`): implementa diversos operadores de agregación y permite definir otros nuevos.

[1] Sherpa.ai FL se puede descargar desde el siguiente repositorio de Github: https://github.com/sherpaai/Sherpa.ai-Federated-Learning-Framework
[2] https://dasci.es/
[3] https://sherpa.ai/

- Model (`model`): se encarga de definir la arquitectura del modelo que entrenará cada cliente y se agregará en el nodo central. Ofrece modelos predefinidos para distintos problemas, así como las interfaces necesarias para crear los nuestros.
- Differential Privacy (`differential_privacy`): a través de este modulo podemos preservar la privacidad de los clientes mediante diversas técnicas de privacidad diferencial.

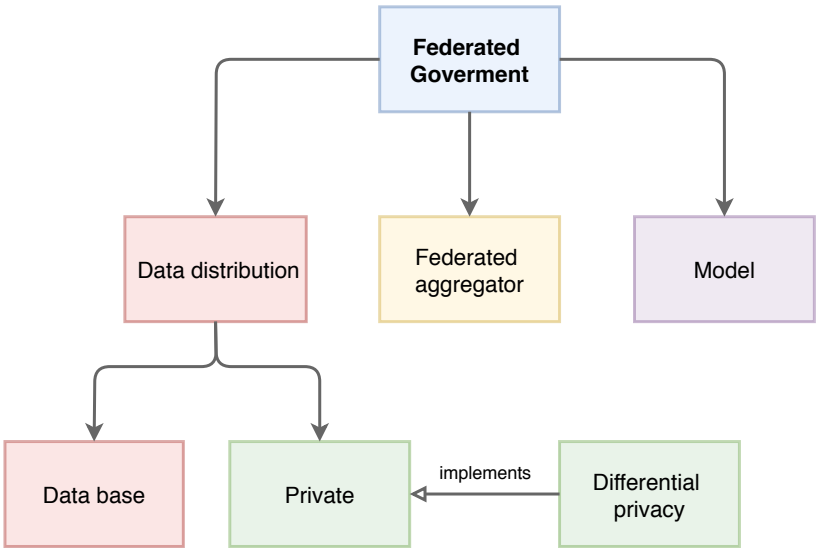


Figura 8.2.: Módulos de Sherpa.ai FL.

8.3.2. Funcionalidades

A continuación vamos a señalar qué cosas nos permite hacer esta librería, centrándonos en aquellas que más útiles han resultado en el desarrollo de este trabajo:

- Definir y customizar una simulación del proceso de AF con un número predeterminado de clientes o nodos.
- Observar la evolución del modelo central y de los modelos centrales en cada ronda de aprendizaje.
- Entrenar modelos de aprendizaje automático en cada cliente usando las herramientas de las librerías Keras[4], Tensorflow[5], Pytorch[6] y Scikit-Learn[7].
- Usar los operadores clásicos de agregación de parámetros (FedAvg, WFedAvg) o definir los nuestros propios.
- Comparar la aproximación federada a la centralizada.

Gracias a la implementación jerárquica del framework es posible extender las posibilidades de la librería sobreescribiendo las clases existentes o añadiendo otras nuevas.

[4] https://keras.io/
[5] https://www.tensorflow.org/
[6] https://pytorch.org/
[7] https://scikit-learn.org/

8.4. Desafíos y problemas en aprendizaje federado

Cuando llevamos el problema que acabamos de presentar a una situación real nos encontramos con diversos desafíos que debemos solventar. La realidad es que es un problema muy complejo en el que intervienen diversos factores. A continuación señalamos los más importantes.

8.4.1. IID vs Non-IID

Las distribuciones de probabilidad P_k son completamente desconocidas, sólo accesibles a través de los conjuntos de entrenamiento $\mathcal{D}_k$. Distinguimos dos tipos de escenarios:

- El escenario *idénticamente distribuido (IID)* es aquel en que todas las distribuciones coinciden.
- El escenario es *no idénticamente distribuido (Non-IID)* cuando no ocurre lo anterior.

En el escenario IID existe una distribución de probabilidad P tal que para todo nodo k se cumple $P_k = P$. En (8.1) recuperamos el problema de optimización original del aprendizaje, pero ahora los ejemplos de entrenamiento están repartidos en K conjuntos de entrenamiento distintos, a los que no podemos acceder simultáneamente.

El escenario Non-IID es el caso que se presenta con más frecuencia en la realidad, ya que los datos con lo que cuenta cada dispositivo son generados por él mismo y por tanto, la distribución de cada nodo será distinta. Uno de los mayores desafíos en AF consiste en obtener buenos resultados en este escenario.

Por otra parte, dado que cada cliente genera sus propios datos, los conjuntos pueden ser muy distintos. Puede haber clientes que aporten muchos datos, mientras que otros aporten muy pocos. Este será un aspecto a tener en cuenta a la hora de agregar los parámetros.

8.4.2. Comunicación entre nodos

El proceso de aprendizaje en la configuración federada comprende dos tipos de comunicaciones. Por una parte, cada nodo debe enviar sus parámetros actualizados al servidor central. Por otra parte, el servidor central debe difundir los parámetros agregados a cada nodo. Además, esto es algo que sucede en cada ronda de aprendizaje.

En la práctica las comunicaciones se realizan a través de conexiones de internet. Todos los problemas que encontramos frecuentemente en estas conexiones afectarán a la eficiencia del proceso de aprendizaje. Las comunicaciones conllevan un coste y el esquema de AF tratará de minimizar este coste. Sin embargo, se comprueba que la convergencia de los modelos a un resultado estable no sucede hasta que se llevan a cabo un número alto de rondas de aprendizaje.

Por otra parte, como hemos dicho, habrá ocasiones en que la inestabilidad de las conexiones entre clientes limitará las comunicaciones que podemos realizar. Puede ocurrir que a la hora de agregar parámetros no todos los clientes estén disponibles o haya que esperar por algunos de ellos. Por ello debemos diferenciar entre aprendizaje *síncrono* ([19]) y aprendizaje *asíncrono* ([36]). En este trabajo vamos a lidiar con un problema de naturaleza síncrona.

8.4.3. Seguridad y privacidad

El intercambio de parámetros entre los clientes y el servidor central supone un riesgo para la información que albergan los nodos. En [9] se demuestra cómo es posible reconstruir los ejemplos de entrenamiento a través de *ataques de inversión de modelo*. En [40] se muestra cómo es posible reconstruir ejemplos de entrenamiento a partir de la información de los gradientes. En [38] se muestran diversos métodos a través de los que se puede reconstruir la información presente en el conjunto de entrenamiento.

Así surge la necesidad de desarrollar técnicas para proteger los datos presente en los intercambios entre clientes. Una de ellas recibe el nombre de *privacidad diferencial*, o *differential privacy* en inglés ([8]), la cual establece rigurosamente una medida de la pérdida de privacidad que se sufre en los algoritmos aleatorios.

Parte IV.

Diagnosis de COVID-19 a partir de radiografías de tórax

9. Descripción del problema

Las técnicas de aprendizaje automático descritas en capítulos anteriores llevan varios años jugando un papel importante en el diagnóstico de enfermedades. Numerosas patologías se suelen detectar mediante técnicas como las radiografías o tomografías, cuya información se recoge en imágenes que posteriormente son analizadas por los profesionales de la medicina.

Las radiografías de tórax pueden servir para detectar la COVID-19. El tiempo en que un paciente puede ser evaluado mediante ellas es más bajo que el conocido test RT-PCR. Un modelo robusto y preciso de aprendizaje podría servir como un método de triaje y de apoyo a las decisiones de los doctores. Esto motiva el problema al que nos enfrentamos en este trabajo y que describimos a continuación.

9.1. La diagnosis de la COVID-19

Detectar a tiempo la enfermedad COVID-19 es el paso más importante para poder tratar la enfermedad con las técnicas que se conocen. Actualmente, la mayoría de mecanimos empleados con este fin con métodos serológicos, moleculares y virales [29]. Algunos métodos que se emplean actualmente son:

- Test RT-PCR: detecta la presencia de ácido nucleico viral (RNA) a partir de muestras respiratorias extraidas de los pacientes ([6]). Es el método más común debido a que proporciona resultados muy fiables. Sin embargo, requiere instrumentación específica y ofrece resultados a partir de 24 horas.

- Análisis de tomografías computadas (CT) de tórax: es un método rápido, fácil y altamente sensible a la hora de detectar la enfermedad ([37]). Este tipo de análisis pueden mostrar anormalidades pulmonares en pacientes que presentan un resultado de test RT-PCR negativo. Sin embargo, el equipamiento necesario no está disponible en la mayoría de hospitales.

- Análisis de radiografías de tórax (CXR): su principal ventaja es que el equipamiento necesario es mucho más accesible y transportable que el de otros métodos. Además, puede ofrecer resultados en menos de 15 segundos por paciente.

Diversos estudios aportan estimaciones de la sensibilidad de los diferentes métodos. En [35] se presenta un estudio en 64 pacientes con una sensibilidad del 69 % del mecanismo CXR y del 91 % para el test RT-PCR. Por su parte, [34] informa de que la mayoría de los pacientes de los centros de urgencias que padecían la enfermedad presentaban anormalidad en las radiografías de tórax; pero sólo el 58.3 % de ellos eran diagnosticados correctamente por el ojo humano.

En este trabajo exploramos el diagnóstico de la enfermedad a partir de radiografías de tórax. Dada una imagen de una radiografía de tórax, nuestro objetivo consiste en decidir si tal imagen corresponde a un paciente que padece la enfermedad COVID-19 o no. En el primer caso, diremos que se trata de un caso positivo, y le asignaremos la etiqueta P. En el

segundo caso, diremos que se trata de un caso negativo, y le asignaremos la etiqueta N. Este es un problema de clasificación binaria, que vamos a afrontar entrenando el modelo que se ha descrito en la Sección 7.6.

9.2. Conjunto de datos: COVIDGR-FL

El conjunto de datos con el que vamos a tratar está compuesto por 940 imágenes etiquetadas por radiólogos de cuatro hospitales: Universitario Clínico San Cecilio de Granada, Motril, Elche y Baza. Hay 490 imágenes de la clase negativa y 450 de la positiva.

A parte del diagnóstico de cada radiografía, de cada una se conoce la siguiente información:

- Clase observada por el radiólogo, que puede ser positivo o negativo.
- Clase test RT-PCR, que puede ser verdadero positivo (VP), verdadero negativo (VN), falso positivo (FP) o falso negativo (FN).
- Procedencia, que indica el centro del que procede la radiografía; y cuya distribución se encuentra en la Tabla 9.1

Procendencia	**Imágenes N**	**Imágenes P**
San Cecilio	449	334
Motril	29	68
Elche	8	24
Baza	2	19

Tabla 9.1.: Distribución de imágenes según la procedencia.

- Edad y sexo. La distribución de imágenes respecto a estas características se encuentra en la Tabla 9.2.

(Sexo, edad e)	**Imágenes N**	**Imágenes P**
(Hombre, $e \leq 50$)	112	60
(Mujer, $e \leq 50$)	166	62
(Hombre, $50 < e \leq 65$)	46	89
(Mujer, $50 < e \leq 65$)	79	69
(Hombre, $65 \leq e$)	57	101
(Mujer, $65 \leq e$)	30	69

Tabla 9.2.: Distribución de imágenes según el sexo y la edad.

- Puntos RALE, que es un entero que indica la gravedad de la enfermedad. Este valor varía de 0 (ausencia de evidencias de la enfermedad) a 8 (mayor gravedad posible). La distribución de imágenes respecto a los puntos RALE se encuentra en la Tabla 9.3.

Puntos RALE	Imágenes
0 PCR-	490
0 PCR+	79
1	49
2	53
3	68
4	50
5	63
6	48
7	25
8	15

Tabla 9.3.: Distribución de imágenes según los puntos RALE.

- Nivel RALE, el cual está asociado a los puntos RALE. Puede ser Normal PCR- (0 puntos RALE con PCR negativa), Normal PCR+ (0 puntos RALE con PCR positiva), Leve (entre 1 y 2 puntos RALE), Moderado (entre 3 y 5 puntos RALE) y Grave (entre 6 y 8 puntos RALE). En la Tabla 9.4 se muestra la distribución de imágenes respecto a estos niveles.

Nivel RALE	Imágenes
Normal PCR-	490
Normal PCR+	79
Leve	102
Moderado	183
Grave	86

Tabla 9.4.: Distribución de imágenes según los niveles RALE.

Cada una de las cuatro características anteriores (Edad y sexo, puntos RALE, nivel RALE y procedencia) configuran una partición de los datos del conjunto de entrenamiento. Estas particiones serán las que usaremos para plantear el experimento federado en el siguiente capítulo.

Basándonos en el conjunto que acabamos de describir, afrontaremos el problema en dos escenarios o fases:

1. Escenario centralizado, en el pretendemos encontrar un buen modelo de aprendizaje que estime la clase de cada imagen para nuevos ejemplos.
2. Escenario federado, en el que "federaremos" el problema. Esto quiere decir que repartiremos los datos en varios nodos de acuerdo a distintos criterios y ajustaremos un modelo usando técnicas de optimización federada.

Pretendemos comparar la potencia del mejor modelo obtenido en cada escenario, para lo cual usaremos las métricas que describimos en la Sección 9.3. En el siguiente capítulo abordaremos el entrenamiento y la experimentación.

9.3. Medidas de error

Para evaluar el modelo ajustado usaremos las siguientes medidas de error o métricas, calculadas sobre el conjunto de datos de test de COVIDGR-FL. Para describirlas vamos a denotar por $\mathcal{D}$ al conjunto de datos y por h a la función de etiquetado que implementa el modelo. Llamamos:

$$TP = |\{(x,y) \in \mathcal{D} \colon h(x) = P, y = P\}|$$
$$TN = |\{(x,y) \in \mathcal{D} \colon h(x) = N, y = N\}|,$$

al número de verdaderos positivos y verdaderos negativos, respectivamente. Para cada clase $c \in \{P, N\}$ definimos:

$$\text{Recall}(c) = \frac{Tc}{|\{(x,y) \in \mathcal{D} \colon y = c\}|}$$
$$\text{Precision}(c) = \frac{Tc}{|\{(x,y) \in \mathcal{D} \colon h(x) = c\}|}$$
$$\text{F1}(c) = 2\frac{precision(c)recall(c)}{precision(c) + recall(c)}$$

Obsérvese que Recall(c) hace referencia a la proporción de ejemplos de la clase c que el modelo identifica correctamente. Recall(P) también recibe el nombre de sensibilidad o *sensivity* en inglés, mientras que a Recall(N) se le llama especificidad o *specificity*.

En problemas de diagnósticos médicos la sensibilidad y especificidad son especialmente importantes. Si un método de diagnóstico que tiene una sensibilidad muy alta califica a un paciente como negativo, podremos descartar la enfermedad. De la misma forma, si un método tiene una especificidad muy alta, raramente calificará a un paciente que que no padece la enfermedad como positivo.

Una de las medidas más importantes es:

$$\text{Accuracy} = \frac{TP + TN}{|\mathcal{D}|} = \frac{|\{(x,y) \in \mathcal{D} \colon h(x) = y\}|}{|\mathcal{D}|},$$

la cual mide la proporción de ejemplos del total que el modelo clasifica correctamente. Esta métrica se puede usar para evaluar el comportamiento global del modelo. Sin embargo, si en el conjunto la proporción de ejemplos negativos y positivos es muy diferente no podemos confiar en ella.

10. Experimentación

En este último capítulo relatamos el proceso por el cual se ha entrenado el modelo COVIDSDNET (que fue descrito en la Sección 7.6) en los dos escenarios planteados.

La primera parte de este capítulo se centra en describir el software que se ha desarrollado como medio para evaluar los modelos que hemos descrito en el trabajo. Este software es, fundamentalmente, el resultado de implementar y adaptar a cada escenario todas las etapas de las que se compone COVIDSDNET.

La segunda parte de este capítulo se centra en presentar, analizar y comparar los resultados de cada uno de los escenarios. Es importante señalar que COVIDSDNET proporciona dos clasificadores: el del preprocesado con FUCITNET y el de la última etapa de inferencia. Para referirnos al primero simplemente escribiremos *clasificador de FUCITNET* y para el segundo *clasificador de SDNET*. La precisión y capacidad de predecir la diagnosis de un paciente se encuentra en estos clasificadores.

Las tablas con los resultados se pueden consultar en el Apéndice A. En particular, en las tablas A.1, A.2 y A.3 se encuentran recogidos todos los datos para facilitar su comparación.

10.1. Software desarrollado

Para poder realizar los experimentos que describimos en las siguientes secciones se ha desarrollado el paquete SDNETLearning, que encapsula todas las funcionalidades que presenta la metodología SDNET. En particular, contiene los elementos necesarios para realizar el preprocesado (recorte) de pulmones, entrenar un modelo FUCITNET para un problema de clasificación arbitrario y entrenar el clasificador de SDNET para un problema de clasificación binaria.

Se encuentra disponible en Github[1] bajo la licencia GNU General Public License. En esta sección describimos las herramientas utilizadas y los componentes del propio paquete.

10.1.1. Herramientas utilizadas

El software ha sido desarrollado en Python 3.6.0. Este es un lenguaje interpretado que destaca por su versatilidad e intuitividad. Uno de los elementos que motivan la elección de este lenguaje es la existencia de bibliotecas para trabajar con redes neuronales profundas como Pytorch y Tensorflow.

Pytorch se define a sí misma como una librería dedicada al cálculo de operaciones con tensores para aprendizaje profundo, optimizada para ser usada en GPUs y CPUs ([2]). Esta librería destaca por su gran libertad a la hora de implementar cualquier tipo de red.

Tensorflow es una plataforma creada por el equipo de Google Brain para problemas de aprendizaje automático. Como Pytorch, proporciona funcionalidades para cálculo de tensores. Tensorflow destaca porque puede llegar a ser muy flexible a través de las APIs que ofrece.

[1] https://github.com/Franblueee/TFGCOVID

Una de estas APIs es Keras, que aporta un nivel de abstracción a la hora de construir modelos de aprendizaje profundo ([3]).

Pytorch ha sido usada para crear la implementación de FUCITNET partiendo de la existente en el repositorio original[2]. Tensorflow se ha usado para crear el clasificador de la última etapa de SDNET.

Por otra parte se encuentra la librería Sherpa.ai-FL (de ahora en adelante shfl) que fue descrita en el Capítulo 8, y que nos aporta los elementos necesarios para crear escenarios federados.

Junto a estas, se han empleado otras librerías de Python para multitud de fines. El lector interesado puede consultar el listado de dependencias en el repositorio de Github.

10.1.2. SDNETLearning

Para proveer las funcionalidades de FUCITNET y SDNET el paquete desarrollado proporciona las siguientes clases:

- `CITmodel`: implementa las herramientas necesarias para entrenar (método `train`), evaluar (método `evaluate`) el modelo FUCITNET y transformar un conjunto de datos usando los generadores (método `transform_data`).
- `ClassifierModel`: se usa para para entrenar (método `train`) y evaluar (método `evaluate`) el clasificador de SDNET. Esta clase hereda de la clase `DeepLearningModel` de shfl.
- `LungsCropper`: se encarga de realizar la segmentación y el recorte de los pulmones a partir de un modelo preentrenado. Para ello se usan los métodos `cropImage`, `crop_directory` y `crop_data`.
- `SDNETmodel`: basándose en las anteriores clases implementa la metodología SDNET. A parte de las funciones necesarias para entrenar y evaluar un modelo, permite realizar la lectura de datos desde un directorio y un archivo `.csv` con el método `get_data_csv`.
- `FederatedSDNETModel`: hereda de la clase SDNETmodel para proporcionar la "versión federada"de SDNET. Para ello aporta los métodos `get_federated_data_csv` (para leer la partición de datos desde un archivo `.csv`) y `run_rounds` (para entrenar el modelo SDNET durante un número de rondas determinado).

En la Figura 10.1 se encuentra el diagrama de clases con los métodos y dependencias de cada clase.

Como hemos señalado, las clases `SDNETmodel` y `FederatedSDNETModel` presentan métodos para realizar la lectura del conjunto de datos. Para ello, será necesario proporcionar:

1. La ruta del directorio `dir` donde se encuentran las imágenes debidamente etiquetadas. Para etiquetar las imágenes es necesario crear los subdirectorios `dir/N` y `dir/P` y colocar cada imagen en el directorio de su clase correspondiente.
2. Un fichero `.csv` donde se especifica la partición de las imágenes. El fichero debe tener las columnas `name` (nombre de la imagen), `class` (clase de la imagen) y `set` (train o test). Cada fila del fichero hace referencia a una imagen. Para la versión federada de SDNET deberá existir una cuarta columna `node` donde se indica el nodo al que pertenece cada imagen.

[2] https://github.com/manurare/Class-Inherent-Transformations

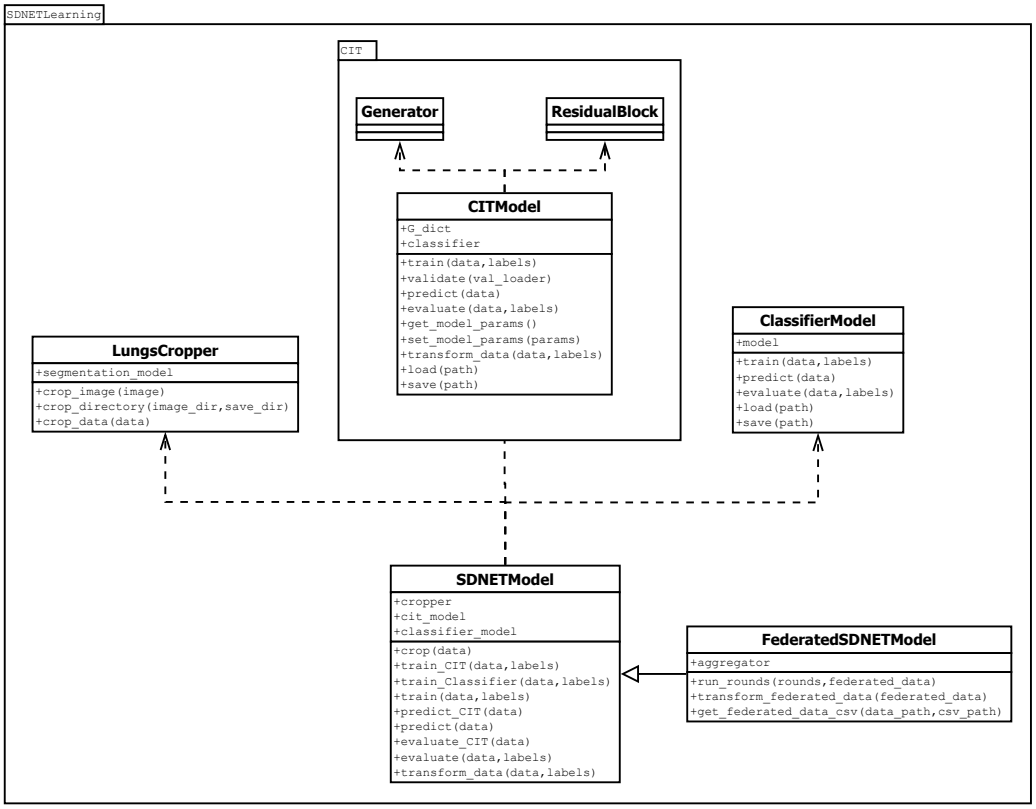


Figura 10.1.: Diagrama de clases de SDNETLearning.

Junto al paquete SDNETLearning se proporcionan los ficheros Python que se han usado para reproducir los experimentos. Estos ficheros constituyen una pequeña aplicación que permite entrenar y evaluar instancias de SDNET modificando sólo un fichero `.json`.

10.2. Escenario centralizado

Para entrenar y evaluar el comportamiento de ambos clasificadores se ha llevado a cabo un proceso de validación cruzada en tres capas, manteniendo una proporción de datos de 80/20. Esto es, hemos creado tres particiones distintas de los datos, dividiendo en cada una el conjunto en *train* (752 imágenes, el 80 %) y *test* (188 imágenes, el 20 %). Para cada partición, se entrena el modelo sobre el conjunto *train* y se evalúa en el de *test*. Este proceso se ha repetido tres veces con cada partición, completando en total 9 ejecuciones. Todos los resultados que se presentan en esta sección se obtienen mediante el proceso anteriormente descrito y calculando el valor medio de todas las ejecuciones.

Los generadores de FUCITNET parten de parámetros aleatorios mientras que el clasificador está preentrenado en Imagenet y se ajustan todas las capas. El algoritmo de optimización usado para ambos tipos de red es Adam [13]. La tasa de aprendizaje para el clasificador es 10^{-3} y para los generadores es 10^{-4}. Para evitar sobreajuste en el clasificador se añade un término de decaimiento de pesos con factor de 10^{-4}. Cada 5 épocas las tasas de aprendizaje

se multiplican por 0.1. El tamaño de batch se fija a 8 y el entrenamiento continúa durante 50 épocas, con early stopping de 10 épocas de paciencia. En el entrenamiento de FUCITNET se añaden imágenes usando técnicas de aumento de datos (rotaciones aleatorias).

En cuanto al clasificador de SDNET, también es preentrenado en Imagenet y se ajustan todas las capas. Como optimizador usamos SGD con tasa de aprendizaje fija de 10^{-4}. El tamaño de batch es también 8 y se entrena durante 50 épocas, con early stopping de 10 épocas de paciencia. No se usa ninguna forma de regularización en la función de pérdida ni técnicas de aumento de datos.

10.2.1. Elección del parámetro λ

Uno de los pasos más importantes para poder entrenar los modelos es elegir el mejor valor posible del parámetro λ de FUCITNET (7.5). La experimentación de [25] confirma que este parámetro debe ser elegido cuidadosamente en cada problema, y eso es lo que nosotros hacemos aquí.

Elegimos varios valores del parámetro y evaluamos el entrenamiento usando cada uno de ellos. Para no sesgar los resultados finales, evitamos usar el conjunto de *test* para evaluar. Elegimos uno de los conjuntos de entrenamiento y realizamos validación cruzada de 5 capas.

En Tabla 10.1 se recopila el valor de accuracy y de la función de pérdida del clasificador de FUCITNET. En Tabla 10.2 se encuentran los valores de accuracy del clasificador de SDNET.

Como vemos, los mejores valores difieren para uno y otro clasificador, pero son prometedores dada la gran cantidad de imágenes que estamos dejando fuera en cada entrenamiento. Dado que nuestra intención es obtener el mejor comportamiento posible del clasificador SDNET, fijamos $\lambda = 0.05$ de ahora en adelante.

λ	**Accuracy**	**Loss**
0.1	0.75313	0.59971
0.05	0.75491	0.58469
0.01	0.74295	0.60552
0.001	0.75508	0.62740
0.0001	0.75547	0.61103
1^{-5}	0.74334	0.63030
1^{-6}	**0.75743**	**0.56611**
1^{-7}	0.74178	0.68018
1^{-8}	0.73513	0.63232
1^{-9}	0.74882	0.60802

Tabla 10.1.: Clasificador FUCITNET.

λ	**Accuracy**$_2$	**Accuracy**$_4$
0.1	0.73709	0.715574
0.05	**0.75351**	0.73151
0.01	0.73082	0.73356
0.001	0.72809	0.72066
0.0001	0.73591	0.72124
1^{-5}	0.74061	0.73034
1^{-6}	0.73083	0.71596
1^{-7}	0.72495	0.72065
1^{-8}	0.74100	**0.74618**
1^{-9}	0.72144	0.70510

Tabla 10.2.: Clasificador SDNET.

10.2.2. Análisis de resultados

Los resultados de este escenario centralizado se muestran en la tabla Tabla A.4. Observamos que FUCITNET es ligeramente más preciso que SDNET y se comporta mejor al predecir imágenes de la clase N. En contraposición, el clasificador de SDNET funciona mejor sobre

la clase P. Conviene recordar que el conjunto de datos en que estamos entrenando está ligeramente desequilibrado hacia la clase N, habiendo 40 imágenes más.

Comparemos estos números con los que figuran en el trabajo original [32]. El conjunto de datos que se usó originalmente se llama COVIDGR1.0 y cuenta con 426 imágenes de cada clase (852 en total). Por tanto, tiene 88 imágenes menos y sí que está equilibrado. Esto puede explicar que FUCITNET obtenga peores resultados con un valor de accuracy de 0.7435.

Por otra parte, el clasificador SDNET sí que obtiene resultados muy parecidos en cada una de las métricas, lo cual puede llevarnos a pensar que SDNET es un modelo más robusto. Confirmaremos este hecho en el escenario federado, en el que cada nodo trabaja con conjuntos distintos.

10.3. Escenario federado

A continuación planteamos la versión federada de nuestro problema. Cuando tenemos un conjunto de datos, para crear un escenario federado simplemente necesitamos decidir un número de nodos y una forma de repartir los datos entre los nodos. Obsérvese que al repartir los datos vamos a inducir una distribución de probabilidad en cada nodo.

Estudiaremos dos casos o configuraciones. Empezaremos por el caso en que los datos se distribuyen de forma independiente e idéntica entre un número de nodos prefijado. Diremos que esta es la *configuración IID* (siglas en inglés de datos independientes e idénticamente distribuidos). Cuando la hayamos estudiado, pasaremos a hablar de la *configuración noIID* en la cual crearemos diferentes particiones de los datos basándonos en las características del conjunto COVIDGR-FL.

10.3.1. Esquema de la experimentación

En cuanto al entrenamiento que realiza cada nodo, son válidas las mismas consideraciones que hacíamos para el escenario centralizado. De hecho, en el paradigma de aprendizaje federado, cada nodo entrena sobre su conjunto de datos observando un enfoque centralizado, con la única diferencia de que al terminar cada ronda de entrenamiento debe compartir sus parámetros con el resto.

Los experimentos de esta sección responden al siguiente esquema de aprendizaje federado:

1. Como el modelo que se usa para segmentar los pulmones es preentrenado y no necesita entrenamiento sobre el conjunto de datos nuevo, se parte de las imágenes originales y se segmentan. Las imágenes recortadas se dividen en *train* y *test* y las de *train* se reparten entre los nodos considerados. Se obtienen así los conjuntos de entrenamiento de cada nodo: $\mathcal{D}_1, \ldots, \mathcal{D}_K$.

2. Se entrena el modelo FUCITNET siguiendo el esquema de aprendizaje federado: en cada ronda de aprendizaje, se entrena el par

$$\textit{(Generadores, Clasificador)} = (G, C) = ([G_P, G_N], C)$$

   en cada nodo. Al final de cada ronda, se agregan los parámetros del par (G, C), de forma que cada nodo parte de los mismos pesos en la siguiente ronda.

3. Cuando ha finalizado el entrenamiento de FUCITNET, todos los nodos albergan el mismo modelo (G, C) pues se ha realizado la última agregación. Con este modelo

global, se transforman las imágenes de cada nodo: para cada ejemplo (x, y), se genera un par de imágenes: $G_P(x)$, $G_N(x)$ correspondientes a las transformaciones de cada generador. Se sustituye el conjunto de entrenamiento de cada nodo por las imágenes del conjunto de entrenamiento anterior transformadas:

$$\mathcal{D}_k^T = \{(G_P(x), NTP) \colon (x, N) \in \mathcal{D}_k\} \cup \{(G_N(x), NTN) \colon (x, N) \in \mathcal{D}_k\} \cup \\ \{(G_P(x), PTP) \colon (x, P) \in \mathcal{D}_k\} \cup \{(G_N(x), PTN) \colon (x, P) \in \mathcal{D}_k\}.$$

4. En cada nodo k se entrena el clasificador de SDNET sobre el conjunto $\mathcal{D}_k^T$ siguiendo nuevamente el esquema de aprendizaje federado.

5. Los clasificadores de FUCITNET y SDNET obtenidos tras la última agregación de parámetros se evalúan sobre el conjunto de test.

Tras estudiar diversos valores, se ha decidido fijar el número de rondas en 30 para FUCITNET y en 50 para SDNET. Cada ronda de FUCITNET se entrena durante 10 épocas, mientras que cada ronda de SDNET se entrena durante 20 épocas. En cada caso justificaremos esta elección ayudándonos de gráficas de convergencia.

Para agregar los parámetros de los nodos usaremos los operadores FedAvg y WFedAvg. Para WFedAvg tomaremos como pesos la proporción de datos de cada nodo. Es decir, si n_k es el número de datos en el nodo k y $n = n_1 + \cdots + n_K$ es el número de datos total, se tomará $w_k = n_k/n$.

Conduciremos cada experimento con ambos agregadores para compararlos. Debemos tener en cuenta que si los datos están equilibrados (como en la configuración IID) tendrán el mismo impacto. El comportamiento de WFedAvg será especialmente interesante en aquellas configuraciones en las que el número de datos de los nodos no está equilibrado.

10.3.2. Configuración IID

10.3.2.1. Reparto de datos

Para que todos los nodos tengan la misma distribución repartimos los datos de la siguiente forma. Supongamos que K es el número de nodos:

1. Dividimos el conjunto de datos en *train* (752 imágenes, el 80 %) y *test* (188 imágenes, el 20 %).

2. El conjunto de entrenamiento se distribuye entre los nodos. Para cada ejemplo (x, y) del conjunto de entrenamiento (ordenado de forma aleatoria) se escoge de forma aleatoria un número $k \in \{1, \ldots, K\}$ y se añade el ejemplo (x, y) al conjunto $\mathcal{D}_k$ de ejemplos del nodo k.

Como en el escenario centralizado, realizamos validación cruzada. Es decir, el proceso anterior se repite 3 veces para cada valor de K obteniendo tres particiones del conjunto de datos. Para realizar los experimentos, se ha tomado $K = 3$, $K = 6$ y $K = 9$. El número de datos de cada nodo en cada partición figura en la Tabla 10.3. Como no podía ser de otra forma, el reparto que hemos hecho conduce a que el número de datos en cada nodo esté equilibrado.

K	Nodo 0	Nodo 1	Nodo 2	Nodo 3	Nodo 4	Nodo 5	Nodo 6	Nodo 7	Nodo 8
3	256	248	248						
	244	264	244						
	256	261	235						
6	129	131	114	115	135	128			
	118	125	122	126	125	136			
	123	125	126	135	122	121			
9	90	78	90	71	81	91	78	96	77
	78	88	87	84	81	87	82	88	77
	94	90	87	84	74	85	72	84	82

Tabla 10.3.: Configuración IID, número de ejemplos del conjunto de entrenamiento en cada nodo.

10.3.2.2. Análisis de resultados

Los resultados del entrenamiento en esta configuración se muestran en las tablas A.5, A.6 y A.7. Un resumen de los valores de accuracy se muestra en la Tabla 10.4. En la Figura 10.2 se representa esta misma información en un gráfico de columnas.

Podemos observar una tendencia descendente en la potencia de los modelos conforme aumenta el número de nodos. Esto se debe a que cuantos más nodos tenemos en cuenta, menos datos hay en cada uno. Tener un conjunto de datos suficientemente grande es un hecho que debemos tener presente en cualquier problema de aprendizaje y, como vemos, en el aprendizaje federado también se cumple esta premisa.

Obsérvese que cuando $K = 9$ los nodos están entrenando con menos de la mitad de datos que cuando $K = 3$ y, sin embargo, se siguen obteniendo resultados aceptables que sólo se alejan dos puntos porcentuales del escenario centralizado. En estos detalles es donde se muestra la potencia del paradigma federado; y es que, con muy pocos datos, somos capaces de obtener muy buenos resultados.

Asimismo, debemos señalar el buen comportamiento de FUCITNET. Obtiene siempre mejores resultados que SDNET y con 6 nodos los resultados son mejores que con 3 nodos. Podemos ver cómo la reducción en el número de datos afecta mucho menos a FUCITNET que al clasificador de SDNET.

	FUCITNET	SDNET
Centralizado	**0.78618**	0.76183
IID 3 nodos	**0.77128**	0.75532
IID 6 nodos	**0.77896**	0.748818
IID 9 nodos	**0.769276**	0.74668

Tabla 10.4.: Resultados obtenidos en el escenario IID expresados en términos de accuracy.

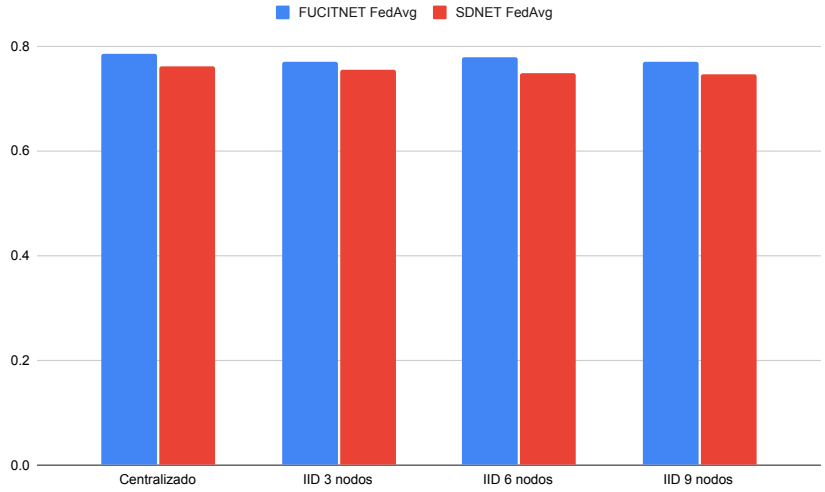


Figura 10.2.: Gráfico de columnas de los resultados obtenidos en el escenario IID expresados en términos de accuracy.

10.3.3. Configuración noIID

10.3.3.1. Reparto de datos

Como ya hemos adelantado, para repartir los datos en la configuración noIID nos basaremos en las distintas características de las imágenes de COVIDGR-FL, las cuales ya han sido presentadas en el capítulo anterior. Repartiremos los datos de acuerdo a cuatro criterios: según centro de procedencia, según datos demográficos (edad y sexo), según los puntos RALE y según los niveles RALE.

Para crear cada partición, primero se dividen las imágenes en nodos según el criterio considerado y posteriormente se extrae aleatoriamente el 20 % de imágenes de cada nodo para formar el conjunto de *test*. Se obtienen así los siguientes tipos de partición:

- **Hospitales:** es el tipo obtenido al dividir los ejemplos según el centro de procedencia. Se consideran cuatro nodos, correspondiendo cada uno a un hospital distinto. Esta partición representa una situación que podría darse en la realidad: varios hospitales colaboran para entrenar un método de diagnóstico.

Nodo	N	P
San Cecilio	346	280
Motril	23	54
Elche	8	17
Baza	2	14

Tabla 10.5.: Número de imágenes presentes en cada nodo en una de las particiones por hospitales.

En la tabla Tabla 10.5 observamos que el hospital de San Cecilio es claramente predo-

minante y existen nodos cuyos conjuntos están muy desequilibrados. Esto representa una situación real en que clientes con un conjunto de datos pobre se ven beneficiados por clientes con más información.

- **Demográfica:** es la obtenida al considerar datos demográficos (edad y sexo).

Nodo	N	P
(Hombre, $e \leq 50$)	87	50
(Mujer $e \leq 50$)	133	49
(Hombre, $50 < e \leq 65$)	34	74
(Mujer, $50 < e \leq 65$)	68	50
(Hombre, $65 \leq e$)	45	81
(Mujer, $65 \leq e$)	24	55

Tabla 10.6.: Número de imágenes presentes en cada nodo en una de las particiones por datos demográficos.

- **Puntos RALE:** se obtiene al dividir las imágenes según los puntos RALE. Si n_i es el número de imágenes positivas del nivel de puntos RALE i, n_P es el número de imágenes positivas y n_N es el número de imágenes negativas, el nodo i está formado por las imágenes positivas del nivel de puntos RALE i junto con $\frac{n_i}{n_P} n_N$ imágenes negativas elegidas aleatoriamente.

Nodo	N	P
0	62	59
1	53	41
2	70	39
3	56	52
4	44	39
5	48	49
6	31	42
7	16	21
8	15	10

Tabla 10.7.: Número de imágenes presentes en cada nodo en una de las particiones por puntos RALE.

- **Niveles RALE:** obtenida al repartir los ejemplos según los niveles RALE. Si n_i es el número de imágenes positivas del nivel de gravedad RALE i, n_P es el número de imágenes positivas y n_N es el número de imágenes negativas, el nodo i está formado por las imágenes positivas del nivel RALE i junto con $\frac{n_i}{n_P} n_N$ imágenes negativas elegidas aleatoriamente.

Nodo	N	P
Normal	63	57
Leve	118	87
Moderado	153	139
Grave	67	65

Tabla 10.8.: Número de imágenes presentes en cada nodo en una de las particiones por niveles RALE.

En los dos últimos tipos es necesario añadir imágenes negativas a cada nodo para que el aprendizaje pueda tener efecto. Se ha elegido añadir tal cantidad para que los nodos estén más o menos equilibrados.

Para poder realizar validación cruzada, para cada tipo de partición se crean tres particiones distintas, en las que el modelo es entrenado para su posterior evaluación.

10.3.3.2. Análisis de resultados

En la Tabla 10.9 se presenta un cuadro con un resumen de los valores de accuracy obtenidos por cada modelo en cada partición. En la Figura 10.3 se representa esta misma información de en un gráfico de columnas. Podemos observar que casi todos los valores del escenario noIID se encuentran varios puntos porcentuales por debajo del escenario centralizado y del escenario IID. Este es un comportamiento que esperábamos pues el reparto de datos no arbitrario entre nodos debe comprometer el comportamiento del modelo.

Debemos señalar, sin embargo, que la partición con puntos RALE obtiene resultados muy cercanos al escenario IID. De hecho, SDNET FedAvg se comporta como en el caso centralizado.

Por otra parte, y al contrario de lo que podríamos pensar, el uso de WFedAvg no siempre supone una mejora. Hay casos que sí lo es, como por ejemplo el reparto por hospitales, y otro donde el uso de este operador conduce a los peores resultados de todo el trabajo (caso del reparto por niveles RALE).

	FUCITNET FedAvg	SDNET FedAvg	FUCITNET WFedAvg	SDNET WFedAvg
Centralizado	**0.78618**	0.76182		
IID 3 nodos	**0.77128**	0.75532		
IID 6 nodos	**0.778960**	0.748818		
IID 9 nodos	**0.769276**	0.746680		
noIID Hospitales	0.72840	0.72369	**0.73369**	0.72957
noIID Demográfica	0.71579	**0.73684**	0.72164	0.72807
noIID Puntos RALE	0.74004	**0.76318**	0.75622	0.73895
noIID Niveles RALE	0.71670	0.71379	0.70506	**0.72251**

Tabla 10.9.: Resultados obtenidos en cada escenario expresados en términos de accuracy.

A continuación nos vamos a detener en los aspectos más interesantes de cada una de las particiones.

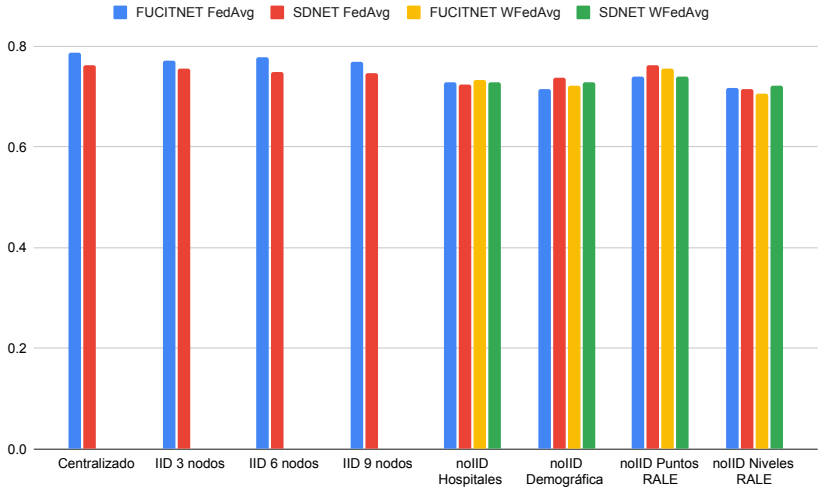


Figura 10.3.: Gráfico de columnas de los resultados obtenidos en cada escenario expresados en términos de accuracy.

Hospitales

Los resultados de la partición por hospitales se encuentran en las tablas A.8 y A.9. Como esperábamos, todas las métricas se encuentran por debajo tanto del escenario centralizado como del escenario IID.

FUCITNET se sitúa ligeramente por encima del clasificador de SDNET en cuanto a accuracy con ambos operadores. Con FedAvg este último presenta mayor sensibilidad, pero al usar WFedAVg las tornas cambian. Como vemos, FedAvg reduce las diferencias entre ambos clasificadores, pero los mejores resultados se consiguen al agregar parámetros con WFedAvg.

A pesar de que los nodos de Elche y Baza apenas tienen datos, hemos conseguido un buen error de generalización. Estos nodos se han beneficiado de las imágenes presentes en San Cecilio y en Motril. Al usar WFedAvg Elche y Baza apenas están presentes en el entrenamiento, y el mejor comportamiento de este operador puede indicar que nodos con tan pocos datos influyen negativamente en el aprendizaje.

Demográfica

Los resultados se presentan en las tablas A.10 y A.11 y se encuentran en la misma línea que en la partición anterior, pero apreciamos pequeñas diferencias.

En esta ocasión los mejores valores de accuracy los obtiene el clasificador de SDNET, que es mejor usando los dos operadores. De hecho el mejor valor de accuracy lo obtiene SDNET agregando con FedAvg, al contrario que en la partición por hospitales, que era mejor FUCITNET con WFedAvg.

Otro detalle importante es que WFedAvg no resulta beneficioso para SDNET, pero sí para FUCITNET. No siempre tener en cuenta el aporte de cada nodo en términos de número de datos supone una mejora.

La partición demográfica está compuesta por seis nodos, de los cuales cinco apenas albergan más de 110 imágenes. Aún así el esquema federado nos permite obtener resultados

competitivos.

Puntos RALE

Los resultados para esta partición se presentan en las tablas A.12 y A.13 y se encuentran al mismo nivel que las particiones IID.

En primer lugar, cabe destacar que, nuevamente, WFedAvg mejora a FUCITNET pero no a SDNET, que pierde hasta tres puntos porcentuales al usarlo. El mejor modelo es otra vez SDNET, aunque FUCITNET presenta valores de sensibilidad muy interesantes (superiores a 0.8).

Vamos a señalar el comportamiento de SDNET con FedAvg que obtiene una puntuación de accuracy de 0.763, especificidad de 0.783 y sensibilidad de 0.744. Estas cantidades son cercanas a las que presentamos en el escenario centralizado. Es importante tener en cuenta que hemos obtenido estos resultados usando ocho nodos, de los cuales sólo tres tienen más de cien imágenes.

Niveles RALE

El mejor modelo, en términos de accuracy, es SDNET con el operador WFedAvg. Esta partición es la única en la que FUCITNET no se ve beneficiado del operador WFedAvg. De hecho, usando tal operador obtenemos los peores resultados de todo el trabajo.

Por otro lado debemos destacar que FUCITNET obtiene valores de sensibilidad muy altos, mientras que si hablamos de especificidad SDNET presenta mejor puntuación en ambos casos. Llama la atención también los resultados tan bajos de la métrica Precision(P) que encontramos en FUCITNET.

Los resultados, que se encuentran en las tablas A.14 y A.15, son muy diferentes a los anteriores. Esto nos alerta especialmente ya que los criterios que hemos usado para repartir los datos son muy parecidos. De hecho, esta partición puede obtenerse de la anterior agrupando algunos nodos en uno sólo.

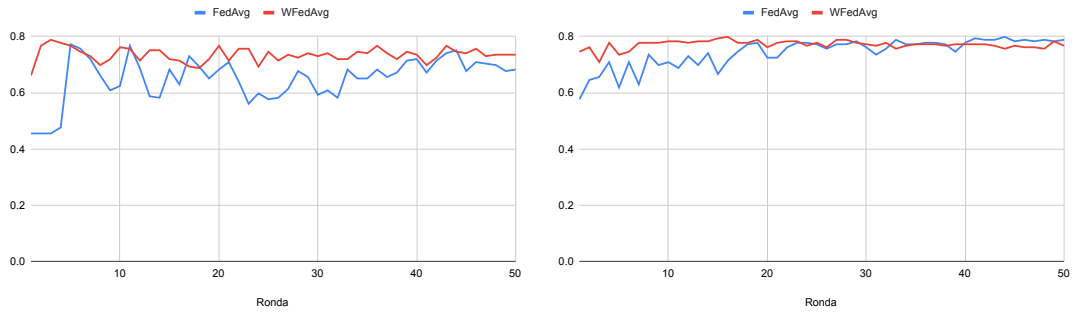
10.3.4. Convergencia

Dedicamos esta sección a analizar la convergencia de los operadores de agregación que hemos usado. Para ello, vamos a representar cómo evolucionan distintos valores de las métricas consideradas a lo largo del entrenamiento. Las gráficas que se presentan a continuación fueron recogidas para decidir y justificar el número de rondas y épocas que debíamos fijar.

Empecemos fijándonos en las figuras 10.4, 10.5 y 10.6. En ellas se representa la evolución de accuracy de FUCITNET y del clasificador de SDNET a lo largo de diversos entrenamientos de 50 rondas.

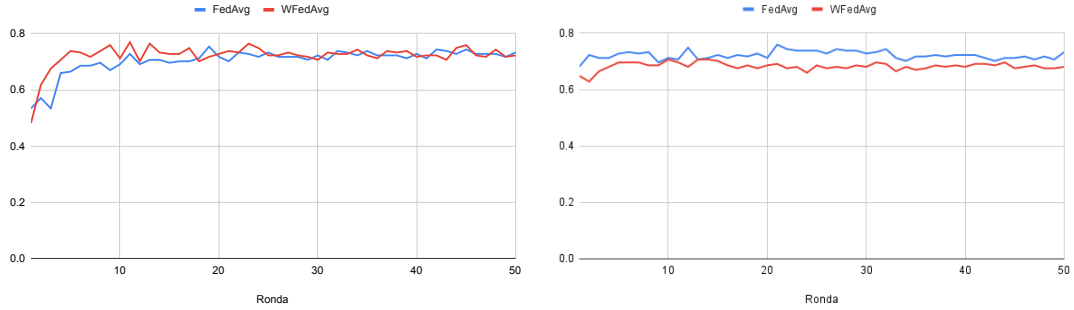
En la partición por hospitales, en la que WFedAvg aporta mejores resultado, la curva de este operador es mucho más estable y la convergencia acontece antes. Esto no se cumple siempre, y un ejemplo de ello es el comportamiento del clasificador de SDNET en las dos particiones restantes.

Por otra parte, en las primeras rondas la curva de WFedAvg suele encontrarse por encima de forma que la convergencia parece acontecer antes. También nos llama la atención que el entrenamiento de SDNET comience siempre desde valores más altos de la métrica. Este hecho queda explicado por el preprocesado "inteligente"que se realiza al transformar las imágenes con los generadores



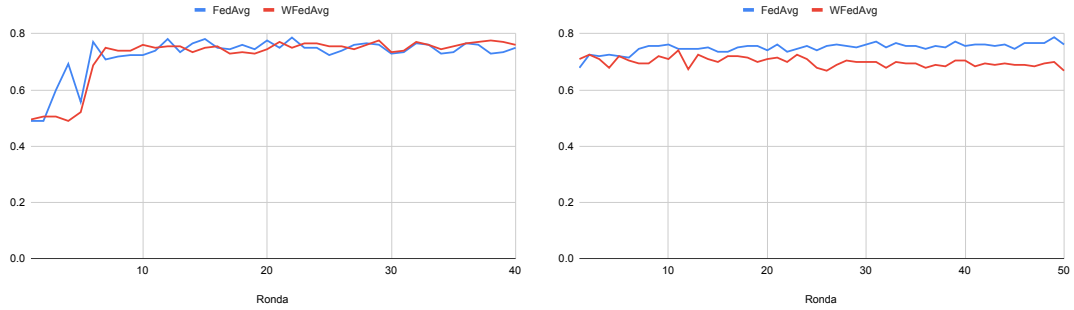
(a) FUCITNET. (b) SDNET.

Figura 10.4.: Evolución de accuracy en la partición por hospitales.



(a) FUCITNET. (b) SDNET.

Figura 10.5.: Evolución de accuracy en la partición por niveles RALE.



(a) FUCITNET. (b) SDNET.

Figura 10.6.: Evolución de accuracy en la partición por puntos RALE.

Por último vamos a analizar una de las funciones de pérdida. En la Figura 10.7 se representa la gráfica de la evolución de la función de pérdida en la misma ejecución que la Figura 10.5 (a). Podemos observar cómo en las primeras rondas el valor de la función decrece, pero llega

una ronda en que crece indefinidamente. De hecho, en el caso de WFedAvg el crecimiento es mucho más acusado, lo que puede sugerir que se llega antes a una situación de sobreajuste.

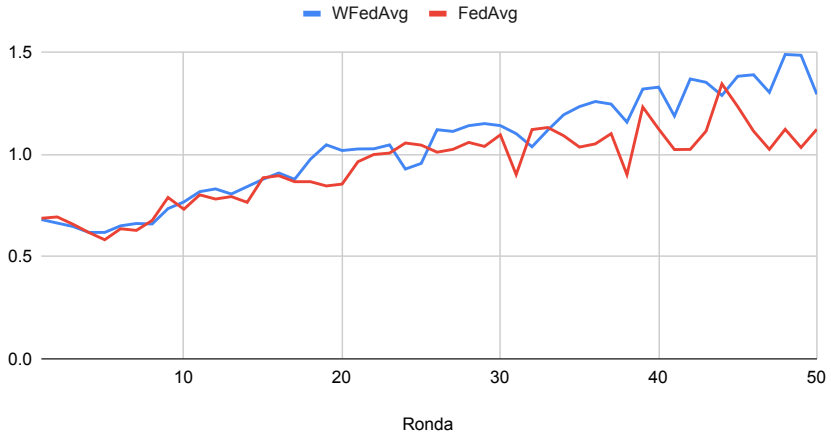


Figura 10.7.: Función de pérdida de FUCITNET.

Parte V.

Conclusiones

11. Conclusiones y vías futuras

Al principio de este trabajo nos proponíamos encontrar un modelo de aprendizaje profundo para predecir la enfermedad COVID-19 a partir de radiografías de tórax. Tal solución debía ser robusta en un escenario clásico de aprendizaje, para después extenderla al caso en que los datos se encuentran repartidos en distintos clientes.

Para comprender el problema, hemos empezado por la base desarrollando los elementos de probabilidad, teoría de la información y optimización convexa en que se fundamenta el problema del aprendizaje. De esta forma lo podemos formular de forma clásica, conocer la teoría de la generalización y uno de sus resultados más importantes: la desigualdad de Vapnik-Chervonenkis. De ella deducimos consecuencias importantes que debemos tener en cuenta en cualquier problema de aprendizaje al que nos enfrentemos.

Siguiendo con nuestro objetivo de estudiar los antecedentes teóricos, hemos entrado de lleno en el ámbito del aprendizaje profundo. Para ello, se han definido formalmente la estructura de una red neuronal, describiendo y estudiando dos de sus variantes más potentes: las redes convolucionales y las redes generativas antagónicas. No nos conformamos con una mera descripción, sino que hemos sido capaces de llevar a cabo un análisis suficientemente exhaustivo de la técnica que presentan. Gracias a ello podemos de exponer con todo detalle la metodología SDNET y el modelo FUCITNET, las cuales, posteriormente, hemos implementado.

Finalizando ya el bloque más teórico, hemos ofrecido una formulación del problema de aprendizaje federado. A través de ella presentamos la técnica de optimización federada más popular, que se basa en extender el método del gradiente descendente a este nuevo escenario. También somos capaces ofrecer una pincelada de los desafíos más importantes del mismo.

En el bloque más práctico, y como consecuencia del estudio de SDNET y FUCITNET, hemos conseguido una implementación software estable y consistente de ambos modelos, la cual hemos llamado SDNETLearning. Además, la hemos extendido al ámbito del aprendizaje federado, usando para ello una nueva biblioteca que trata con este tipo de problemas.

Usando SDNETLearning, hemos conseguido establecer un conjunto de experimentos en dos escenarios: centralizado y federado:

- En el escenario centralizado, hemos obtenido resultados consistentes con los que se exponen en el artículo original de COVID-SDNET ([32]).

- En el escenario federado, los resultados muestran que la extensión de la metodología SDNET a este nuevo escenario es ciertamente robusta y estable. Además, hemos trabajado con particiones que pudieran presentarse en un escenario real, lo que nos ha dado una idea de cómo funcionaría el modelo en tal situación.

En el punto en que nos encontramos, podemos sentirnos satisfechos ya que hemos logrado cumplir todos los objetivos que inicialmente planteábamos. Los resultados son prometedores y se encuentran ciertamente por encima de los que puede obtener el ojo humano en un experto radiólogo, pero aún distan de poderse considerar aceptables para construir un sistema que asesore en la toma de decisiones.

Por ello mismo, el análisis de resultados que hemos llevado a cabo deja abiertas diversas cuestiones que en un futuro se pueden plantear:

- Aunque los resultados son prometedores, estos han sido obtenidos usando un conjunto de datos ciertamente reducido. En esta línea, sería muy interesante y positivo ampliar el conjunto con imágenes de nuevos pacientes. De esta forma, sería posible plantear un experimento en el que todos los hospitales estén bien representados; acercando de esta forma la predicción a una situación cada vez más real.
- Los experimentos planteados comparan el comportamiento de dos operadores de agregación. Hemos concluido que tener en cuenta la proporción de datos de cada nodo no parece un criterio muy válido, pues esta información no tendría por qué ser compartida. Además, no arroja resultados suficientemente buenos para poder plantearlo. En esta línea, sería interesante estudiar nuevos operadores de agregación, que tuviesen en cuenta las características particulares de cada problema y la aportación que cada nodo está haciendo al mismo.
- Hemos observado que el entrenamiento en el escenario federado puede continuarse un número indeterminado de rondas hasta que la convergencia del modelo central acontece a cierto modelo ideal. Sin embargo, aún no tenemos criterios para detectar esta convergencia sin usar un conjunto de validación (lo cual tampoco es muy informativo). Explorar formas de determinar esta convergencia puede ser importante a la hora de entrenar cualquier modelo federado.
- En un plano mucho más teórico, no existe una teoría del aprendizaje en el escenario federado. La aproximación que hemos empleado es natural y ofrece buenos resultados, pero no tenemos garantías teóricas del proceso (las cuales sí se tienen en el escenario clásico). Podría ser interesante explorar esta línea de investigación.

A. Tablas y gráficos

Clase		N			P			Accuracy
		Precision	Recall	F1	Precision	Recall	F1	
Centralizado	FedAvg	0.90150	0.75455	0.82074	0.65432	0.85397	0.73937	0.78618
IID 3 nodos	FedAvg	0.852248	0.778533	0.811783	0.659715	0.769820	0.706221	0.771276
IID 6 nodos	FedAvg	0.872295	0.765626	0.813539	0.62414	0.795667	0.692481	0.768996
IID 9 nodos	FedAvg	0.837837	0.788135	0.812227	0.675324	0.742857	0.707482	0.769276
noIID Hospitales	FedAvg	0.732309	**0.768021**	0.742831	**0.720187**	0.701887	**0.703526**	0.728395
	WFedAvg	**0.833385**	0.728739	**0.774206**	0.610401	**0.759166**	0.669671	**0.733686**
noIID Demográfica	FedAvg	**0.741652**	0.716216	**0.726209**	0.689436	**0.722124**	0.702549	0.715789
	WFedAvg	0.683882	**0.753641**	0.712705	**0.758365**	0.702531	**0.725924**	**0.721637**
noIID Puntos RALE	FedAvg	**0.872272**	0.691125	0.769539	0.609398	**0.832464**	0.699754	0.740039
	WFedAvg	0.842038	**0.718079**	**0.774267**	**0.671105**	0.814067	**0.733874**	**0.756216**
noIID Niveles RALE	FedAvg	**0.877420**	**0.658237**	**0.749936**	**0.562357**	**0.840743**	**0.667302**	**0.716696**
	WFedAvg	0.860650	0.648502	0.739368	0.557478	0.808599	0.659505	0.705061

Tabla A.1.: Resultados de los experimentos de FUCITNET. Para cada partición del escenario noIID se señala en negrita el mejor resultado de cada métrica.

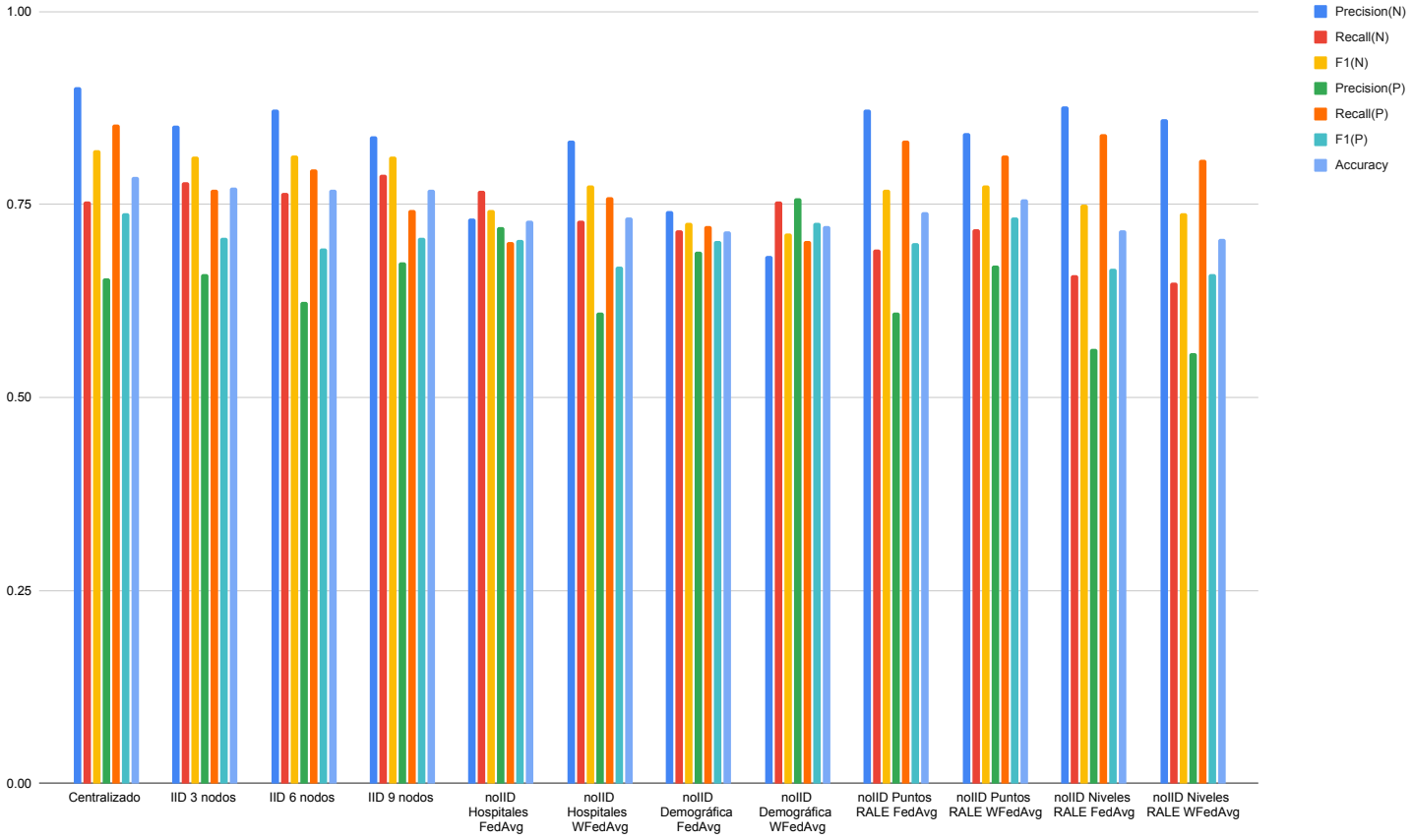


Figura A.1.: Gráfico de columnas de los resultados de los experimentos de FUCITNET.

Clase		N			P			$Accuracy_2$	$Accuracy_4$
Métrica		Precision	Recall	F1	Precision	Recall	F1		
Centralizado	FedAvg	0.804954	0.745373	0.76876	0.72732	0.778449	0.746675	0.76182	0.751195
IID 3 nodos	FedAvg	0.787338	0.795011	0.79057	0.711971	0.700418	0.705042	0.755319	0.75325
IID 6 nodos	FedAvg	0.797066	0.765576	0.777381	0.699653	0.723506	0.705437	0.748817	0.742021
IID 9 nodos	FedAvg	0.794392	0.765765	0.779816	0.679012	0.714285	0.696202	0.746680	0.747340
noIID Hospitales	FedAvg	0.770858	0.718542	0.741339	0.677425	0.724839	0.697872	0.723692	0.723398
	WFedAvg	**0.771781**	**0.726627**	**0.745176**	**0.690814**	**0.732717**	**0.708159**	**0.729571**	0.723104
noIID Demográfica	FedAvg	**0.731016**	0.765739	**0.747446**	**0.745072**	**0.706212**	**0.724430**	**0.736842**	**0.733626**
	WFedAvg	0.718570	**0.770892**	0.742613	0.743459	0.683281	0.710399	0.728070	0.727193
noIID Puntos RALE	FedAvg	**0.753802**	**0.783073**	**0.766930**	**0.776600**	**0.744464**	**0.758933**	**0.763181**	**0.754225**
	WFedAvg	0.725266	0.770229	0.745810	0.757954	0.708255	0.730803	0.738951	0.735762
noIID Niveles RALE	FedAvg	**0.687271**	0.760120	0.720905	0.746844	**0.669783**	**0.704952**	0.713787	0.713496
	WFedAvg	0.683085	**0.802478**	**0.737810**	**0.774718**	0.645721	0.704029	**0.722513**	**0.718441**

Tabla A.2.: Resultados de los experimentos de SDNET. Para cada partición del escenario noIID se señala en negrita el mejor resultado de cada métrica.

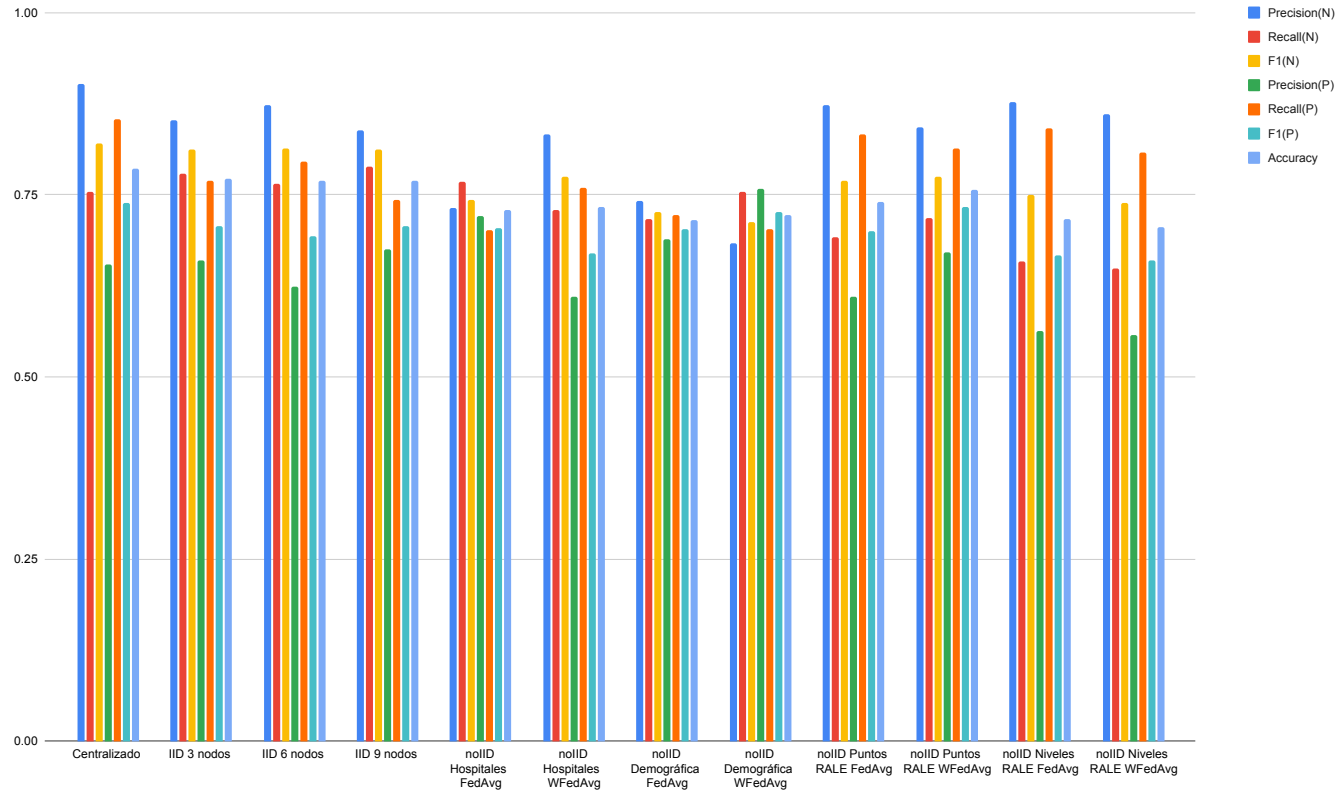


Figura A.2.: Gráfico de columnas de los resultados de los experimentos de SDNET.

Clase			N			P			$Accuracy_2$	$Accuracy_4$
Métrica			Precision	Recall	F1	Precision	Recall	F1		
Centralizado	FedAvg	FUCITNET	**0.90150**	**0.75455**	**0.82074**	0.65432	**0.85397**	0.73937	**0.78618**	
		SDNET	0.804954	0.745373	0.76876	**0.72732**	0.778449	**0.746675**	0.76182	0.751195
IID 3 nodos	FedAvg	FUCITNET	**0.852248**	0.778533	**0.811783**	0.659715	**0.769820**	**0.706221**	**0.771276**	
		SDNET	0.787338	**0.795011**	0.79057	**0.711971**	0.700418	0.705042	0.755319	0.75325
IID 6 nodos	FedAvg	FUCITNET	**0.872295**	**0.765626**	**0.813539**	0.62414	**0.795667**	0.692481	**0.768996**	
		SDNET	0.797066	0.765576	0.777381	**0.699653**	0.723506	**0.705437**	0.748817	0.742021
IID 9 nodos	FedAvg	FUCITNET	**0.837837**	**0.788135**	**0.812227**	0.675324	**0.742857**	**0.707482**	**0.769276**	
		SDNET	0.794392	0.765765	0.779816	**0.679012**	0.714285	0.696202	0.746680	0.747340
noIID Hospitales	FedAvg	FUCITNET	0.732309	**0.768021**	0.742831	**0.720187**	0.701887	0.703526	0.728395	
		SDNET	0.770858	0.718542	0.741339	0.677425	0.724839	0.697872	0.723692	0.723398
	WFedAvg	FUCITNET	**0.833385**	0.728739	**0.774206**	0.610401	**0.759166**	0.669671	**0.733686**	
		SDNET	0.771781	0.726627	0.745176	0.690814	0.732717	**0.708159**	0.729571	0.723104
noIID Demográfica	FedAvg	FUCITNET	**0.741652**	0.716216	0.726209	0.689436	**0.722124**	0.702549	0.715789	
		SDNET	0.731016	0.765739	**0.747446**	0.745072	0.706212	0.724430	**0.736842**	0.733626
	WFedAvg	FUCITNET	0.683882	0.753641	0.712705	**0.758365**	0.702531	**0.725924**	0.721637	
		SDNET	0.718570	**0.770892**	0.742613	0.743459	0.683281	0.710399	0.728070	0.727193
noIID Puntos RALE	FedAvg	FUCITNET	**0.872272**	0.691125	0.769539	0.609398	**0.832464**	0.699754	0.740039	
		SDNET	0.753802	**0.783073**	0.766930	**0.776600**	0.744464	**0.758933**	**0.763181**	0.754225
	WFedAvg	FUCITNET	0.842038	0.718079	**0.774267**	0.671105	0.814067	0.733874	0.756216	
		SDNET	0.725266	0.770229	0.745810	0.757954	0.708255	0.730803	0.738951	0.735762
noIID Niveles RALE	FedAvg	FUCITNET	**0.877420**	0.658237	**0.749936**	0.562357	**0.840743**	0.667302	0.716696	
		SDNET	0.687271	0.760120	0.720905	0.746844	0.669783	**0.704952**	0.713787	0.713496
	WFedAvg	FUCITNET	0.860650	0.648502	0.739368	0.557478	0.808599	0.659505	0.705061	
		SDNET	0.683085	**0.802478**	0.737810	**0.774718**	0.645721	0.704029	**0.722513**	0.718441

Tabla A.3.: Resultados de todos los experimentos. Para cada partición se señala en negrita el mejor resultado de cada métrica.

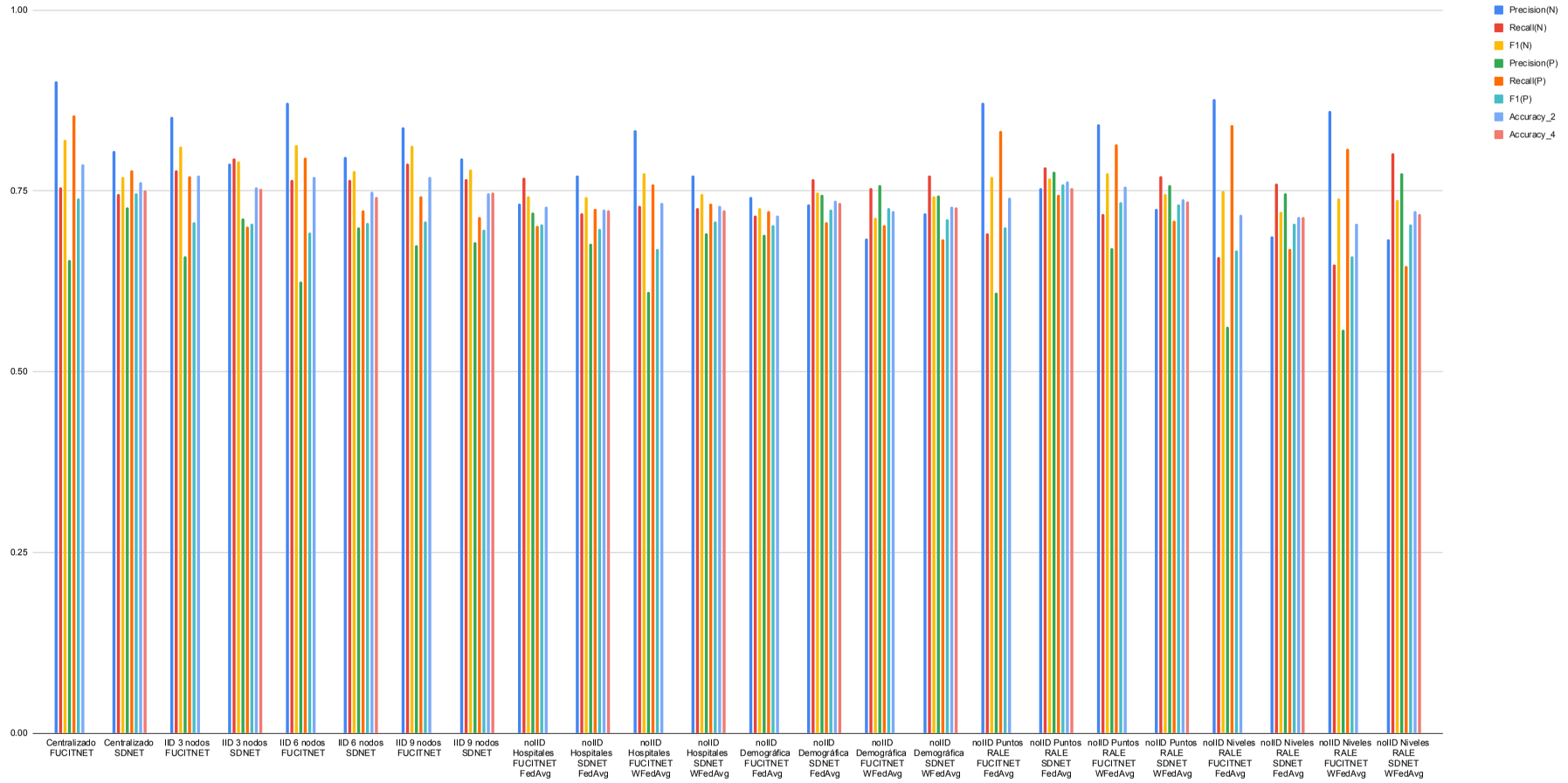


Figura A.3.: Gráfico de columnas de los resultados de todos los experimentos.

		Clasificador FUCITNET		Clasificador SDNET	
Clase	Medida	Media	Desv. Típica	Media	Desv. Típica
N	Precision	**0.90150**	0.04046	0.80495	0.04927
	Recall	**0.75455**	0.04451	0.74537	0.08870
	F1	**0.82074**	0.03537	0.76877	0.03975
P	Precision	0.65432	0.04776	**0.72733**	0.05592
	Recall	**0.85397**	0.04112	0.77845	0.08259
	F1	0.73937	0.02916	**0.74668**	0.02874
	$Accuracy_2$	**0.78618**	0.03268	0.76183	0.02341
	$Accuracy_4$			0.7512	0.01771

Tabla A.4.: Escenario centralizado. Para cada métrica se señala en negrita el mejor resultado.

		Clasificador FUCITNET		Clasificador SDNET	
Clase	Medida	Media	Desv. Típica	Media	Desv. Típica
N	Precision	**0.85225**	0.06549	0.78734	0.01820
	Recall	0.77853	0.02854	**0.79501**	0.03536
	F1	**0.81178**	0.02526	0.79057	0.01623
P	Precision	0.65972	0.06723	**0.71197**	0.03857
	Recall	**0.76982**	0.07426	0.70042	0.03902
	F1	**0.70622**	0.03659	0.70504	0.02561
	$Accuracy_2$	**0.77128**	0.02646	0.75532	0.01823
	$Accuracy_4$			0.75325	0.01691

Tabla A.5.: Escenario federado, configuración IID, 3 nodos. Para cada métrica se señala en negrita el mejor resultado.

		Clasificador FUCITNET		Clasificador SDNET	
Clase	Medida	Media	Desv. Típica	Media	Desv. Típica
N	Precision	**0.869842**	0.071219	0.797067	0.038379
	Recall	**0.778555**	0.030646	0.765577	0.086272
	F1	**0.819858**	0.031411	0.777381	0.040371
P	Precision	0.651621	0.079351	**0.699654**	0.059734
	Recall	**0.796529**	0.094969	0.723507	0.103234
	F1	**0.710252**	0.045892	0.705437	0.056606
	$Accuracy_2$	**0.77896**	0.033967	0.748818	0.035660
	$Accuracy_4$			0.742021	0.037115

Tabla A.6.: Escenario federado, configuración IID, 6 nodos. Para cada métrica se señala en negrita el mejor resultado.

		Clasificador FUCITNET		Clasificador SDNET	
Clase	Medida	Media	Desv. Típica	Media	Desv. Típica
N	Precisión	**0.837837**	0.04879	0.794392	0.047529
	Recall	**0.788135**	0.05625	0.765765	0.087451
	F1	**0.812227**	0.038741	0.779816	0.052036
P	Precisión	0.675324	0.032755	**0.679012**	0.063398
	Recall	**0.742857**	0.084512	0.714285	0.09542
	F1	**0.707482**	0.045421	0.696202	0.021158
	$Accuracy_2$	**0.769276**	0.033967	0.746680	0.032175
	$Accuracy_4$			0.747340	0.038512

Tabla A.7.: Escenario federado, configuración IID, 9 nodos. Para cada métrica se señala en negrita el mejor resultado.

		Clasificador FUCITNET		Clasificador SDNET	
Clase	Medida	Media	Desv. Típica	Media	Desv. Típica
N	Precision	0.73231	0.12808	**0.77086**	0.04091
	Recall	**0.76802**	0.02300	0.71854	0.05160
	F1	**0.74283**	0.06481	0.74134	0.01446
P	Precision	**0.72019**	0.08403	0.67743	0.01873
	Recall	0.70189	0.07544	**0.72484**	0.09874
	F1	**0.70353**	0.02150	0.69787	0.05245
	$Accuracy_2$	**0.72840**	0.03656	0.72369	0.02011
	$Accuracy_4$			0.72340	0.01949

Tabla A.8.: Escenario federado, configuración noIID-Hospitales, FedAvg. Para cada métrica se señala en negrita el mejor resultado.

		Clasificador FUCITNET		Clasificador SDNET	
Clase	Medida	Media	Desv. Típica	Media	Desv. Típica
N	Precisión	**0.83339**	0.08731	0.77178	0.01467
	Recall	**0.72874**	0.03462	0.72663	0.09191
	F1	**0.77421**	0.03361	0.74518	0.05086
P	Precisión	0.61040	0.09290	**0.69081**	0.06080
	Recall	**0.75917**	0.08297	0.73272	0.05356
	F1	0.66967	0.05195	**0.70816**	0.03228
	$Accuracy_2$	**0.73369**	0.03272	0.72957	0.03564
	$Accuracy_4$			0.72310	0.03497

Tabla A.9.: Escenario federado, configuración noIID-Hospitales, WFedAvg. Para cada métrica se señala en negrita el mejor resultado.

		Clasificador FUCITNET		Clasificador SDNET	
Clase	Medida	Media	Desv. Típica	Media	Desv. Típica
N	Precisión	**0.74165**	0.06877	0.73102	0.03091
	Recall	0.71622	0.03927	**0.76574**	0.04794
	F1	0.72621	0.03132	**0.74745**	0.03347
P	Precisión	0.68944	0.06863	**0.74507**	0.03390
	Recall	**0.72212**	0.04968	0.70621	0.03405
	F1	0.70255	0.03617	**0.72443**	0.02379
	$Accuracy_2$	0.71579	0.02883	**0.73684**	0.02760
	$Accuracy_4$			0.73363	0.02400

Tabla A.10.: Escenario federado, configuración noIID-Demografica, FedAvg. Para cada métrica se señala en negrita el mejor resultado.

		Clasificador FUCITNET		Clasificador SDNET	
Clase	Medida	Media	Desv. Típica	Media	Desv. Típica
N	Precisión	0.68388	0.08813	0.71857	0.02081
	Recall	0.75364	0.03964	**0.77089**	0.05000
	F1	0.71271	0.03917	**0.74261**	0.01927
P	Precisión	**0.75836**	0.07899	0.74346	0.03486
	Recall	**0.70253**	0.04456	0.68328	0.04629
	F1	**0.72592**	0.03132	0.71040	0.01943
	$Accuracy_2$	0.72164	0.02447	**0.72807**	0.01393
	$Accuracy_4$			0.72719	0.01109

Tabla A.11.: Escenario federado, configuración noIID-Demografica, WFedAvg. Para cada métrica se señala en negrita el mejor resultado.

		Clasificador FUCITNET		Clasificador SDNET	
Clase	Medida	Media	Desv. Típica	Media	Desv. Típica
N	Precisión	**0.87227**	0.04256	0.75380	0.04343
	Recall	0.69113	0.03786	**0.78307**	0.02988
	F1	**0.76954**	0.01101	0.76693	0.01821
P	Precisión	0.60940	0.07670	**0.77660**	0.02049
	Recall	**0.83246**	0.03171	0.74446	0.05537
	F1	0.69975	0.03891	**0.75893**	0.02617
	$Accuracy_2$	0.74004	0.01905	**0.76318**	0.02141
	$Accuracy_4$			0.75422	0.02269

Tabla A.12.: Escenario federado, configuración noIID-PuntosRALE, FedAvg. Para cada métrica se señala en negrita el mejor resultado.

		Clasificador FUCITNET		Clasificador SDNET	
Clase	Medida	Media	Desv. Típica	Media	Desv. Típica
N	Precisión	**0.84204**	0.04303	0.72527	0.04688
	Recall	0.71808	0.01931	**0.77023**	0.05223
	F1	**0.77427**	0.01361	0.74581	0.03741
P	Precisión	0.67111	0.04554	**0.75795**	0.04542
	Recall	**0.81407**	0.03598	0.70825	0.06503
	F1	**0.73387**	0.01997	0.73080	0.04504
	$Accuracy_2$	**0.75622**	0.01160	0.73895	0.03976
	$Accuracy_4$			0.73576	0.03331

Tabla A.13.: Escenario federado, configuración noIID-PuntosRALE, WFedAvg. Para cada métrica se señala en negrita el mejor resultado.

		Clasificador FUCITNET		Clasificador SDNET	
Clase	Medida	Media	Desv. Típica	Media	Desv. Típica
N	Precisión	**0.87742**	0.07352	0.68727	0.02546
	Recall	0.65824	0.02497	**0.76012**	0.03595
	F1	**0.74994**	0.01952	0.72091	0.01182
P	Precisión	0.56236	0.08367	**0.74684**	0.03088
	Recall	**0.84074**	0.06012	0.66978	0.04797
	F1	0.66730	0.04406	**0.70495**	0.02763
	$Accuracy_2$	**0.71670**	0.01154	0.71379	0.01481
	$Accuracy_4$			0.71350	0.01142

Tabla A.14.: Escenario federado, configuración noIID-NivelesRALE, FedAvg. Para cada métrica se señala en negrita el mejor resultado.

		Clasificador FUCITNET		Clasificador SDNET	
Clase	Medida	Media	Desv. Típica	Media	Desv. Típica
N	Precisión	**0.86065**	0.03595	0.68308	0.02519
	Recall	0.64850	0.01867	**0.80248**	0.01571
	F1	**0.73937**	0.02070	0.73781	0.01872
P	Precisión	0.55748	0.02394	**0.77472**	0.01937
	Recall	**0.80860**	0.04899	0.64572	0.04039
	F1	0.65951	0.02805	**0.70403**	0.03119
	$Accuracy_2$	0.70506	0.02127	**0.72251**	0.01994
	$Accuracy_4$			0.71844	0.01598

Tabla A.15.: Escenario federado, configuración noIID-NivelesRALE, WFedAvg. Para cada métrica se señala en negrita el mejor resultado.

Bibliografía

[1] What are generative adversarial networks? Accesible en: https://2018.igem.org/Team:Vilnius-Lithuania-OG/Gan_Introduction. [Citado en pág. 60]

[2] Documentación de Pytorch. Accesible en: https://pytorch.org/docs/stable/index.html. [Citado en pág. 85]

[3] Documentación de las APIs de Tensorflow. Accesible en: https://www.tensorflow.org/api_docs. [Citado en pág. 86]

[4] Y. S. Abu-Mostafa, M. Magdon-Ismail, and H.-T. Lin. *Learning from data*, volume 4. AMLBook New York, NY, USA, 2012. [Citado en págs. 35 and 41]

[5] S. Boyd and L. Vandenberghe. *Convex Optimization*. Cambridge University Press, 2004. [Citado en pág. 5]

[6] K. M. M. R. M. A. C. D. K. B. T. B. S. S. J. S. M. L. M. D. G. H. B. L. v. d. V. B. v. d. B. S. W. L. G. G. R. J.-L. E. J. Z. M. P. M. G. H. R. C. K. M. P. D. C. Corman Victor M, Landt Olfert. Detection of 2019 novel coronavirus (2019-nCoV) by real-time RT-PCR. *Euro Surveill*, 25, 2020. [Citado en págs. 3 and 81]

[7] R. Durrett. *Probability: Theory and Examples*. Cambridge Series in Statistical and Probabilistic Mathematics. Cambridge University Press, 4 edition, 2010. [Citado en págs. 11 and 26]

[8] C. Dwork. Differential privacy. pages 1–12, 2006. [Citado en pág. 77]

[9] M. Fredrikson, S. Jha, and T. Ristenpart. Model inversion attacks that exploit confidence information and basic countermeasures. page 1322–1333, 2015. [Citado en pág. 77]

[10] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative adversarial nets. pages 2672–2680, 2014. [Citado en págs. xvii, 59, 61, and 63]

[11] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. 2015. [Citado en págs. 65 and 68]

[12] Y. B. Ian Goodfellow and A. Courville. *Deep Learning*. MIT Press, 2016. [Citado en págs. 6 and 35]

[13] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. 2017. [Citado en págs. 30 and 87]

[14] A. Klenke. *Probability Theory: A Comprehensive Course*. Universitext. Springer London, 2013. [Citado en págs. 5, 11, 14, and 61]

[15] J. Konecny, H. B. McMahan, F. X. Yu, P. Richtarik, A. T. Suresh, and D. Bacon. Federated learning: Strategies for improving communication efficiency. 2016. [Citado en pág. 4]

[16] S. Kullback and R. A. Leibler. On information and sufficiency. *The annals of mathematical statistics*, 22(1):79–86, 1951. [Citado en págs. xvi and 20]

[17] H. Kumar. Quick intro to instance segmentation: Mask R-CNN. Accesible en: https://kharshit.github.io/blog/2019/08/23/quick-intro-to-instance-segmentation, 2019. [Citado en pág. 59]

[18] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. Backpropagation applied to handwritten zip code recognition. *Neural Computation*, 1(4):541–551, 1989. [Citado en pág. 3]

[19] H. B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas. Communication-efficient learning of deep networks from decentralized data. 2017. [Citado en págs. 71, 73, and 76]

[20] E. Mineo. U-Net lung segmentation. Accesible en: `https://www.kaggle.com/eduardomineo/u-net-lung-segmentation-montgomery-shenzhen`, 2020. [Citado en pág. 67]

[21] T. M. Mitchell. *Machine Learning*. McGraw-Hill, New York, 1997. [Citado en pág. 35]

[22] V. Nair and G. Hinton. Rectified linear units improve restricted boltzmann machines vinod nair. 27:807–814, 06 2010. [Citado en pág. 57]

[23] A. B. Novikoff. On convergence proofs for perceptrons. 1963. [Citado en pág. 49]

[24] K.-S. Oh and K. Jung. GPU implementation of neural networks. *Pattern Recognition*, 37(6):1311–1314, 2004. [Citado en pág. 3]

[25] M. Rey-Area, E. Guirado, S. Tabik, and J. Ruiz-Hidalgo. Fucitnet: Improving the generalization of deep learning networks by the fusion of learned class-inherent transformations. *Information Fusion*, 63:188–195, 2020. [Citado en págs. xvii, 4, 6, 63, 65, 66, and 88]

[26] N. Rodríguez-Barroso, G. Stipcich, D. Jiménez-López, J. A. Ruiz-Millán, E. Martínez-Cámara, G. González-Seco, M. V. Luzón, M. A. Veganzones, and F. Herrera. Federated learning and differential privacy: Software tools analysis, the sherpa.ai fl framework and methodological guidelines for preserving data privacy. *Information Fusion*, 64:270–292, Dec 2020. [Citado en págs. xviii, 6, 73, and 74]

[27] V. Rohatgi and A. Saleh. *An Introduction to Probability and Statistics, Second Edition*. 2011. [Citado en pág. 11]

[28] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 2015. [Citado en págs. 65 and 68]

[29] E. Seyed Hosseini, N. Riahi Kashani, H. Nikzad, J. Azadbakht, H. Hassani Bafrani, and H. Haddad Kashani. The novel coronavirus disease-2019 (COVID-19): Mechanism of action, detection and recent therapeutic strategies. *Virology*, 551:1–9, 2020. [Citado en págs. 3 and 81]

[30] C. E. Shannon. A mathematical theory of communication. *The Bell System Technical Journal*, 27(4): 623–656, 1948. [Citado en pág. 19]

[31] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. 2014. [Citado en pág. 64]

[32] S. Tabik, A. Gómez-Ríos, J. L. Martín-Rodríguez, I. Sevillano-García, M. Rey-Area, D. Charte, E. Guirado, J. L. Suárez, J. Luengo, M. A. Valero-González, P. García-Villanova, E. Olmedo-Sánchez, and F. Herrera. COVIDGR Dataset and COVID-SDNet Methodology for Predicting COVID-19 Based on Chest X-Ray Images. *IEEE Journal of Biomedical and Health Informatics*, 24(12):3595–3605, 2020. [Citado en págs. xvii, xviii, 4, 6, 7, 66, 67, 69, 89, and 101]

[33] V. N. Vapnik. *The Nature of Statistical Learning Theory*. Springer, second edition, 1999. [Citado en págs. xvii, 5, 35, and 38]

[34] M. Weinstock and et al. Chest X-Ray findings in 636 ambulatory patients with COVID-19 presenting to an urgent care center: A normal chest X-Ray is no guarantee. *J Urgent Care Med*, 14(7): 13–18, 2020. [Citado en pág. 81]

[35] H. Wong and et al. Frequency and distribution of chest radiographic findings in COVID-19 positive patients. *Radiology*, page 201160, 2020. [Citado en pág. 81]

[36] C. Xie, S. Koyejo, and I. Gupta. Asynchronous federated optimization. 2020. [Citado en pág. 76]

[37] X. Xie, Z. Zhong, W. Zhao, C. Zheng, F. Wang, and J. Liu. Chest CT for typical coronavirus disease 2019 (COVID-19´) pneumonia: Relationship to negative rt-pcr testing. *Radiology*, 296(2):E41–E45, 2020. [Citado en pág. 81]

[38] Q. Yang, L. Fan, and H. Yu. Federated learning: Privacy and incentive. *Federated Learning*, 2020. [Citado en pág. 77]

[39] M. Yani, S. Irawan, and M. S.T. Application of transfer learning using convolutional neural network method for early detection of terry's nail. *Journal of Physics: Conference Series*, 1201, 05 2019. [Citado en pág. 59]

[40] L. Zhu, Z. Liu, and S. Han. Deep leakage from gradients. 2019. [Citado en pág. 77]