



UNIVERSIDAD DE GRANADA

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

DOBLE GRADO EN INGENIERÍA INFORMÁTICA Y MATEMÁTICAS

TRABAJO DE FIN DE GRADO

Sistemas de optimización para el balance entre resiliencia a ataques a la privacidad y rendimiento en Aprendizaje Federado

Presentado por:
Julio Pérez Cabeza

Tutores:
Francisco Javier Merí de la Maza
Nuria Rodríguez Barroso

Curso académico 2024-2025

Sistemas de optimización para el balance entre resiliencia a ataques a la privacidad y rendimiento en Aprendizaje Federado

Julio Pérez Cabeza

Julio Pérez Cabeza *Sistemas de optimización para el balance entre resiliencia a ataques a la privacidad y rendimiento en Aprendizaje Federado.*

Trabajo de fin de Grado. Curso académico 2024-2025.

**Responsable de
tutorización**

Francisco Javier Merí de la Maza
Departamento de Análisis Matemático

Facultad de Ciencias

Nuria Rodríguez Barroso
*Departamento de Ciencias de la
Computación e Inteligencia Artificial*

Escuela Técnica Superior
de Ingenierías Informática
y de Telecomunicación

Doble Grado en Ingeniería
Informática y Matemáticas

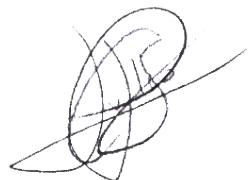
Universidad de Granada

DECLARACIÓN DE ORIGINALIDAD

D./Dña. Julio Pérez Cabeza

Declaro explícitamente que el trabajo presentado como Trabajo de Fin de Grado (TFG), correspondiente al curso académico 2024-2025, es original, entendido esto en el sentido de que no he utilizado para la elaboración del trabajo fuentes sin citarlas debidamente.

En Granada a 11 de julio de 2025



Fdo: Julio Pérez Cabeza

*A mis padres, hermanos y
a Ire.*

Agradecimientos

Me gustaría comenzar agradeciendo a mis tutores, Javier y Nuria, pues han sido un apoyo y ayuda constante durante el transcurso de estos meses de gran trabajo.

También a mis padres, no solo por este último tramo, sino por todos los años de dedicación, esfuerzo y cariño que me han dado para no perder las ganas. Gracias a mi hermano y a mi hermana, a los que siempre llevo cerca y que, de una forma u otra, siempre me han ayudado.

Gracias a mis amigos. Primero, a los compañeros desde el principio, Juan Antonio y Guillermo, compañeros de alegrías (y penurias) durante estos años. Y segundo, a los que, lamentablemente, han llegado algo más tarde, pero estoy increíblemente agradecido de haber conocido. Mario, Jaime, Nuria (de nuevo), Álvaro y Adri. Muchas gracias a todos.

Por último, a Ire. Gracias por la paciencia infinita, por escucharme incondicionalmente y por validar continuamente todo el trabajo que he hecho.

Sistemas de optimización para el balance entre resiliencia a ataques a la privacidad y rendimiento en Aprendizaje Federado

Palabras clave

ML - Machine learning, FL - Federated Learning, PD - Privacidad Diferencial, PL - Programación Lineal, IA - Inteligencia Artificial

Resumen

El creciente uso de sistemas de ML en aplicaciones críticas ha incrementado la necesidad de modelos seguros y que preserven la privacidad. El FL ha surgido como un paradigma prometedor de aprendizaje automático descentralizado. Este permite que múltiples clientes colaboren en el entrenamiento de un modelo compartido sin necesidad de transferir sus datos en crudo a un servidor central. En su lugar, solo se comparten actualizaciones del modelo, lo que hace que FL sea atractivo para entornos sensibles a la privacidad.

Sin embargo, esta configuración descentralizada no elimina completamente el riesgo de ataques a la privacidad. Incluso cuando los datos en crudo no se transmiten directamente, las actualizaciones del modelo pueden filtrar información sensible sobre puntos individuales del conjunto de datos. Existen además ataques que implementan técnicas que consiguen corromper el modelo compartido, empeorando su rendimiento.

Para abordar este desafío, la PD se ha convertido en un mecanismo clave. Esta proporciona una definición matemáticamente rigurosa de privacidad, asegurando que la presencia o ausencia de cualquier individuo en el conjunto de datos tenga un impacto mínimo en la salida de un algoritmo. Esto se logra típicamente añadiendo ruido calibrado a las actualizaciones del modelo o a las salidas de ciertas consultas.

No obstante, una limitación importante de PD en la práctica es la degradación del rendimiento del modelo. La adición de ruido a menudo conduce a modelos menos precisos, especialmente cuando el presupuesto de privacidad es pequeño. Esta disyuntiva entre privacidad y rendimiento es el núcleo de este proyecto. El objetivo central es explorar si las técnicas de optimización (especialmente la PL) pueden aprovecharse para ajustar dinámicamente los parámetros de PD durante el proceso de entrenamiento, de manera que se equilibre la protección de la privacidad con la utilidad del modelo.

El trabajo se divide en dos partes principales: una sección matemática y una sección de informática aplicada. El componente matemático proporciona una base rigurosa en análisis lineal y convexo y teoría de optimización. Se estudian en detalle resultados clave como el teorema de Minkowski-Carathéodory, el desarrollo del algoritmo del Simplex para resolver problemas lineales, y propiedades de conjuntos y funciones convexas. Estos forman la base teórica para las aplicaciones computacionales posteriores.

Para probar el enfoque, se llevan a cabo una serie de experimentos utilizando conjuntos de datos públicos. Se simulan diferentes tipos de ataques a la privacidad sobre modelos de FL. En cada caso, se analiza cómo la aplicación de PD afecta la resiliencia del modelo y cómo el sistema de optimización adapta el nivel de ruido para mitigar el impacto. Los resultados muestran que optimizar dinámicamente los parámetros de PD mediante programación lineal mejora significativamente la relación entre privacidad y rendimiento, en comparación con elecciones estáticas de parámetros.

Además, el trabajo enfatiza la sinergia entre la teoría matemática y el aprendizaje automático práctico. Herramientas clásicas de la optimización lineal, como el método del Simplex, resultan no solo teóricamente sólidas sino también efectivas en la práctica para abordar preocupaciones reales en sistemas de IA.

En conclusión, este proyecto aporta un marco novedoso que integra privacidad diferencial, aprendizaje federado y optimización en un sistema coherente y eficiente. Demuestra que las técnicas matemáticas clásicas siguen siendo poderosas y relevantes en la ciencia de datos moderna. El marco es adaptable, matemáticamente fundamentado y práctico, lo que lo convierte en una herramienta valiosa para diseñar sistemas de IA confiables, que sean tanto precisos como respetuosos con la privacidad del usuario.

Optimization Systems for Balancing Privacy Attack Resilience and Performance in Federated Learning

Keywords

ML - Machine learning, FL - Federated Learning, DP - Differential Privacy, LP - Linear Programming, AI - Artificial Intelligence

Abstract

The growing prevalence of ML systems in critical applications has increased the need for secure, privacy-preserving models. FL has emerged as a promising decentralized machine learning paradigm. It allows multiple clients to collaboratively train a shared model without transferring their raw data to a central server. Instead, only model updates are shared, which makes FL appealing for privacy-sensitive environments.

However, this decentralized setup does not fully eliminate the risk of privacy attacks. Even when raw data is not directly transmitted, model updates can leak sensitive information about individual data points.

To address this challenge, DP has become a key mechanism. It provides a mathematically rigorous definition of privacy, ensuring that the presence or absence of any individual in the dataset has a minimal impact on the output of an algorithm. This is typically achieved by adding calibrated noise to the model updates or to the outputs of queries.

Nevertheless, a major limitation of DP in practice is the degradation in model performance. The addition of noise often leads to less accurate models, especially when the privacy budget is small. This trade-off between privacy and performance lies at the core of this project. The central objective is to explore whether optimization techniques (specially LP) can be leveraged to dynamically adjust DP parameters during the training process, in a way that balances privacy protection and model utility.

The work is divided into two main parts: a mathematical section and an applied computer science section. The mathematical component provides a rigorous foundation in convex analysis and optimization theory. Key results such as the Minkowski-Carathéodory theorem, the Simplex algorithm development for solving linear programs, and properties of convex sets and functions are studied in detail. These form the theoretical backbone for the later computational applications.

To test the approach, we conduct a series of experiments using publicly available datasets. We simulate different types of privacy attacks on FL models. In each case, we analyze how the application of DP affects the resilience of the model and how the optimization system adapts the noise level to mitigate the impact. The results show that dynamically optimizing the DP parameters using linear programming significantly improves the privacy-performance trade-off compared to static parameter choices.

Moreover, the work emphasizes the synergy between mathematical theory and practical machine learning. Classical tools from linear optimization, such as the Simplex method, prove to be not only theoretically sound but also practically effective in addressing real-world concerns in AI systems.

In conclusion, this project contributes a novel framework that integrates differential privacy, federated learning, and optimization in a coherent and efficient system. It demonstrates that classical mathematical techniques remain powerful and relevant in modern data science. The framework is adaptable, mathematically grounded, and practical, making it a valuable tool for designing trustworthy AI systems that are both accurate and respectful of user privacy.

Índice general

Agradecimientos	v
Índice de figuras	xiii
Índice de tablas	xv
Introducción	xvii
1. Contexto	xvii
2. Motivación	xviii
3. Estructura del trabajo	xxi
4. Objetivos	xxi
5. Metodología y planificación	xxii
I. Optimización lineal y convexa	1
1. Preliminares y fundamentos teóricos	3
1.1. Introducción	3
1.2. Preliminares	3
1.2.1. Método de Gauss	4
1.2.2. Método de Eliminación de Gauss-Jordan	5
1.2.3. Cálculo de la inversa de una matriz invertible	6
1.3. Conjuntos afines	7
1.4. Conjuntos convexos en espacios normados	9
1.5. Convexidad en \mathbb{R}^n	10
1.6. El Teorema de Minkowski-Carathéodory	17
2. Problemas de Programación Lineal	21
2.1. Extremos de un conjunto factible	24
3. El método Simplex	29
3.1. Motivación del algoritmo	29
3.2. Notación y definiciones previas	32
3.3. Fase II del método Simplex	33
3.3.1. Terminación y ciclo	39
3.4. Fase I del método Simplex	41
3.4.1. Problema Auxiliar	41
4. Programación convexa	45
4.1. El problema de la optimización convexa	45
4.1.1. Método de direcciones factibles	49
4.1.2. Método del gradiente proyectado	50
4.2. El teorema de Krein-Milman	52

Índice general

4.3. Principio del máximo de Bauer	55
II. Aplicación práctica de la optimización en la privacidad en Aprendizaje Federado	59
5. El concepto de Aprendizaje	61
5.1. Aprendizaje Automático	61
5.2. Aprendizaje Profundo	62
5.2.1. Entrenamiento de la red	66
5.2.2. Redes y capas convolucionales	67
6. Aprendizaje Federado	71
6.1. Formalización del Aprendizaje Federado	73
6.2. Flujo de trabajo en Aprendizaje Federado	74
6.3. Arquitecturas de aprendizaje	76
6.4. Categorías de Aprendizaje Federado	77
6.5. Privacidad Diferencial	77
6.5.1. Definición matemática	80
6.5.2. Mecanismos de privacidad diferencial	81
6.5.3. Propiedades clave	82
6.6. Ataques en el Aprendizaje Federado	83
6.6.1. Ataques al modelo	85
6.6.2. Ataques a la privacidad	89
7. Optimización en la aplicación de Privacidad Diferencial	93
8. Entorno experimental	97
8.1. Conjuntos de datos	97
8.2. Escenarios de ataque	99
8.3. Métricas empleadas	103
8.4. Detalles de implementación	104
9. Análisis de los resultados experimentales	107
9.1. Ataque <i>Label Flipping</i>	107
9.2. Ataque gaussiano	110
9.3. Ataque <i>Backdoor</i>	112
9.4. Ataque de reconstrucción con GAN	115
III. Conclusiones y Trabajos futuros	117
10. Conclusiones	119
11. Trabajos futuros	121
Bibliografía	125

Índice de figuras

1.	Pirámide de riesgos según la UE	xviii
2.	Interés en PD a lo largo de los años	xix
3.	Interés en FL a lo largo de los años	xx
1.1.	Demostración del Teorema de <i>Minkowski-Carathéodory</i>	19
3.1.	Ejemplos 1 y 2 de maximización - Símplex	30
3.2.	Ejemplo 3 de maximización - Símplex	31
5.1.	Función de activación ReLU	64
5.2.	Ejemplo de convolución	68
5.3.	Arquitectura de red neuronal LeNet5	69
6.1.	Caso de uso de Aprendizaje Federado	72
6.2.	Flujo y pasos de un escenario de Aprendizaje Federado	74
6.3.	Estructura de cliente-servidor en Aprendizaje Federado	76
6.4.	Categorización de los escenarios de Aprendizaje Federado	78
6.5.	Esquema de aplicación de Privacidad Diferencial en <i>Deep Learning</i>	79
6.6.	Ejemplo de mecanismo ingenuo de Privacidad Diferencial	79
6.7.	Definición de Privacidad Diferencial	81
6.8.	Clasificación de ataques según su objetivo	86
6.9.	Taxonomía para los ataques al modelo	87
6.10.	Taxonomía para los ataques a la privacidad	90
8.1.	Ejemplo de las muestras de MNIST	97
8.2.	Ejemplo de las muestras de CIFAR10	98
8.3.	Ejemplo de las muestras de FashionMNIST	98
8.4.	Escenario de <i>Label Flipping</i>	100
8.5.	Patrón añadido en el ataque <i>backdoor</i>	102
8.6.	Comparativa de la reconstrucción de dos muestras de la GAN	103
9.1.	Resultados del ataque <i>Label Flipping</i> en distintos datasets	108
9.2.	Resultados de <i>Label Flipping</i> para distinto número de adversarios	109
9.3.	Resultados del ataque <i>gaussiano</i> en distintos datasets	111
9.4.	Resultados del ataque <i>gaussiano</i> para distinto número de adversarios	112
9.5.	Resultados del ataque <i>backdoor</i> en distintos datasets	114
9.6.	Resultados del ataque <i>backdoor</i> para distinto número de adversarios	115
9.7.	Resultados de DMUG	116

Índice de tablas

1.	Diagrama de Gantt detallado	xxiii
2.	Tabla de presupuesto del trabajo	xxiii
8.1.	Resumen de los conjuntos de datos	98
8.2.	Configuración de los experimentos	105
8.3.	Tiempos de ejecución para la experimentación	106
9.1.	Resumen de <i>Label Flipping</i> en distintos datasets	109
9.2.	Resumen del ataque <i>gaussiano</i> para los tres datasets	110
9.3.	Resumen del ataque <i>backdoor</i>	113
9.4.	Resumen de DMUG	116

Introducción

1. Contexto

Nos encontramos en la era de la **Inteligencia Artificial (IA)** y, en este contexto, el Aprendizaje Automático o **Machine Learning (ML)** y el Aprendizaje Profundo o **Deep Learning (DL)** han sido dos de las áreas de investigación más activas en los últimos años. Tomar decisiones basadas en aprendizaje a través de datos de múltiples fuentes ha sido clave en avances tanto científicos como en el ámbito tecnológico y empresarial.

En ese sentido, surge un cambio de paradigma lógico, motivando el uso de **ML** distribuido frente al clásico centralizado. Consecuentemente, la privacidad de los datos se ha convertido en un tema de crucial importancia [Nad18, TTS15]. Sin embargo, en el aprendizaje automático distribuido, los costos de comunicación siguen siendo mucho mayores que los de cómputo, haciendo el proceso de entrenamiento muy ineficiente.

En este contexto surge el Aprendizaje Federado o **Federated Learning (FL)**, propuesto por *Google* en el año 2017, en [MMR⁺17a]. Se trata de un enfoque de entrenamiento totalmente descentralizado que permite aprender de los datos sin que estos se encuentren en un servidor central, como se venía haciendo en la mayoría de escenarios. De ahí la privacidad brindada por **FL**, pues los datos nunca abandonan el dispositivo de origen.

Sin embargo, siempre que existe una comunicación entre dos partes, existe también un riesgo en la privacidad de las mismas. En esta línea, es primordial establecer cuáles son los principales escenarios de riesgo, así como parámetros y consejos a seguir para tratar de mantener espacios seguros dentro de ellos. Es justo esto lo que hizo la Unión Europea en el llamado **Ley o Acto de Inteligencia Artificial de la UE (AI Act)** [Eur24], propuesto en 2021 y aprobado desde 2024. Esta propuesta legislativa pretende que la regularización, comercialización y uso de sistemas de **IA** se haga de forma segura, siguiendo unos requisitos y normas necesarias.

El **AI Act** establece un sistema normativo que prioriza la seguridad, la transparencia y la protección de los derechos fundamentales en el desarrollo y uso de sistemas de **IA**. De hecho, dentro de los cuatro niveles de riesgo que se definen en el acto (ver Figura 1), el **FL** resulta especialmente valioso en sistemas de *alto riesgo*, ya que permite entrenar modelos sin transferir datos personales, facilitando el cumplimiento de los requisitos de privacidad, trazabilidad y supervisión humana. En niveles de *riesgo limitado o mínimo*, también refuerza la confianza del usuario al evitar la centralización de datos. Así, el **FL** se alinea con el enfoque del **AI Act**, promoviendo una **IA** segura y responsable.

Con el objetivo de mejorar la eficiencia del proceso de aprendizaje distribuido, siempre manteniendo la mayor privacidad posible, surge este trabajo. Se pretende combinar el **FL** con la optimización de parámetros de la **Privacidad Diferencial (PD)**, el mecanismo que se estudia en esta memoria que trata de preservar la privacidad en los escenarios de **FL**.

Véase ahora el siguiente caso real, que pone en evidencia la importancia de un buen mecanismo de privacidad y su criticidad en escenarios aparentemente seguros.

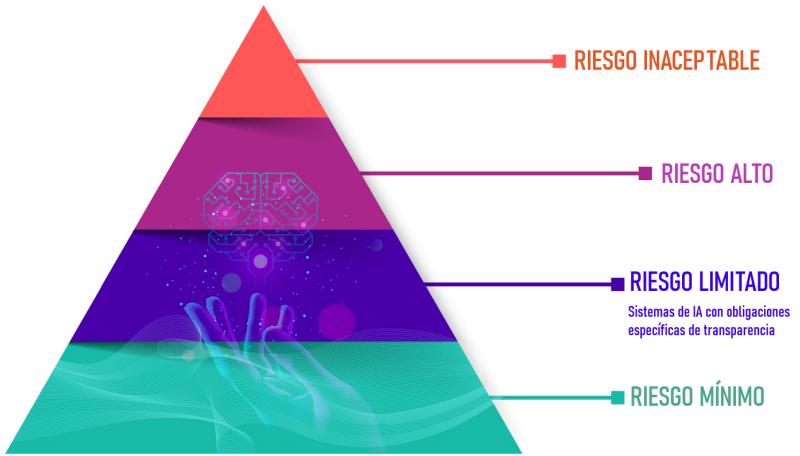


Figura 1.: Pirámide con los cuatro niveles de riesgo para sistemas de IA, según la Unión Europea. Fuente de la imagen: [Eur24].

El caso Netflix

En el año 2006, Netflix lanzó una competición llamada *The Netflix Price*. Los equipos apuntados a la competición debían crear un algoritmo capaz de predecir cómo una persona aleatoria puntuaría una película. Para ello, la gran plataforma de *streaming* suministró un conjunto de datos del orden de 100 millones de puntuaciones, de alrededor de 480 mil usuarios distintos. Una cantidad de datos suficiente para poder crear este tipo de algoritmos.

El proceso de “anonymización” de los datos hecho por Netflix consistió en eliminar el nombre de los usuarios de las puntuaciones, y reemplazar algunas de estas por puntuaciones falsas de manera aleatoria. En una primera instancia, se podría pensar que este proceso era seguro. Sin embargo, en 2008, Arvid Narayanan y Vitaly Shmatikov, de la Universidad de Texas, demostraron en [NSo8] que habían identificado usuarios reales a partir de este conjunto de datos aparentemente anónimo. Se llevó a cabo un denominado *Linkage Attack*, que trata de identificar entidades en conjuntos de datos combinando los datos de otros conjuntos conocidos.

Este incidente dejó en evidencia una problemática fundamental en la publicación de datos: el hecho de eliminar identificadores directos (como nombres o direcciones de correo electrónico) no garantiza la privacidad de los mismos. A pesar de que los datos parecían anónimos, fue posible reidentificar usuarios cruzando la información con otros conjuntos de datos públicos, como bases de datos de puntuaciones de películas publicadas en sitios como *IMDb*. Esto ilustra que la simple anonimización no es suficiente cuando se trata de proteger la privacidad individual en presencia de información auxiliar externa.

2. Motivación

La privacidad en el ámbito de los datos ha adquirido una importancia creciente, situándose en el núcleo de numerosas investigaciones. La vulnerabilidad derivada de los avances tecnológicos ha convertido la protección de la información en una necesidad fundamental. Asimismo, la creciente concienciación de los usuarios sobre la sensibilidad de sus datos ha

contribuido a que este tema cobre una relevancia cada vez mayor en el ámbito académico y profesional.

En el contexto de la problemática anterior (*El caso Netflix*) es donde surge la necesidad de métodos más sólidos y matemáticamente fundados para proteger la privacidad de los individuos. Uno de los enfoques más prometedores y ampliamente adoptados en la actualidad es la mencionada **PD**, propuesta por Cynthia Dwork y otros investigadores en [DR⁺14], donde establecen los fundamentos más teóricos y matemáticos de este mecanismo, evidenciando por qué realmente funciona.

La **PD** propone una definición rigurosa de privacidad, basada en la idea de que la inclusión o exclusión de cualquier individuo en un conjunto de datos no debe afectar significativamente los resultados de los análisis realizados sobre el mismo. En otras palabras, un mecanismo que respeta la **PD** garantiza que un atacante no pueda inferir si un individuo en particular está o no en la base de datos, incluso si tiene acceso a información externa.

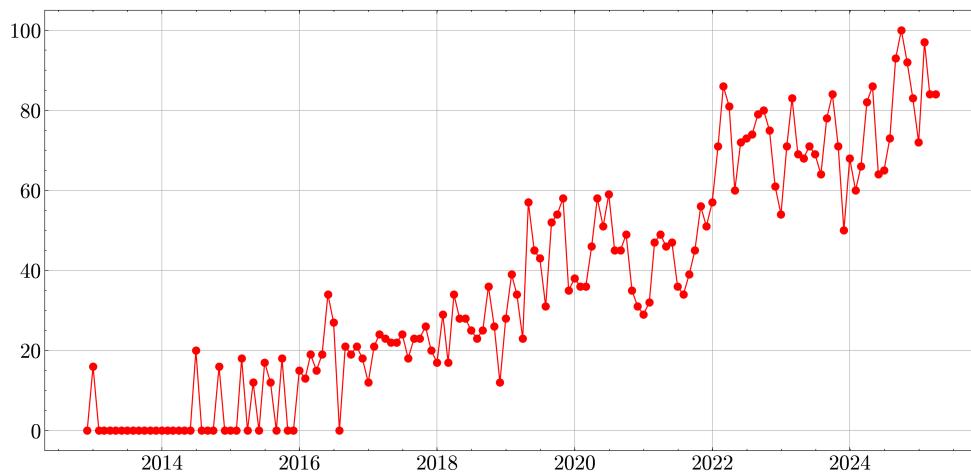


Figura 2.: Interés en **PD** a lo largo de los años 2013-2025, según los datos recogidos por *Google Trends*. Un valor de 100 indica la popularidad máxima de un término, mientras que 50 y 0 indican que un término es la mitad de popular en relación con el valor máximo o que no había suficientes datos del término, respectivamente.

A través del uso de técnicas probabilísticas (como la adición de ruido calibrado a las respuestas de consultas estadísticas) la **PD** permite extraer valor de los datos sin comprometer la privacidad de los individuos.

Varias empresas tecnológicas líderes han implementado la **PD** como estrategia fundamental para proteger la información de los usuarios, combinando innovación técnica con cumplimiento normativo. Este enfoque se caracteriza por su aplicación en sistemas operativos y servicios en la nube, entre otros. Empresas multinacionales como *Google* y *Apple* dedican grandes esfuerzos de investigación en esta materia.

Como contrapartida a este método, se ha visto que el uso del mismo conlleva una gran pérdida en el rendimiento de los modelos entrenados [BPS19]. Es una técnica que se ha de usar con cautela, pues implica un impacto importante en el aprendizaje.

Por tanto, una vez establecida la importancia de la **PD** en el **FL**, surge la necesidad de encontrar un equilibrio adecuado entre privacidad y utilidad. En este contexto, la **utilidad** se puede definir como el grado en que el modelo resultante conserva su precisión o rendimiento

Introducción



Figura 3.: Interés en **FL** a lo largo de los años 2019-2025, según los datos recogidos por *Google Trends*.

al mismo tiempo que se preserva la privacidad de los datos. Este equilibrio se ve condicionado por ciertos parámetros de la **PD**, cuya elección influye directamente en el rendimiento del modelo.

Para abordar la problemática anterior, resulta natural recurrir a técnicas de **optimización**. Se busca, por una parte, obtener como resultado un modelo de calidad entrenado de manera colaborativa y *maximizando* su rendimiento, mientras que, a la vez, se trata de preservar la privacidad de los participantes. En particular, la programación lineal y convexa ofrecen herramientas potentes y más que suficientes para modelar y resolver problemas de este tipo, permitiendo formalizar objetivos y restricciones de forma eficiente y tratable.

Diversos estudios han demostrado que problemas de optimización relacionados con la **PD** pueden formularse y resolverse eficazmente mediante programación convexa. Por ejemplo, Yuan et al. [YYZH16] proponen un enfoque basado en optimización convexa para el procesamiento de consultas lineales bajo ciertas restricciones de **PD**, logrando soluciones óptimas con sólidas garantías de convergencia. Asimismo, Dvorkin et al. [DFVH⁺20] desarrollan un marco novedoso que combina programación estocástica y **PD** para resolver problemas de optimización convexa con datos sensibles, asegurando la factibilidad de las soluciones obtenidas.

Estos enfoques serán la vía a seguir en este trabajo y destacan la viabilidad y eficacia de aplicar técnicas de optimización lineal y convexa en el diseño de algoritmos de **FL** que respeten la **PD**, facilitando un balance adecuado entre la protección de la privacidad y el mantenimiento de la utilidad del modelo. En concreto, la **Programación Lineal (PL)** será crucial en la experimentación realizada en este trabajo y será el punto de unión o nexo entre la parte matemática y la informática. Se hará uso del famoso método de optimización *Simplex*, que, pese a ser un método de los más clásicos, se comprobará que cumplirá perfectamente con su propósito, que será optimizar los parámetros que definen la **PD**.

Haciendo síntesis de todo lo anterior, la idea general de este TFG es encontrar un balance entre la utilidad y la privacidad brindada por la **PD**. Para ello, se llevan a cabo distintos experimentos, simulando distintos ataques al **FL** ya conocidos, tratando de optimizar la calidad del aprendizaje y preservando la privacidad de los datos en todo momento.

3. Estructura del trabajo

Este Trabajo de Fin de Grado se divide en dos bloques principales, reflejo de la doble vertiente en la que se sustenta el proyecto: por un lado, una base teórica sólida desde el ámbito matemático de la optimización; y por otro, una aplicación práctica en el contexto del **ML** y, en particular, del **FL** con mecanismos de protección de la privacidad.

- **Primera parte: Fundamentos Matemáticos y Teoría de la Optimización.** Se presentan los conceptos esenciales de álgebra lineal, geometría convexa y teoría de la optimización necesarios para formalizar el problema. Se estudian con rigor resultados como el Teorema de Minkowski-Carathéodory, el Teorema de Krein-Milman y el Principio del Máximo de Bauer. Se detallan los métodos de optimización utilizados, haciendo especial hincapié en la **PL** y el algoritmo Símplex, que será la herramienta clave en la parte aplicada.
- **Segunda parte: Aplicación en FL con PD.** Esta sección aborda el marco teórico y computacional de ambos conceptos. Se formalizan los flujos de trabajo, arquitecturas y tipos de amenazas en **FL**, así como las propiedades y mecanismos de **PD**. A continuación, se describe cómo se integra la optimización lineal en la calibración o aplicación dinámica de los parámetros de **PD** durante el entrenamiento federado, buscando un equilibrio entre precisión y privacidad.
- **Entorno experimental y análisis.** Se detallan los escenarios experimentales planteados, incluyendo los conjuntos de datos utilizados y los distintos tipos de ataques simulados. Se realiza un análisis comparativo entre distintos niveles de protección de privacidad y su impacto en la resiliencia del modelo frente a ataques, evaluando tanto rendimiento como gasto de privacidad.
- **Conclusiones y trabajos futuros.** Finalmente, se resumen los principales hallazgos del trabajo y se proponen posibles líneas de continuación, incluyendo mejoras metodológicas, generalización a otros tipos de ataques y enfoques alternativos de optimización.

Esta organización permite una lectura progresiva, partiendo desde fundamentos teóricos generales hasta llegar a aplicaciones concretas y análisis cuantitativos, con el objetivo de ofrecer una visión integral y justificada del problema abordado.

4. Objetivos

El principal objetivo de este Trabajo de Fin de Grado es desarrollar un sistema que permita optimizar dinámicamente los parámetros de privacidad diferencial durante el entrenamiento de modelos en un entorno de Aprendizaje Federado. El fin último es lograr un equilibrio entre la protección de la privacidad de los datos individuales y el rendimiento del modelo global.

Para alcanzar este propósito general, se plantean los siguientes objetivos específicos:

- Estudiar los fundamentos teóricos de la **PL** y su aplicación en problemas de optimización con restricciones, con especial atención al método del Símplex.
- Analizar el funcionamiento del **FL** y los principales riesgos a la privacidad asociados, incluyendo ataques al modelo y ataques de reconstrucción de datos.

- Profundizar en los principios matemáticos de la **PD**, así como en sus mecanismos y propiedades clave, con el objetivo de comprender su impacto sobre el rendimiento de los modelos de **FL**.
- Diseñar un sistema de optimización que, en base a la **PL**, ajuste dinámicamente los parámetros de privacidad diferencial (ϵ, δ) en función de las métricas obtenidas durante el entrenamiento.
- Implementar un entorno experimental que simule distintos ataques a la privacidad en **FL**, y que permita comparar el rendimiento del modelo bajo distintas configuraciones de parámetros de **PD** (estáticos frente a optimizados dinámicamente).
- Evaluar empíricamente el impacto del sistema de optimización propuesto, mediante métricas que midan tanto la utilidad del modelo como el nivel de privacidad alcanzado.

En conjunto, estos objetivos buscan demostrar la viabilidad de integrar técnicas clásicas de optimización matemática con herramientas modernas de **ML** para mejorar la privacidad en escenarios colaborativos de entrenamiento de modelos.

5. Metodología y planificación

Un Trabajo de Fin de Grado en el Doble Grado de Ingeniería Informática y Matemáticas consta de 18 créditos ECTS. En [Fac25] se establece que cada crédito es equivalente a 25 horas de trabajo. Por esto, en principio, el desarrollo de este trabajo equivaldría a 450 horas en cómputo total. Sin embargo, el proyecto se empezó en febrero de 2025 y se han dedicado unas 26 – 28 horas a la semana. Por tanto, se estima que este trabajo ha recibido alrededor de unas 600 horas de trabajo, desde febrero hasta principios de julio.

Se divide el desarrollo de la planificación en:

- **Lectura e investigación:** La idea en la planificación de todo proyecto de índole investigativa es tratar de dedicar unos primeros esfuerzos a la pura investigación, lectura y asimilación de conceptos, literatura y teoría necesaria, para luego llevar a la práctica todo el conocimiento adquirido. Es por eso que la fase siempre debe ser la lectura de artículos y trabajos ya realizados.
- **Diseño:** Con el fin de tener una planificación organizada y poder trabajar de manera continua, se llevan a cabo reuniones con los tutores académicos y se planea el modo de trabajo.
- **Software e implementación de la experimentación:** Las primeras semanas que aparecen en la Tabla 1 (17 marzo - 21 abril) sirvieron para familiarizarse con el entorno de trabajo y lanzar algunas pruebas simples. El resto se usó para el propio desarrollo e implementación de los ataques propuestos en este documento.
- **Redacción:** Se comienza en marzo la parte de matemáticas de la memoria, para tratar de tener al menos la parte lineal totalmente desarrollada para cuando se pasase a la experimentación. Es en la segunda semana de julio cuando se da por terminada la redacción, tras haber refinado y modificado todo lo necesario.

- **Revisión:** A lo largo de la redacción e implementación de software, se ha ido haciendo una revisión por parte de los dos tutores de distintas versiones del trabajo, llevando esto a realizar cambios continuos sobre el trabajo realizado, en pos de una buena calidad y de no arrastrar errores.

La división realizada para la planificación queda reflejada en la Tabla 1, en la que se muestra un diagrama de *Gantt* con todas las semanas desde febrero hasta julio. Se trata de una estimación del desglose del desarrollo del Trabajo de Fin de Grado.

Gantt	Febrero				Marzo				Abril				Mayo				Junio				Julio			
	3	10	17	24	3	10	17	24	31	7	14	21	28	5	12	19	26	2	9	16	23	30	7	14
Lectura																								
Diseño																								
Software																								
Redacción																								
Revisión																								

Tabla 1.: Diagrama de Gantt del desarrollo del TFG desglosado por semanas de febrero a julio de 2025.

Este trabajo se pensaba entregar en la convocatoria de junio; sin embargo, por falta de tiempo para refinar esta memoria y el trabajo en general, se decide entregar en la convocatoria de julio. Esto hace que, como se ve en la Tabla 1, se dedique más tiempo a la redacción y revisión del trabajo.

Por otra parte, se ha de hacer también una estimación del **coste de trabajo**. Para ello, podemos comenzar haciendo una estimación basada en el sueldo medio de un investigador *junior* en España, como se hace en [Gla25]. Basándonos en esto, pongamos un sueldo de 25,000€ anuales, se reduce a alrededor de 12,82€ la hora.

Teniendo en cuenta también el *hardware* usado, se han de contar también los costes del equipo usado para el desarrollo del trabajo. Con la misma filosofía de estimar un presupuesto para el trabajo, aunque el uso de servidores GPU en la nube haya sido totalmente gratis, añade riqueza a la estimación el saber cuánto costaría el uso de estos servicios si no fueran gratis en un caso de uso real. Todo esto queda reflejado en la Tabla 2.

Item	Coste Total
Sueldo medio investigador <i>junior</i>	7.692,00€
Portátil personal gama media	800,00€
Uso de GPU	96,80€ ¹
Total	8.588,8€

Tabla 2.: Resumen de los costes para el presupuesto del trabajo.

¹Según lo discutido en el uso de GPU realizado en este trabajo, hecho en la Tabla 8.3, son un total de 110 horas. Según [Goo19], el precio de estos servicios ofrecidos por Google es en torno a unos 0.88€ la hora.

Parte I.

Optimización lineal y convexa

1. Preliminares y fundamentos teóricos

1.1. Introducción

En el desarrollo teórico de las matemáticas, resulta crucial partir de una base bien consolidada, así como de unos fundamentos sólidos que puedan respaldar todo el trabajo que se realice. Es por esto que esta sección introductoria está dedicada a tratar los fundamentos matemáticos necesarios que sustentan los cimientos teóricos de la **PL** y **Programación Convexa (PC)** y sus problemas derivados, además de fijar una notación común para todo el documento. Como se ha visto a lo largo del grado, es indispensable este tipo de rigurosidad si se pretende entender en profundidad el trabajo desarrollado y no caer en inexactitudes matemáticas.

Muchos de los resultados y recordatorios hechos en esta sección, servirán para entender en su integridad los resultados más relevantes que se desarrollarán más tarde. Dentro de estos, está, por ejemplo, el método Simplex, usado para solucionar de manera eficiente ciertos problemas de **PL** o el Teorema de *Krein-Milman*, fundamental en el desarrollo de la optimización convexa en el ámbito infinito-dimensional.

En esta sección introducimos los conceptos y resultados fundamentales del álgebra lineal, el Análisis Funcional y la teoría de optimización que serán recurrentemente utilizados. Asimismo, se establecerá y fijará una notación común que facilitará la lectura y comprensión de los enunciados posteriores.

1.2. Preliminares

En general, se necesitarán conocimientos sólidos en los conceptos matemáticos de relevancia tanto para la parte de informática del trabajo, como para la parte del desarrollo teórico de los distintos tipos de optimización que se van a abordar en el trabajo. Por ello, se recordarán a continuación muchos conceptos estudiados y desarrollados durante el grado. Comenzamos tratando los conceptos más simples:

- Trabajaremos continuamente con elementos dentro del espacio euclídeo \mathbb{R}^n , $n \geq 2$. Utilizaremos la norma euclídea en dicho espacio y su distancia asociada, que recordamos viene dada por:

$$d(x, y) = \|x - y\| = \left(\sum_{i=1}^n (x_i - y_i)^2 \right)^{1/2} \quad (1.1)$$

- Trataremos con **vectores** como elementos de \mathbb{R}^n , en forma de columna. Es decir, tendrán la forma:

$$v = \begin{pmatrix} v_1 \\ v_2 \\ \vdots \\ v_n \end{pmatrix}$$

1. Preliminares y fundamentos teóricos

donde cada v_i es una componente del vector en \mathbb{R} . Denotaremos en lo que sigue también con $\langle \cdot \rangle$ el producto escalar usual de vectores. La mayoría de los conceptos presentados y por presentar también se pueden definir sobre el conjunto \mathbb{C} , sin embargo, nos restringiremos únicamente al caso real, pues es el que aplica en la problemática de este trabajo.

- De igual manera trabajaremos con matrices con elementos en \mathbb{R} , y nos referiremos a sus elementos mediante el uso de subíndices. Por ejemplo, si tenemos una matriz de orden $m \times n$, el elemento a_{ij} , con $i \leq m$ y $j \leq n$, corresponderá al valor real en la fila i y la columna j . Se debe recordar también el concepto de **inversa** de una matriz y el concepto de **matriz invertible**, así como todos los conceptos básicos de matrices, pues nos serán de gran utilidad en el desarrollo de este trabajo.
- Se recuerda que una matriz se dice *no singular* cuando su determinante es distinto de 0. En otras palabras, una matriz es no singular si es **invertible**, lo que significa que existe otra matriz que, al multiplicarse con la matriz de partida, da como resultado la matriz identidad, que denotaremos por I .

Ahora consideramos un sistema de ecuaciones lineales

$$Ay = b \quad (1.2)$$

donde A es una matriz de orden $m \times n$, de rango m , y es un vector en \mathbb{R}^n , b es un vector en \mathbb{R}^m . Otro sistema de ecuaciones lineales se dice **equivalente** si el conjunto de soluciones (valores para y que cumplen la igualdad) es igual para ambos sistemas. Trabajaremos continuamente con sistemas de ecuaciones lineales en el desarrollo de los problemas de **PL**. De igual manera, se pueden definir los sistemas si en lugar de una igualdad usamos una desigualdad.

Con objeto de poder operar con matrices y por ende con sistemas de ecuaciones lineales, recordamos ahora el concepto de **operación elemental** en el sistema anterior, que son las siguientes:

- (I) Intercambiar dos ecuaciones.
- (II) Multiplicar una ecuación por una constante distinta de 0.
- (III) Sumar un múltiplo (distinto de 0) de una ecuación a otra.

Cualquier secuencia de operaciones elementales aplicadas a un sistema de ecuaciones lineales produciría un nuevo sistema equivalente al inicial. Por último, se recuerdan los métodos de Gauss y Gauss-Jordan, pues servirán posteriormente en el desarrollo del método Simplex.

1.2.1. Método de Gauss

Se describe ahora el **proceso de eliminación de Gauss**, que nos permitirá llegar a la solución general del sistema. Empezamos denotando por R a la submatriz de A de dimensión $k \times k$, de rango k , y sea S la submatriz de A formada por las restantes $n - k$ columnas de A . Entonces podemos escribir (1.2) como

$$(R, S) \begin{pmatrix} y_r \\ y_s \end{pmatrix} = b$$

donde y_r es el vector obtenido de y escogiendo las componentes que corresponden a las columnas en R e y_s análogamente para S . Entonces podemos usar las operaciones elementales definidas en la sección anterior para obtener un sistema equivalente, con la forma

$$(U, V) \begin{pmatrix} y_r \\ y_s \end{pmatrix} = b' \quad (1.3)$$

donde U es una matriz triangular superior $k \times k$ con todos los elementos de la diagonal principal iguales a 1 y V es una matriz de dimensiones $k \times (n - k)$. Para simplificar la notación, asumimos que la submatriz R consiste en las primeras k columnas de A , aunque podría ser de otra forma. Entonces, las componentes de (1.3) pueden escribirse como:

$$\begin{aligned} y_1 + x_{12}y_2 + \cdots + x_{1k}y_k + x_{1,k+1}y_{k+1} + \cdots + x_{1n}y_n &= b'_1, \\ y_2 + \cdots + x_{2k}y_k + x_{2,k+1}y_{k+1} + \cdots + x_{2n}y_n &= b'_2, \\ &\vdots \\ y_k + x_{k,k+1}y_{k+1} + \cdots + x_{kn}y_n &= b'_k. \end{aligned} \quad (1.4)$$

Así, si tomamos la última ecuación de (1.4), esta nos expresa y_k en términos de y_{k+1}, \dots, y_n . Si se sustituye hacia atrás, podemos expresar y_1, \dots, y_k en términos de y_{k+1}, \dots, y_n y así obtener una **solución general** de (1.3) y por tanto de (1.2).

1.2.2. Método de Eliminación de Gauss-Jordan

El **método de eliminación de Gauss-Jordan** es una extensión del método de eliminación de Gauss, que no solo transforma la matriz del sistema en una forma triangular superior, sino que la lleva a su forma reducida por filas. Esto significa que la matriz de coeficientes se convierte en la matriz identidad en las primeras k columnas, facilitando la obtención directa de la solución sin necesidad de sustitución hacia atrás.

Partimos de la ecuación obtenida en el método de Gauss:

$$(U, V) \begin{pmatrix} y_r \\ y_s \end{pmatrix} = b'$$

donde U es una matriz triangular superior de dimensión $k \times k$ con unos en la diagonal principal, y V es una matriz de dimensiones $k \times (n - k)$.

El procedimiento de **Gauss-Jordan** continúa aplicando operaciones elementales por filas para eliminar los coeficientes por encima de la diagonal principal en U , de modo que la matriz resultante sea de la forma:

$$(I, V') \begin{pmatrix} y_r \\ y_s \end{pmatrix} = b''$$

donde I es la matriz identidad de dimensión $k \times k$ y V' es una matriz transformada.

1. Preliminares y fundamentos teóricos

Así, el sistema puede escribirse como:

$$y_r + V'y_s = b''$$

lo que nos da una solución explícita para y_r :

$$y_r = b'' - V'y_s.$$

Esto significa que cualquier solución del sistema depende de los valores libres en y_s , lo que proporciona la solución general del sistema.

El método de Gauss-Jordan es especialmente útil cuando se requiere obtener la inversa de una matriz o resolver sistemas con múltiples vectores de términos independientes simultáneamente. Vamos a ver justo cómo obtener la inversa de una matriz en la siguiente sección.

1.2.3. Cálculo de la inversa de una matriz invertible

Para calcular la inversa de una matriz invertible R (submatriz de A de dimensión $k \times k$, de rango k), podemos utilizar una secuencia de operaciones elementales que transformen R en la matriz identidad I . Esto se basa en el hecho de que cada operación elemental sobre R equivale a una premultiplicación por una matriz elemental E , la cual es siempre invertible.

Una matriz cuadrada E se dice **elemental** cuando se obtiene aplicando una única operación elemental de fila a la matriz identidad. Estas matrices son fundamentales en la transformación de matrices mediante operaciones elementales y en el cálculo de la inversa de una matriz. Vamos entonces con el procedimiento del cálculo de la inversa de una matriz invertible:

Dado que R es no singular, existe una secuencia de matrices elementales E_p, E_{p-1}, \dots, E_1 tal que:

$$E_p E_{p-1} \dots E_1 R = I.$$

Multiplicando por la derecha por R^{-1} , obtenemos:

$$E_p E_{p-1} \dots E_1 I = R^{-1}.$$

Esto implica que la inversa de R es el producto de las matrices elementales aplicadas en orden.

Para calcular R^{-1} , seguimos estos pasos:

1. Construimos la matriz aumentada $(R | I)$.
2. Aplicamos operaciones elementales fila a fila sobre R hasta reducirlo a la matriz identidad I .
3. Las mismas operaciones aplicadas en I transforman esta parte en R^{-1} , resultando en la matriz $(I | R^{-1})$.

Por lo tanto, el bloque resultante en la parte derecha de la matriz aumentada es la inversa buscada.

Este método de cálculo de la inversa es útil en la implementación del método Simplex, discutido más adelante, donde la llamada *forma producto* de la inversa se utiliza para actualizar la base de soluciones de forma eficiente.

1.3. Conjuntos afines

Antes de introducir los conjuntos convexos, es importante entender una estructura más básica y fundamental en geometría y álgebra lineal: *los conjuntos afines*. Resulta crucial entender este concepto, pues generaliza al de conjunto convexo, ya que en un conjunto convexo solo se permiten combinaciones afines con coeficientes no negativos (combinaciones convexas). Por tanto, todo conjunto convexo está contenido en algún conjunto afín, pero no todo conjunto afín es necesariamente convexo.

Proposición 1.1. *Un conjunto $A \subset \mathbb{R}^n$ se dice afín si para cualesquiera $x_1, x_2 \in A$ y cualquier escalar $\lambda \in \mathbb{R}$, el punto*

$$\lambda x_1 + (1 - \lambda)x_2 \in A$$

De manera más general, un conjunto es afín si contiene todas las combinaciones afines de sus puntos.

Justamente, una combinación afín de los puntos $x_1, \dots, x_k \in \mathbb{R}^n$ es una expresión de la forma:

$$\sum_{i=1}^k \lambda_i x_i \quad \text{con} \quad \sum_{i=1}^k \lambda_i = 1, \lambda_i \in \mathbb{R}$$

Proposición 1.2. *La envolvente afín de un conjunto A es el conjunto formado por todos los puntos de la forma*

$$x = \lambda_1 x_1 + \dots + \lambda_k x_k,$$

tales que $x_i \in A$, $\lambda_1, \dots, \lambda_k \in \mathbb{R}$ con $\lambda_1 + \dots + \lambda_k = 1$, y k es un número natural. Se denota por $\text{aff}(A)$.

Demostración. Sea M el conjunto de todos los puntos de la forma anterior. Si $x, y \in M$, por ejemplo:

$$x = \sum_{i=1}^k \lambda_i x_i, \quad y = \sum_{j=1}^h \mu_j y_j, \quad \text{con } x_i, y_j \in A, \quad \sum_{i=1}^k \lambda_i = \sum_{j=1}^h \mu_j = 1,$$

entonces, para cualquier $\alpha \in \mathbb{R}$, podemos escribir:

$$(1 - \alpha)x + \alpha y = \sum_{i=1}^k (1 - \alpha)\lambda_i x_i + \sum_{j=1}^h \alpha \mu_j y_j,$$

donde:

$$\sum_{i=1}^k (1 - \alpha)\lambda_i + \sum_{j=1}^h \alpha \mu_j = (1 - \alpha) + \alpha = 1,$$

1. Preliminares y fundamentos teóricos

por lo tanto, $(1 - \alpha)x + \alpha y \in M$. Así, M es un conjunto afín, y en consecuencia $M \supseteq \text{aff}(A)$.

Por otro lado, es fácil ver que si

$$x = \sum_{i=1}^k \lambda_i x_i, \quad \text{con } x_i \in A, \quad \sum_{i=1}^k \lambda_i = 1,$$

entonces $x \in \text{aff}(A)$. (Para $k = 2$ esto se deduce directamente de la definición de conjuntos afines; para $k \geq 3$ se deduce por inducción).

Por lo tanto, $M \subseteq \text{aff}(A)$, y así, $M = \text{aff}(A)$. □

Se recuerda al lector que $x_1, \dots, x_k \in A$ se dicen afínmente independientes si los vectores $x_2 - x_1, \dots, x_k - x_1$ son linealmente independientes.

Lema 1.3. *Sea A un conjunto afín. Los puntos $x_1, \dots, x_k \in A$ son afínmente dependientes si y sólo si existen $\alpha_1, \dots, \alpha_k \in \mathbb{R}$ no todos nulos tales que $\sum_{i=1}^k \alpha_i = 0$ y $\sum_{i=1}^k \alpha_i x_i = 0$.*

Demostración. Si los puntos $x_1, \dots, x_k \in A$ son afínmente dependientes, entonces los vectores $x_2 - x_1, \dots, x_k - x_1$ son linealmente dependientes. Por tanto, existen $\lambda_2, \dots, \lambda_k \in \mathbb{R}$ no todos nulos tales que

$$0 = \sum_{i=2}^k \lambda_i(x_i - x_1) = \sum_{i=2}^k \lambda_i x_i + \alpha_1 x_1,$$

donde $\alpha_1 = -\sum_{i=2}^k \lambda_i$. Llamamos $\alpha_i = \lambda_i \forall i = 2, \dots, k$, que cumplen las condiciones deseadas.

Recíprocamente, supongamos que existen $\alpha_1, \dots, \alpha_k \in \mathbb{R}$ no todos nulos tales que $\sum_{i=1}^k \alpha_i = 0$ y $\sum_{i=1}^k \alpha_i x_i = 0$. Entonces,

$$\sum_{i=2}^k \alpha_i(x_i - x_1) = \sum_{i=2}^k \alpha_i x_i - \left(\sum_{i=2}^k \alpha_i \right) x_1 = \sum_{i=2}^k \alpha_i x_i + \alpha_1 x_1 = 0,$$

luego los vectores $x_2 - x_1, \dots, x_k - x_1$ son linealmente dependientes. □

Corolario 1.4. *La envolvente afín S de un conjunto de k puntos afínmente independientes $\{x_1, \dots, x_k\}$ en \mathbb{R}^n es un conjunto afín de dimensión $(k - 1)$. Todo punto $x \in S$ puede representarse de forma única como:*

$$x = \sum_{i=1}^k \lambda_i x_i, \quad \sum_{i=1}^k \lambda_i = 1.$$

Demostración. Por la Proposición 1.2, todo punto $x \in S$ tiene la forma presentada. Si

$$x = \sum_{i=1}^k \mu_i x_i$$

es otra representación, entonces, usando que $\sum_{i=1}^k \lambda_i x_k = \sum_{i=1}^k \mu_i x_k$, obtenemos:

$$\sum_{i=1}^k (\lambda_i - \mu_i)(x_i - x_k) = 0,$$

por lo tanto, $\lambda_i = \mu_i$ para todo $i = 1, \dots, k$. □

1.4. Conjuntos convexos en espacios normados

Con la finalidad de concretar y trabajar finalmente en \mathbb{R}^n en las próximas secciones, se han de introducir correctamente las generalizaciones de conjunto convexo y conjunto extremal para espacios normados, entre otras. Empezamos recordando las siguientes definiciones de manera rápida pues es algo conocido y ya trabajado en el grado.

- Un *espacio normado* es un par $(X, \|\cdot\|)$, donde X es un espacio vectorial sobre \mathbb{R} o \mathbb{C} , y $\|\cdot\| : X \rightarrow \mathbb{R}$ es una función que asigna a cada vector su norma, cumpliendo:
 - $\|x\| \geq 0$ y $\|x\| = 0 \iff x = 0$,
 - $\|\lambda x\| = |\lambda| \cdot \|x\|$,
 - $\|x + y\| \leq \|x\| + \|y\|$ (desigualdad triangular),
 para todo $x, y \in X, \lambda \in \mathbb{R}$ (o \mathbb{C}).
- Sea X un espacio normado. Un subconjunto $C \subseteq X$ se dice **convexo** si para todo $x, y \in C$ y todo $\lambda \in [0, 1]$, se cumple:

$$\lambda x + (1 - \lambda)y \in C.$$

- Sea $C \subseteq X$ un conjunto convexo. Un punto $x \in C$ se dice **punto extremal o extremo** de C si no puede escribirse como combinación convexa no trivial de otros dos puntos de C . Es decir, si

$$x = \lambda y + (1 - \lambda)z \quad \text{con } y, z \in C, \lambda \in (0, 1),$$

entonces se tiene que $y = z = x$.

En lo siguiente, al conjunto de los puntos extremos de un conjunto K lo llamaremos $ex(K)$.

- Un subconjunto $E \subseteq C$ se dice **conjunto extremal** en C si para toda combinación convexa $x = \lambda y + (1 - \lambda)z \in E$, se tiene que $y, z \in E$. En otras palabras, E es un subconjunto extremal de C si siempre que un segmento con extremos en C tenga un punto intermedio en E , los extremos de dicho segmento pertenecen a E .
- Dado un conjunto de puntos $x_1, x_2, \dots, x_k \in X$, una **combinación convexa** es una expresión de la forma:

$$x = \sum_{i=1}^k \lambda_i x_i, \quad \text{donde } \lambda_i \geq 0, \sum_{i=1}^k \lambda_i = 1.$$

- La *envolvente convexa* de un conjunto $A \subseteq X$, denotada $conv(A)$, es el conjunto de todas las combinaciones convexas finitas de puntos de A . De hecho, es el menor conjunto convexo que contiene a A . De manera formal, se tiene la siguiente proposición.

Proposición 1.5. *La envolvente convexa de un conjunto $A \subset \mathbb{R}^n$ consiste en todas las combinaciones convexas de sus elementos.*

1. Preliminares y fundamentos teóricos

Demostración. Sea C el conjunto de todas las combinaciones convexas de A . Claramente $C \subset \text{conv}(A)$. Para probar la inclusión recíproca, basta con mostrar que C es convexo (ya que obviamente $C \supset A$).

Si $x = \sum_{i \in I} \lambda_i a_i$ e $y = \sum_{j \in J} \mu_j b_j$ con $a_i, b_j \in A$ y $0 \leq \alpha \leq 1$, entonces

$$(1 - \alpha)x + \alpha y = \sum_{i \in I} (1 - \alpha)\lambda_i a_i + \sum_{j \in J} \alpha \mu_j b_j.$$

Como

$$\sum_{i \in I} (1 - \alpha)\lambda_i + \sum_{j \in J} \alpha \mu_j = (1 - \alpha) \sum_{i \in I} \lambda_i + \alpha \sum_{j \in J} \mu_j = (1 - \alpha) + \alpha = 1,$$

se sigue que $(1 - \alpha)x + \alpha y \in C$. Por tanto, C es convexo. \square

La proposición 1.5 es realmente interesante, pues muestra que todo punto $x \in \text{conv}(E)$ puede expresarse como una combinación convexa de un número finito de puntos de E . Este resultado fundamenta muchas otras propiedades en Análisis Funcional y Geometría Convexa.

1.5. Convexidad en \mathbb{R}^n

Los conjuntos convexos y las funciones convexas desempeñan un papel fundamental en numerosos problemas de optimización. El estudio de los conjuntos convexos en \mathbb{R}^n empieza por las nociones de segmentos de línea, rectas e hiperplanos en \mathbb{R}^n . Por ello, comenzaremos recordando brevemente la notación y uso de estos conceptos.

El segmento **cerrado** que une dos puntos, x_1 y x_2 se denota por

$$[x_1, x_2] := \{x \in \mathbb{R}^n : x = (1 - t)x_1 + tx_2, 0 \leq t \leq 1\}$$

El segmento **abierto** que une dos puntos, x_1 y x_2 se denota por

$$(x_1, x_2) := \{x \in \mathbb{R}^n : x = (1 - t)x_1 + tx_2, 0 < t < 1\}$$

El segmento semiabierto que incluye x_1 pero no x_2 se denotará por $[x_1, x_2)$ y se obtiene restringiendo t al intervalo semiabierto $0 \leq t < 1$.

De manera similar, el segmento de línea semiabierto que incluye x_2 pero no x_1 se denotará por $(x_1, x_2]$ y se obtiene restringiendo t al intervalo semiabierto $0 < t \leq 1$.

En concreto, en este trabajo nos centraremos en \mathbb{R}^n como espacio métrico con la función euclídea descrita en (1.1).

Denotaremos la *bola abierta* centrada en x y con radio $r > 0$ por $B(x, r)$, y se define como el conjunto de puntos cuya distancia a x es menor que r . Esto es,

$$B(x, r) = \{y \in \mathbb{R}^n : d(y, x) < r\}$$

La *bola cerrada* con centro x y radio $r > 0$ la denotaremos por $\overline{B(x, r)}$.

Ahora, partiendo de la definición de plano en \mathbb{R}^n , generalizamos y recordamos la definición de hiperplano, pues la necesitaremos más adelante.

Definición 1.6. Un *hiperplano* H_a^α en \mathbb{R}^n se define como el conjunto de puntos que cumplen la ecuación $\langle a, x \rangle = \alpha$. Esto es,

$$H_a^\alpha = \{x \in \mathbb{R}^n : \langle a, x \rangle = \alpha\}$$

Se dice que el vector $a \in \mathbb{R}^n$ es normal al hiperplano. También se suele denotar como $H = [f = \alpha]$, con $f(x) = \langle a, x \rangle$, al hiperplano de la definición anterior.

Proposición 1.7. Sea X un conjunto contenido en la mitad cerrada negativa determinada por el hiperplano H_a^α , es decir, $X \subset \{x \in \mathbb{R}^n : \langle a, x \rangle \leq \alpha\}$. Si se cumple $x \in \text{int}(X)$ entonces $\langle a, x \rangle < \alpha$.

Demostración. Intuitivamente, como x pertenece al interior de X , debe existir una vecindad alrededor del punto que esté completamente contenida en X . Esto implica que x no puede estar en la propia frontera del semiespacio, es decir, la desigualdad debe ser estricta. Formalicemos y habremos terminado la demostración.

Supongamos por contradicción que $\langle a, x \rangle = \alpha$. Esto quiere decir que x pertenece a la frontera del semiespacio definido por $\{x \in \mathbb{R}^n : \langle a, x \rangle \leq \alpha\}$ o, en otras palabras, que x pertenece al hiperplano H_a^α .

Por hipótesis X está contenido en el mencionado semiespacio cerrado, y por tanto los puntos de X no pueden estar contenidos en el *lado* positivo del hiperplano $\{x \in \mathbb{R}^n : \langle a, x \rangle > \alpha\}$. Al pertenecer x al hiperplano, cualquier vecindad alrededor de x contiene puntos en ambos semiespacios. Es decir, si tomamos $B(x, r)$, la bola abierta centrada en x y radio $r > 0$, existe un punto $y \in B(x, r)$ tal que $\langle a, y \rangle > \alpha$.

Sin embargo, esto contradice el hecho de que x sea un punto interior en X . Si $x \in \text{int}(X) \implies B(x, r) \subseteq X \implies \forall y \in B(x, r) \text{ se tiene que } \langle a, y \rangle \leq \alpha$, lo cual hemos visto que no es cierto. Esto implica que x no puede pertenecer al hiperplano y por tanto $\langle a, x \rangle < \alpha$. \square

Antes de continuar con el desarrollo del trabajo, se presentan y recuerdan algunos enunciados y teoremas fundamentales que serán utilizados posteriormente. Cabe señalar que estos resultados ya han sido estudiados con profundidad durante el grado, por lo que aquí se exponen únicamente a modo de recordatorio, sin profundizar en sus demostraciones.

Se recuerdan brevemente los siguientes conceptos necesarios para el siguiente teorema. Un conjunto en \mathbb{R}^n es:

- **Cerrado:** si contiene todos sus puntos límite, es decir, si una sucesión de puntos dentro del conjunto converge, su límite también pertenece al conjunto.
- **Compacto:** si es cerrado y acotado. Es decir, no se extiende indefinidamente y contiene todos sus puntos límite. En \mathbb{R}^n , un conjunto es compacto si toda sucesión de puntos del conjunto tiene una subsucesión convergente cuyo límite también pertenece al conjunto (teorema de Bolzano-Weierstrass).

1. Preliminares y fundamentos teóricos

Teorema 1.8 (Heine-Borel). *Sea $K \subset \mathbb{R}^n$. Entonces K es compacto si y solo si es cerrado y acotado.*

Introducimos ahora un lema importante para la demostración del teorema de Minkowski-Carathéodory. Recordamos las siguientes nociones del Análisis Funcional:

Un *funcional lineal* es una aplicación lineal $f : V \rightarrow \mathbb{R}$ desde un espacio normado V hacia el cuerpo de los escalares. Es decir, cumple:

$$f(\lambda x + \mu y) = \lambda f(x) + \mu f(y) \quad \forall x, y \in V, \lambda, \mu \in \mathbb{R}.$$

El conjunto de todos los funcionales lineales y continuos sobre V se denota V^* y se llama *espacio dual*.

Lema 1.9. *Sea $E \subset A$ un subconjunto extremal. Entonces, todo punto extremo de E es un punto extremo de A .*

Demostración. La demostración es puramente rutinaria. Sea $x \in E$ un punto extremo. Pongamos que

$$x = \lambda y + (1 - \lambda)z, \quad \text{con } y, z \in A, \lambda \in (0, 1).$$

Ya que E es extremal, por definición se tiene que:

$$x \in E \subset A, x = \lambda y + (1 - \lambda)z \implies y, z \in E,$$

y, como es extremo en E , necesariamente se tiene que $x = y = z$, lo que implica que x es un punto extremo de A . \square

Lema 1.10. *Sea A un subconjunto de un espacio normado V , y sea $f \in V^*$. Supongamos que la función f alcanza su máximo en A . Entonces el conjunto*

$$E = \{x \in A : f(x) = \max f(A)\}$$

es un subconjunto extremal de A .

Demostración. Supongamos que $x \in E$, es decir, $f(x) = \max f(A)$, y que existen $y, z \in A$ y $\lambda \in (0, 1)$ tal que

$$x = \lambda y + (1 - \lambda)z.$$

Como $f \in V^*$, se tiene que

$$f(x) = f(\lambda y + (1 - \lambda)z) = \lambda f(y) + (1 - \lambda)f(z) = \max f(A).$$

Para que una combinación convexa de $f(y)$ y $f(z)$ dé como resultado el máximo, tiene que ocurrir que ambos sean iguales al máximo. Esto es, $f(y) = f(z) = \max f(A)$. Esto implica que $y, z \in E$, y por tanto E es un subconjunto extremal de A . \square

Se enuncian ahora dos teoremas de especial relevancia en el contexto del Análisis Funcional y la teoría de conjuntos convexos, recordando previamente la definición de espacio de Hilbert (la generalización del espacio euclídeo).

Un espacio de Hilbert es un espacio vectorial H (real o complejo), dotado de un producto interno $\langle \cdot, \cdot \rangle$, que además es completo¹ respecto a la norma inducida por este producto:

$$\|x\| = \sqrt{\langle x, x \rangle}.$$

Un ejemplo clásico es \mathbb{R}^n , con el producto interno usual $\langle x, y \rangle = x^T y$.

Teorema 1.11 (Teorema de Represenación de Riesz). *Sea H un espacio de Hilbert y $f \in H^*$ un funcional lineal y continuo. Entonces, existe un único vector $q \in H$ tal que:*

$$f(x) = \langle q, x \rangle \quad \forall x \in H.$$

Observación 1.12. En el caso particular de $H = \mathbb{R}^n$, esto se traduce como:

$$f(x) = q^T x.$$

Por último, se recuerda el importante teorema de Hahn-Banach, en su segunda forma geométrica.

Teorema 1.13 (Hahn-Banach, segunda forma geométrica). *Sea E un espacio vectorial normado y sean $A, B \subset E$ dos conjuntos no vacíos y convexos de tal manera que $A \cap B = \emptyset$. Sea A cerrado y B compacto. Entonces existe un hiperplano cerrado que separa estrictamente A y B .*

Se presentan ahora algunos resultados importantes sobre la solubilidad de un sistema finito de inecuaciones lineales. Algunas de las condiciones necesarias para muchos problemas de optimización se deducen de lo siguiente. Se discutirán y recordarán estos resultados en las siguientes secciones, cuando sea necesario.

Lema 1.14. *Sea C un cono, es decir, el conjunto*

$$C = \left\{ \sum_{i=1}^n \alpha_i a_i : \alpha_i \geq 0 \right\},$$

donde $\{a_1, \dots, a_n\}$, $a_i \in \mathbb{R}^m$, $n \in \mathbb{N}$, es el conjunto de vectores columna que componen una matriz A de orden $m \times n$. Este conjunto se puede reescribir como:

$$C = \{s \in \mathbb{R}^m : s = A\alpha, \alpha \in \mathbb{R}^n, \alpha_i \geq 0\}$$

El conjunto C (denominado como convexo) es cerrado y convexo.

Demostración. Sean $a, b \in C$, y $\lambda \in [0, 1]$, notemos que

$$\lambda a + (1 - \lambda)b = \lambda A\alpha_x + (1 - \lambda)A\alpha_y = A\alpha_{\bar{x}} + A\alpha_{\bar{y}} \in C,$$

¹En análisis matemático, un espacio métrico (X, d) se dice que es *completo* si toda sucesión de Cauchy contenida en X converge a un elemento de X , es decir, existe un elemento del espacio que es el límite de la sucesión.

1. Preliminares y fundamentos teóricos

donde hemos notado $\alpha_{\tilde{x}} = \lambda\alpha_x \geq 0$ y $\alpha_{\tilde{y}} = (1 - \lambda)\alpha_y \geq 0$, pues $\lambda \in [0, 1]$.

Probemos entonces que C es cerrado. En otras palabras, si $\{c_n\}_{n \in \mathbb{N}} \subseteq C$ es una sucesión convergente a c_* , vamos a probar que $c_* \in C$. Para ello, tomando la sucesión

$$c_i = A\alpha_{c_i}, \quad \alpha_{c_i} \in \mathbb{R}^n, \quad \alpha_{c_i} \geq 0, \quad i \in \mathbb{N}$$

La sucesión $\{c_n\}_{n \in \mathbb{N}} \subseteq C$, se puede escribir en términos de los coeficientes, los cuales pertenecen al conjunto

$$W := \{\alpha \in \mathbb{R}^n : \alpha \geq 0\}.$$

Como la sucesión $\{c_n\}$ es convergente en \mathbb{R}^m , sus componentes forman sucesiones convergentes. Debido a que la transformación $A\alpha_{c_n} = c_n$ es lineal, la convergencia de $\{c_n\}$ nos sugiere que la sucesión $\{\alpha_{c_n}\}$ también debe ser convergente en \mathbb{R}^n .

Pero ocurre que $\{\alpha_{c_n}\} \subseteq W$, el cual es un subconjunto cerrado y acotado de \mathbb{R}^n . En \mathbb{R}^n , cualquier sucesión acotada tiene una subsucesión convergente por el *Teorema de Bolzano-Weierstrass*.

Así, existe una subsucesión $\{\alpha_{c_{n_k}}\}$ tal que:

$$\alpha_{c_{n_k}} \rightarrow \alpha_* \quad \text{cuando } k \rightarrow \infty$$

Y como la convergencia se da en W , se preserva la condición $\alpha_* \geq 0$, es decir, el límite sigue estando en W . Hemos probado que $\{\alpha_{c_n}\}$ converge a $\alpha_* \in W$. Por último, tomamos la ecuación $c_n = A\alpha_{c_n}$ y pasamos al límite en ambos lados:

$$\lim_{n \rightarrow \infty} c_n = \lim_{n \rightarrow \infty} A\alpha_{c_n}$$

Por continuidad y linealidad, podemos poner la igualdad como:

$$c_* = A \left(\lim_{n \rightarrow \infty} \alpha_{c_n} \right) = A\alpha_*.$$

Dado que hemos demostrado que $\alpha_* \geq 0$, esto significa que c_* se puede escribir como una combinación de las columnas de A con coeficientes no negativos. Es decir, $c^* \in C$.

□

Todos los enunciados y teoremas recordados hasta ahora, nos servirán ahora para la demostración del siguiente teorema. El *lema de Farkas* es un resultado fundamental en **PL** y análisis de solubilidad de sistemas lineales. Establece una condición alternativa: dado un sistema lineal, o bien existe una solución factible, o bien existe otra combinación lineal que demuestra que no puede haberla, pero no ambas.

Teorema 1.15 (Lema de Farkas). Sean $A \in \mathbb{R}^{m \times n}$ y $b \in \mathbb{R}^m$. Entonces, exactamente uno de los siguientes conjuntos debe ser vacío:

$$1. \quad U_1 := \{x \in \mathbb{R}^n : Ax = b, x \geq 0\}$$

$$2. U_2 := \{y \in \mathbb{R}^m : A^T y \geq 0, b^T y < 0\}$$

Observación 1.16. Nótese que afirmar lo anterior es equivalente a decir que uno y solo uno de los sistemas

$$I : Ax \leq 0, \quad \langle b, x \rangle > 0, \quad x \in \mathbb{R}^n,$$

$$II : A^T y = b, \quad y \geq 0, \quad y \in \mathbb{R}^m.$$

tiene solución. O bien I tiene solución o bien la tiene II , pero nunca ambos.

Demostración. Primeramente, vamos a probar que los conjuntos U_1 y U_2 no pueden ser no vacíos al mismo tiempo. Supongamos por contradicción que no lo son y existen $x \in \mathbb{R}^n$ e $y \in \mathbb{R}^m$ tal que

$$\begin{aligned} Ax &= b, \quad x \geq 0, \\ A^T y &\geq 0, \quad b^T y < 0. \end{aligned} \tag{1.6}$$

Notemos ahora que usando la ecuación (1.6):

$$y^T Ax = y^T (Ax) = y^T b < 0$$

Sin embargo, como $x \geq 0$ y $A^T y \geq 0$, se tiene que

$$y^T Ax \geq 0.$$

Hemos visto entonces que $y^T Ax < 0$ y que $y^T Ax \geq 0$. Llegamos a contradicción y por tanto los conjuntos U_1 y U_2 no pueden ser no vacíos de manera simultánea.

- (i) Supongamos que U_1 es vacío y probemos que U_2 no lo es. Buscamos entonces encontrar un $y \in \mathbb{R}^m$ de tal manera que

$$A^T y \geq 0, \quad b^T y < 0.$$

Consideremos $\{b\}$ como un conjunto compacto y convexo, trivialmente. Consideraremos también el cono C definido por:

$$C = \{s \in \mathbb{R}^m : s = A\alpha, \alpha \in \mathbb{R}^n, \alpha \geq 0\}$$

Por el Lema 1.14 sabemos que C es cerrado y convexo. Además, se tiene que $\{b\}$ y C son disjuntos por hipótesis ($U_1 = \emptyset$). Por el Teorema de Hahn-Banach (2^a forma geométrica) 1.13, existe un hiperplano de separación $H = [f = \gamma]$, donde f es un funcional lineal y continuo por ser dimensión finita, cerrado que separa estrictamente a los conjuntos $\{b\}$ y C . Esto es, existe $\gamma \in \mathbb{R}$ tal que

$$f(b) < \gamma < f(s), \forall s \in C,$$

que reformulando queda como

$$f(b) < \gamma < f(s), \forall \alpha \in \mathbb{R}^n, \alpha_i \geq 0. \quad (1.7)$$

Así, como $0 \in C$ (tomando $\alpha = 0$), entonces

$$f(0) < 0. \quad (1.8)$$

Luego, por el teorema de representación de Riesz, 1.11, existe un $q \in \mathbb{R}^m$, tal que

$$f(s) = q^T s \quad \forall s \in \mathbb{R}^m.$$

De donde, usando lo anterior en (1.7), se sigue que

$$q^T b < \gamma < q^T A\alpha \quad \forall \alpha \in \mathbb{R}^n.$$

De esta manera, en la ecuación (1.8) se tiene que

$$q^T b < 0.$$

Y, por otro lado, tenemos que

$$\begin{aligned} q^T A\alpha > 0 &\implies (A\alpha)^T q > 0 \\ &\implies \alpha^T A^T q > 0 \\ &\implies A^T q \geq 0 \quad \text{pues la desigualdad anterior} \\ &\quad \text{se tiene para todo } \alpha \in \mathbb{R}^n, \text{ con } \alpha_i \geq 0. \end{aligned}$$

Así, se tiene que $q \in U_2$ y por lo tanto U_2 no es vacío, como se quería.

(ii) Supongamos que U_2 es vacío, vamos a probar que U_1 es no vacío.

Buscamos demostrar entonces que el sistema

$$Ax = b, x \geq 0,$$

tiene solución. El hecho de que II no tenga solución equivale a decir que $b \notin C$, el conjunto definido previamente. Sabíamos que es cerrado y convexo. Entonces, por el Teorema de Hahn-Banach 1.13, de nuevo, podemos encontrar un hiperplano $H_{x_0}^\alpha$ que separa estrictamente b y C . Por tanto, $x_0 \neq 0$ y

$$\langle x_0, b \rangle > \alpha > \langle x_0, z \rangle, \forall z \in C.$$

Como $0 \in C$, tenemos que $\alpha > 0$. Es decir, $\langle x_0, b \rangle > 0$. Ahora, para todo $z \in C$ se tiene que

$$\langle x_0, z \rangle = \langle x_0, A^T s \rangle = s^T A x_0 < \alpha,$$

dándose esta última igualdad $\forall s \in \mathbb{R}^m$, con todas las componentes de s mayores o iguales que 0. Veamos que esta última igualdad implica que

$$Ax_0 \leq 0.$$

Si esto último no fuese verdad, existiría, al menos, una componente de Ax_0 (pongamos la j -ésima) que sería positiva. Sea ξ_j dicha componente y sea $y_j = \lambda e_j = \lambda(0, \dots, 0, 1, 0, \dots, 0)$, con $\lambda > 0$. Entonces $\langle y_j, Ax_0 \rangle = \lambda \xi_j < \alpha$. Como se tiene que $\xi_j > 0$, si tomamos un λ suficientemente grande, llegamos a contradicción. Por tanto, $Ax_0 \leq 0$ y $\langle x_0, b \rangle > 0$, lo que implica que x_0 es solución de I.

□

Para cerrar la sección, se recuerdan algunos resultados sobre separación de subconjuntos convexos.

Teorema 1.17. *Sea X un espacio normado sobre \mathbb{R} , A y B subconjuntos no vacíos convexos de X . Supongamos que A tiene interior no vacío y que $B \cap \text{int}(A) = \emptyset$. Entonces existen $f \in X^*$ y $\alpha \in \mathbb{R}$ tales que*

$$f(a) \leq \alpha \leq f(b), \quad \text{con } a \in A, b \in B.$$

Además se tiene

$$f(u) < \alpha, \quad \forall u \in \text{int}(A).$$

Si partimos del teorema anterior y tomamos A como un conjunto cerrado y $B = \{x_0\}$, donde x_0 es un punto de la frontera de A , obtenemos el siguiente corolario.

Corolario 1.18. *Sea X un espacio normado sobre \mathbb{R} y A un subconjunto convexo y cerrado de X , con interior distinto del vacío. Para cada punto x_0 de la frontera de A , existe un funcional $f_0 \in X^*$ tal que*

$$f_0(x_0) = \max\{f_0(x) : x \in A\}.$$

Si se toma $\alpha_0 = f_0(x_0)$, el hiperplano en X dado por

$$H_0 = \{x \in X : f_0(x) = \alpha_0\},$$

pasa por el punto x_0 y deja el conjunto A a un lado. Geométricamente, esta situación se describe diciendo que el funcional f_0 o el hiperplano H_0 **soportan** al conjunto A , en el punto x_0 . También se puede decir que x_0 es un punto de soporte de A .

1.6. El Teorema de Minkowski-Carathéodory

El **Teorema de Minkowski-Carathéodory** es un resultado fundamental en el estudio de los conjuntos convexos dentro del análisis y la geometría convexa. El teorema además nos servirá como motivación para el Teorema de Krein-Milman, de la parte de optimización convexa que se verá más adelante.

Empezaremos demostrando el Teorema de Carathéodory, en el que estableceremos una cota superior en la longitud de la combinación lineal presentada en la proposición 1.5. Este resultado proporciona una forma eficiente de representar cualquier punto en la envolvente convexa de un conjunto en términos de una cantidad finita y limitada de elementos del propio conjunto.

1. Preliminares y fundamentos teóricos

Teorema 1.19 (Carathéodory). *Sea A un conjunto contenido en un conjunto afín de dimensión k . Entonces, cualquier vector $x \in \text{conv}(A)$ puede representarse como una combinación convexa de $k+1$ o menos elementos de A .*

Demostración. Por la Proposición 1.5, para cualquier $x \in \text{conv}(A)$, existe un subconjunto finito de A , pongamos $a_1, \dots, a_m \in A$, tal que:

$$\sum_{i=1}^m \lambda_i a_i = x, \quad \sum_{i=1}^m \lambda_i = 1, \quad \lambda_i \geq 0.$$

Es decir, ya sabemos que existe tal combinación, pero aún no sabemos cuántos puntos se están usando, quizás más de $k+1$.

Hagamos un proceso iterativo sobre m y veamos que en un número finito de pasos habremos terminado.

1. Si $m \leq k+1$ hemos terminado.
2. Si $m > k+1$, hemos visto en el lema 1.3, que $a_2 - a_1, \dots, a_m - a_1$ son afínmente dependientes.
3. Existen $\alpha_i \in \mathbb{R}$, alguno de ellos estrictamente positivo, tales que $\sum_{i=1}^k \alpha_i a_i = 0$, por el lema mencionado.
4. Como queremos encontrar una nueva combinación de $\lambda'_i = \lambda_i - \gamma \alpha_i \geq 0$, ajustaremos este parámetro γ construyendo una nueva combinación lineal con $m-1$ elementos de A . Expresamos

$$x = \sum_{i=1}^m (\lambda_i - \gamma \alpha_i) a_i,$$

donde

$$\gamma = \min\left\{\frac{\lambda_j}{\alpha_j} : \alpha_j > 0\right\} > 0,$$

siendo j el subíndice que recorre los α_i con signo positivo. Ese γ es el máximo valor permitido que garantiza que todos los nuevos λ'_i sean estrictamente positivos. Sin embargo, donde γ alcanza ese mínimo, pongamos que en el índice r , ocurre que

$$\lambda_r - \gamma \alpha_r = 0 \implies \lambda'_r = 0.$$

De esta manera, se reduce en uno el número de elementos de A que usamos en la combinación. Mientras $m-1 > k+1$, volvemos al paso 1. □

El teorema proporciona una caracterización fundamental de los puntos de las combinaciones convexas, mostrando que cualquier punto convexo puede expresarse como combinación convexa de a lo sumo $k+1$ puntos en un espacio de dimensión k . Esta propiedad no solo tiene interés teórico, sino que también es clave en el análisis y diseño de algoritmos de optimización convexa y PL. Veremos ahora que, en combinación con otro resultado importante, se puede obtener un teorema incluso más potente, el teorema de Minkowski-Caratheodory.

Teorema 1.20 (Minkowski-Caratheodory). *Sea K un subconjunto compacto y convexo de \mathbb{R}^n . Entonces, todo $x \in K$ se puede escribir como combinación convexa de, a lo sumo, $n+1$, puntos extremos de K . En particular, $K = \text{conv}(\text{ex}(K))$.*

Demostración. En caso de que $K \subseteq \mathbb{R}$, entonces K es un segmento y el resultado es claro. Hagamos entonces inducción sobre n . Suponemos que el resultado es cierto para $n - 1$ y vemos que es cierto para n .

Sea $K \subseteq \mathbb{R}^n$. Si la dimensión afín de K es menor que n , entonces existe un hiperplano H que contiene a K y aplicamos la hipótesis de inducción.

Podemos entonces suponer que la dimensión de K es directamente n . Esto implica que existen $a_1, \dots, a_n \in K$ tales que son afínmente independientes. Notamos que $\text{conv}(a_1, \dots, a_n) \subseteq K$, que también tiene dimensión afín n . Por tanto, si tomamos el baricentro de a_1, \dots, a_n , que es claramente una combinación convexa, es un punto interior de K . Es decir, $\text{int}(K) \neq \emptyset$.

El corolario 1.18, en concreto para dimensión finita, nos permite escoger un punto x de la frontera de K y encontrar un hiperplano H de \mathbb{R}^n tal que $x \in X \cap H$. Esta intersección es un conjunto compacto y convexo, y además cae sobre un espacio afín de dimensión $n - 1$, por ser un hiperplano. Luego la hipótesis de inducción se afirma que x se puede escribir como combinación convexa de n puntos extremos de $H \cap K$, y por tanto puntos extremos de K (consecuencia del Lema 1.9).

Por último, si x es un punto interior de K , se puede tomar un punto extremo distinto $y \in K$ y notamos que x está en el segmento $[y, z]$, para un punto z de la frontera de K (ver Figura 1.1). Se tiene entonces que $z = \sum_{i=1}^n \lambda_i e_i$ con $e_1, \dots, e_n \in \text{ex}(K)$, $\lambda_1, \dots, \lambda_n \in [0, 1]$ y $\sum_{i=1}^n \lambda_i = 1$. Es decir,

$$x = \alpha \left(\sum_{i=1}^n \lambda_i e_i \right) + (1 - \alpha)y = \sum_{i=1}^n (\alpha \lambda_i) e_i + (1 - \alpha)y,$$

para algún $\alpha \in]0, 1[$.

□

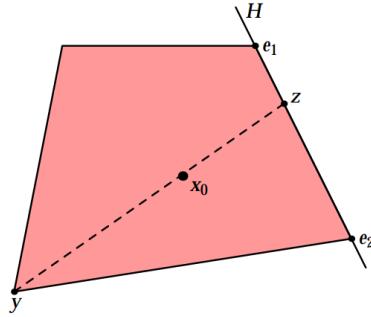


Figura 1.1.: Última parte de la demostración del teorema de Minkowski-Carathéodory. Fuente [Sui1].

Observación 1.21. Como el nombre indica, este teorema es resultado de la combinación de dos. Un teorema debido a H. Minkowski dice que cualquier subconjunto convexo y compacto de \mathbb{R}^n es la envolvente convexa de sus puntos extremos. Por otro lado, el teorema de Carathéodory 1.19 dice que, en dimensión n , cualquier punto de la envolvente convexa de un conjunto se puede escribir como combinación convexa de, a lo sumo, $n + 1$ puntos del conjunto. Combinando ambos resultados, obtenemos el teorema de Minkowski-Carathéodory.

1. Preliminares y fundamentos teóricos

Este resultado respalda la idea de que los puntos extremos (o vértices) de un conjunto convexo juegan un papel central en problemas de optimización, ya que las soluciones óptimas a menudo se alcanzan en estos puntos. Esto será especialmente relevante en el estudio de la **PL**, donde las soluciones óptimas se localizan en los vértices del poliedro factible (definido más adelante), y en la programación convexa, donde la estructura del conjunto convexo determina la complejidad del problema.

En la siguiente sección abordaremos con más detalle los conjuntos convexos, estableciendo las bases geométricas y analíticas necesarias para comprender los problemas de optimización en espacios convexos.

2. Problemas de Programación Lineal

En esta sección se busca explicar los fundamentos teóricos de los problemas de **PL**, empezando por las nociones básicas, así como la notación que se seguirá en el trabajo. De esta manera, podremos pasar posteriormente al estudio de los mismos y a la aplicación de algoritmos para su resolución.

Antes de comenzar, hemos de tener en cuenta que se habla de *convexidad* previamente en este trabajo, pues es estrictamente necesario cuando se van a presentar problemas de programación convexa. Sin embargo, no se debe olvidar que, además, *la programación convexa generaliza la PL de forma natural* al relajar la linealidad de las funciones objetivo y/o restricciones, pero siempre manteniendo la convexidad, lo cual preserva muchas propiedades útiles para la optimización.

Ir de lo particular a lo general, en este trabajo, parece tener sentido. Esto se verá sobre todo cuando en la experimentación se comprobará que el uso de técnicas más sencillas o básicas (como puede ser la **PL** y el método Simplex) es, en muchas ocasiones, suficiente para obtener buenos resultados [FTFC21].

En términos generales, un problema en la teoría de la optimización se enuncia como sigue:

Definición 2.1. Dado un subconjunto $A \subset \mathbb{R}^n$, y una función $f : A \rightarrow \mathbb{R}$, se pretende encontrar un punto a_* de manera que $f(a_*) \geq f(a)$ para todo $a \in A$. Determinar la existencia de dicho punto y, en caso de que exista, encontrarlo, lo consideraremos un **problema de optimización**.

Observación 2.2. Minimizar, en lugar de maximizar, f en A es equivalente a maximizar $-f$ en A . No hay pérdida de generalidad, pero en lo que sigue trabajaremos con la maximización de la función f .

En estos términos, podemos definir entonces lo que se conoce como un problema de **PL**:

Definición 2.3. Sea $X_0 = \mathbb{R}^n$ y sean las funciones $f : X_0 \rightarrow \mathbb{R}$ y $g : X_0 \rightarrow X_0$ lineales tales que

$$f(x) = \langle -b, x \rangle, \quad g(x) = \begin{pmatrix} Ax - c \\ -x \end{pmatrix}$$

con A una matriz $m \times n$, $m, n \in \mathbb{N}$ y c una constante real. Sea también

$$X = \{x \in X_0 : g(x) \leq 0\}.$$

El problema dado por la maximización de la función f (que denotaremos como **función objetivo**) sujeta a las restricciones establecidas en el conjunto X (que lo llamaremos **conjunto factible**), es lo que se conoce como **Problema de Programación Lineal (PPL)**.

Observación 2.4. Cuando X_0 es convexo, y las funciones f y g son convexas también, el problema se denota como **Problema de Programación Convexa (PPC)**.

2. Problemas de Programación Lineal

Un problema de **PL** puede presentarse en distintas formas según cómo se expresen sus restricciones y su función objetivo. Además, cualquier problema puede transformarse en la forma que se quiera, dependiendo de las necesidades de la situación. Se presentan ahora las formas más comunes, discutiendo en profundidad las que se usarán en este trabajo:

- (I) **Problema de Programación Lineal Estándar (PPL)**: Un problema de **PL** en *forma estándar PPLE* se formula como:

$$\text{Maximizar } f(x) = \langle b, x \rangle \text{ sujeto a } Ax \leq c, \text{ con } x \geq 0.$$

Donde $x \in \mathbb{R}^n$, A es una matriz $m \times n$, $b \neq 0$ un vector de \mathbb{R}^m y c es un vector de \mathbb{R}^n .

- (II) **Problema de Programación Lineal Normal (PPLN)**: Partiendo de la notación de la forma estándar, denotamos por

$$\tilde{A} = \begin{pmatrix} A \\ -I \end{pmatrix}, \tilde{c} = \begin{pmatrix} c \\ 0 \end{pmatrix}.$$

Al problema que viene determinado por:

$$\text{Maximizar } f(x) = \langle b, x \rangle \text{ sujeto a } \tilde{A}x \leq \tilde{c}$$

lo denotaremos por **PPLN**. I es la matriz identidad $n \times n$ y 0 es el vector nulo en \mathbb{R}^n . Reformulando de esta manera, se ha incorporado la restricción $x \geq 0$ que teníamos para los **PPLE**.

Antes de continuar con la presentación de los distintos problemas de **PL**, hemos de seguir con el siguiente paso lógico y esto es, dar las condiciones necesarias y suficientes para que un punto x_* sea solución del **PPLN**. Lo hacemos para esta notación para después extender el resultado a las siguientes presentaciones de los problemas.

El teorema expuesto a continuación nos da las bases teóricas y las condiciones suficientes que debe cumplir una solución óptima al problema estudiado. En la literatura, también se conoce como el teorema de las condiciones de *Karush-Kuhn-Tucker*, [GT12]. Veámoslo:

Teorema 2.5. *Sea $x_* \in X_0 = \mathbb{R}^n$ un vector solución para un **PPLN**. Entonces existe un $\lambda \neq 0$ en \mathbb{R}^m tal que el par (x_*, λ) cumple que:*

- (i) $Ax_* \leq c$,
- (ii) $A^\top \lambda = b$,
- (iii) $\lambda \geq 0$,
- (iv) $\langle Ax_* - c, \lambda \rangle = 0$,
- (v) $\langle b, x_* \rangle = \langle c, \lambda \rangle$.

En caso de existir un par que cumpla (i)-(v), se tiene que x_ es solución del problema **PPLN**.*

Demostración. Sea el **PPLN** dado por:

$$\text{Maximizar } \langle b, x \rangle \text{ sujeto a } Ax \leq c,$$

con $A \in \mathbb{R}^{m \times n}$, $b \in \mathbb{R}^n$, $c \in \mathbb{R}^m$.

(i) se cumple trivialmente, ya que $x_* \in X = \{x \in \mathbb{R}^n : Ax \leq c\}$.

Para (ii) y (iii), notamos que como x_* es solución óptima del problema, se tiene que $\langle b, x \rangle \leq \langle b, x_* \rangle$ para todo $x \in X$. Si denotamos $\alpha = \langle b, x_* \rangle$, ocurre que

$$X \subset \{x \in \mathbb{R}^n : \langle b, x \rangle \leq \alpha\}$$

Sean $E = \{i \in \mathbb{N} : \langle a_i, x_* \rangle = c_i\}$ el conjunto de índices correspondientes a las restricciones activas del sistema (restricciones de igualdad), e $I = \{i \in \mathbb{N} : \langle a_i, x_* \rangle < c_i\}$ el resto. Sean entonces A_E y A_I las submatrices de A formadas por las filas que corresponden a dichos índices.

Dado que x_* no es punto interior de X (por la Proposición 1.7), se tiene que $E \neq \emptyset$. Supongamos por contradicción que el siguiente sistema tiene solución:

$$A_E x \leq 0, \quad \langle b, x \rangle > 0.$$

Entonces, para $\alpha > 0$, el punto $x_* + \alpha x$ satisface

$$A_E(x_* + \alpha x) \leq 0,$$

lo que implica que aún pertenece al conjunto factible X , pero mejora el valor objetivo, contradiciendo la optimalidad de x_* . Por tanto, el sistema no tiene solución.

Por el Lema de Farkas, 1.15, existe $y \geq 0$ tal que $A_E^\top y = b$. Definimos

$$\lambda_i = \begin{cases} y_i & \text{si } i \in E, \\ 0 & \text{si } i \in I. \end{cases}$$

Entonces $\lambda \in \mathbb{R}^m$, $\lambda \geq 0$, y $A^\top \lambda = b$, con lo que se prueba (ii) y (iii).

Para probar (iv), notamos que si $\lambda_i > 0$ entonces $i \in E$, y por tanto $\langle a_i, x_* \rangle = c_i$. Así, tenemos

$$\langle Ax_*, \lambda \rangle = \sum_{i \in E} \lambda_i \langle a_i, x_* \rangle = \sum_{i \in E} \lambda_i c_i = \langle c, \lambda \rangle$$

lo cual prueba (iv).

Finalmente, para probar (v), usando (iv) y (ii), notamos que

$$\langle b, x_* \rangle = \langle A^\top \lambda, x_* \rangle = \langle \lambda, Ax_* \rangle = \langle \lambda, c \rangle$$

que prueba (v) y completa la demostración. □

Observación 2.6. Sea el problema de **PL** reformulado con X_0 tomado como un conjunto abierto

2. Problemas de Programación Lineal

arbitrario en lugar de \mathbb{R}^n . Es claro a partir de la demostración que si x_* es una solución a este problema, entonces se cumplen las condiciones necesarias. Recíprocamente, si existe un punto $x_* \in X_0$ en el cual se cumplen las condiciones necesarias, entonces x_* es factible y el máximo se alcanza en x_* .

(III) **Problema de Programación Lineal en forma Canónica (PPLC):** Las restricciones de desigualdad $Ax \leq c$ pueden convertirse en restricciones de igualdad introduciendo un vector no negativo, ξ , cuyas componentes se llaman *variables de holgura* y escribiendo

$$Ax + \xi = c, \quad \xi \geq 0.$$

Así, el problema de **PL** formulado para la *forma normal* puede reescribirse como

$$\text{Maximizar } \langle (b, 0), (x, \xi) \rangle \text{ sujeto a}$$

$$(A, I) \begin{pmatrix} x \\ \xi \end{pmatrix} = c, \quad x \geq 0, \quad \xi \geq 0.$$

Es decir, el **PPLN** puede escribirse como un **PPL** en *forma canónica* o **PPLC**:

$$\text{Maximizar } \langle b, x \rangle \text{ sujeto a}$$

$$Ax = c, \quad x \geq 0.$$

El método Símplex, que veremos en una próxima sección, se aplica a problemas en forma canónica. Como apunte, el **PPLC** puede escribirse como un **PPLN** al expresar la restricción de igualdad $Ax = c$ como dos restricciones de desigualdad, $Ax \leq c$ y $-Ax \leq -c$, y luego incorporando la restricción $x \geq 0$ en la matriz. Es decir, escribimos el **PPLC** como

$$\text{Maximizar } \langle -b, x \rangle \text{ sujeto a}$$

$$\begin{pmatrix} A \\ -A \\ -I \end{pmatrix} x \leq \begin{pmatrix} c \\ -c \\ 0 \end{pmatrix}.$$

De esta manera tan sencilla, todo lo probado y comentado acerca de los problemas en forma normal, aplica y generaliza para los problemas en forma canónica.

2.1. Extremos de un conjunto factible

En esta sección probaremos que si existe una solución para un **PPLC**, entonces existe un punto extremo del conjunto en el cual se alcanza el máximo de f . La motivación de los resultados propuestos a continuación es que, en la búsqueda de una solución para nuestro **PPL**, podemos “acotar” y reducir el análisis a estos puntos extremos del conjunto factible, para los cuales podemos obtener su forma analítica.

Para mantener la nomenclatura seguida en la mayoría de la literatura consultada [Bero3], para estos resultados formularemos el problema en forma PPLC como sigue:

$$\text{Maximizar } \langle b, x \rangle \text{ sujeto a}$$

$$Ax = c, \quad x \geq 0$$

Asumimos que $x, b \in \mathbb{R}^n$, que $c \in \mathbb{R}^m$, que A es una matriz de orden $m \times n$, que $m < n$ y que el rango de A es precisamente m .

Teorema 2.7. *Un punto del conjunto factible, llamémoslo x_0 , es un punto extremo del mismo si y solo si las columnas A_j de la matriz A que corresponden a las componentes positivas de x_0 son linealmente independientes.*

Demostración. Si x_0 no fuera un punto extremo del conjunto factible, entonces existirían dos puntos del conjunto factible distintos x_1 y x_2 , y un escalar λ tal que $0 < \lambda < 1$ y

$$x_0 = \lambda x_1 + (1 - \lambda) x_2. \quad (2.1)$$

Sea B la submatriz de A formada por las columnas de A que corresponden a componentes positivas de x , y sea N la submatriz de A formada por las columnas de A que corresponden a componentes iguales a cero de x . Para simplificar la notación, asumamos que B consiste en las primeras k columnas de A , y que N consiste en las $n - k$ columnas restantes. No hay pérdida de generalidad en hacer esta suposición, ya que se puede lograr mediante una reordenación de las coordenadas. Dado que las columnas de B son linealmente independientes, y el rango de A es m , se sigue que $k \leq m$. También se tiene que $n - k > 0$. Para $x \in \mathbb{R}^n$, definimos:

$$x_B = (x_1, \dots, x_k), \quad x_N = (x_{k+1}, \dots, x_n).$$

Entonces los puntos se reescriben como

$$x_0 = (x_{0B}, x_{0N}) = (x_{0B}, 0_N)$$

$$x_i = (x_{iB}, x_{iN}), \quad i = 1, 2,$$

donde x_0, x_1, x_2 son como en (2.1) y 0_N es el vector cero en \mathbb{R}^{n-k} .

Como $\lambda > 0$, y también $1 - \lambda > 0$, se sigue de (2.1) que una componente de x_0 es igual a cero si y solo si las componentes correspondientes de x_1 y x_2 son cero. Por lo tanto, $x_{0N} = x_{1N} = x_{2N} = 0_N$, y podemos volver a reescribir los puntos como

$$x_i = (x_{iB}, 0_N), \quad i = 0, 1, 2.$$

Dado que x_1 pertenece al conjunto factible,

$$c = Ax_1 = (B, N) \begin{pmatrix} x_{1B} \\ 0_N \end{pmatrix} = Bx_{1B}.$$

De forma similar, obtenemos que $c = Bx_{2B}$. Entonces,

$$B(x_{1B} - x_{2B}) = 0. \quad (2.2)$$

2. Problemas de Programación Lineal

Como $x_1 \neq x_2$ y $x_{1N} = x_{2N}$, se tiene que $x_{1B} - x_{2B} \neq 0$. Pero esto no puede ser, ya que las columnas de B son linealmente independientes. Esta contradicción demuestra que x_0 es un punto extremo.

Ahora supongamos que x_0 es un punto extremo del conjunto factible. Para simplificar la notación, supongamos que las primeras k componentes de x_0 son positivas y las componentes restantes son cero. Mostraremos que las primeras k columnas A_1, \dots, A_k de A son linealmente independientes. Si estas columnas no fueran linealmente independientes, entonces existirían constantes $\alpha_1, \dots, \alpha_k$, no todas cero, tales que

$$\alpha_1 A_1 + \dots + \alpha_k A_k = 0.$$

Sea el vector a definido por

$$a = (\alpha_1, \dots, \alpha_k, 0, \dots, 0).$$

Entonces, para cualquier número real ε ,

$$\varepsilon Aa = \varepsilon \sum_{j=1}^k \alpha_j A_j = 0. \quad (2.3)$$

Para $\varepsilon > 0$ suficientemente pequeño, los vectores $x_0 + \varepsilon a$ y $x_0 - \varepsilon a$ tienen sus primeras k componentes positivas y las restantes $n - k$ componentes iguales a cero.

Además,

$$A(x_0 \pm \varepsilon a) = Ax_0 \pm \varepsilon Aa = c,$$

donde la última igualdad se deduce de (2.3) y el hecho de que x_0 sea solución. Por tanto, para un $\varepsilon > 0$ los vectores $x_0 + \varepsilon a$ y $x_0 - \varepsilon a$ son distintos y ambos factibles, pertenecen al conjunto factible. Sin embargo, la relación

$$x_0 = \frac{1}{2}(x_0 + \varepsilon a) + \frac{1}{2}(x_0 - \varepsilon a)$$

contradice nuestra hipótesis, pues x_0 era un punto extremo. Por tanto, los vectores

$$A_1, A_2, \dots, A_k$$

son linealmente independientes. \square

Corolario 2.8. *Un punto extremo del conjunto factible tiene como máximo m componentes positivas.*

Corolario 2.9. *El número de puntos extremos de un conjunto factible es menor o igual a*

$$\binom{n}{m} = \frac{n!}{m!(n-m)!}$$

Demostración. Si x es un punto extremo del conjunto factible, entonces x tiene $k \leq m$ componentes positivas. Estas k componentes positivas corresponden a k columnas linealmente independientes de A . Afirmamos que si x' es otro punto extremo con las mismas componentes no nulas que x , entonces $x = x'$.

Para ver esto, sea B la submatriz de A correspondiente a las componentes no nulas de x y x' ; sea $x = (x_B, 0)$, y $x' = (x'_B, 0)$. Entonces, por el argumento usado para establecer (2.2),

obtenemos que

$$B(x_B - x'_B) = 0.$$

Dado que las columnas de B son linealmente independientes, se tiene que $x_B = x'_B$, y por tanto $x = x'$.

Si $k < m$, dado que el rango de A es m , podemos añadir $m - k$ columnas adicionales de A para obtener un conjunto de m columnas linealmente independientes asociadas con x . Si $k = m$, esto claramente se cumple. Por tanto, el número de puntos extremos no puede exceder el número de maneras en que se pueden seleccionar m columnas entre n columnas, esto es $\binom{n}{m}$. \square

Teorema 2.10. *Sea un PPLC, con solución x_0 . Entonces existe un punto extremo z del conjunto factible que también es solución al problema.*

Demuestra. Por el Teorema 2.7, si las columnas de A que corresponden a las componentes positivas de x_0 son linealmente independientes, entonces x_0 es un punto extremo del conjunto factible y tomamos $z = x_0$.

Supongamos ahora que las columnas de A que corresponden a las componentes positivas de x_0 son linealmente dependientes. Para simplificar la notación, suponemos que las primeras p componentes de x_0 son positivas y que las restantes $n - p$ componentes son cero.

Entonces, las primeras p columnas de A son linealmente dependientes y existen escalares $\gamma_1, \dots, \gamma_p$, no todos iguales a cero, tales que

$$\sum_{j=1}^p \gamma_j A_j = 0. \quad (2.4)$$

Para cada número real σ , definimos un vector $z(\sigma)$ como sigue:

$$z_j(\sigma) = \begin{cases} x_{0j} - \sigma \gamma_j, & j = 1, \dots, p, \\ 0, & j = p + 1, \dots, n. \end{cases} \quad (2.5)$$

Entonces, del hecho de que $x_{0j} = 0$ para $j = p + 1, \dots, n$, de la factibilidad de x_0 y de (2.4), obtenemos que:

$$Az(\sigma) = Ax_0 - \sigma \sum_{j=1}^p \gamma_j A_j = c.$$

Sea

$$\sigma_0 = \min \left\{ \frac{x_{0j}}{|\gamma_j|} : j = 1, \dots, p \text{ y } \gamma_j \neq 0 \right\}. \quad (2.6)$$

Se deduce de (2.5) que $z(\sigma) \geq 0$ siempre que $|\sigma| \leq \sigma_0$. Por lo tanto, $z(\sigma)$ es factible siempre que $|\sigma| \leq \sigma_0$. Así,

$$\langle b, x_0 \rangle \geq \langle b, z(\sigma) \rangle = \langle b, x_0 \rangle - \sigma \sum_{j=1}^p b_j \gamma_j$$

para todo σ tal que $|\sigma| \leq \sigma_0$. Entonces

$$\sum_{j=1}^p b_j \gamma_j = 0 \quad \text{y} \quad \langle b, x_0 \rangle = \langle b, z(\sigma) \rangle.$$

2. Problemas de Programación Lineal

Así, $z(\sigma)$ es una solución siempre que $|\sigma| \leq \sigma_0$. Sea el mínimo en (2.6) alcanzado en el índice r . Sea $\sigma_1 = x_{0r}/\gamma_r$. Entonces $\sigma_1 = \sigma_0$ si $\gamma_r > 0$ y $\sigma_1 = -\sigma_0$ si $\gamma_r < 0$. Por lo tanto, $z(\sigma_1)$ es factible y es una solución. Además, $z(\sigma_1)$ tiene menos de p componentes positivas.

Si las columnas correspondientes de A son linealmente independientes, entonces $z(\sigma_1)$ es un punto extremo. Si no son linealmente independientes, repetimos el proceso, tomando x_0 como $z(\sigma_1)$. En un número finito de iteraciones obtendremos una solución tal que las columnas correspondientes a las componentes positivas son linealmente independientes. Esta solución será un punto extremo. \square

Antes de introducir el método Simplex o abordar la programación convexa, resulta fundamental comprender el concepto de punto extremo en **PL**. En términos geométricos, el conjunto factible de un problema lineal —es decir, el conjunto de soluciones que satisfacen las restricciones— forma un poliedro convexo en el espacio \mathbb{R}^n . Los puntos extremos de este poliedro desempeñan un papel central en la teoría de optimización. Una propiedad clave de los **PPL** es que, si existe una solución óptima, esta se alcanza en al menos uno de estos puntos extremos (Teorema 2.10). Este resultado no solo simplifica el proceso de búsqueda de soluciones, sino que también permite diseñar algoritmos eficientes que eviten explorar todo el espacio factible.

3. El método Símplex

El método Símplex fue desarrollado por *George B. Dantzig* mientras trabajaba en el Pentágono, como asesor matemático para las Fuerzas Aéreas de los Estados Unidos. Su objetivo era mecanizar procesos de planificación logística y de asignación de recursos. La primera aplicación práctica del método se realizó en 1947, cuando se le encargó mecanizar un proceso de dietas puramente logístico usando calculadoras analógicas, y lo utilizó para resolver un problema que involucraba 9 ecuaciones y 77 variables. Este cálculo, realizado con calculadoras manuales, requirió aproximadamente 120 jornadas de trabajo, pero fue un antes y un después en el mundo de la *programación*¹ [Lab25], pues, casi sin quererlo, *Dantzig* había creado un método eficiente y general de resolución de problemas con restricciones.

Ha sido una de las herramientas más influyentes en la teoría de la optimización y en las aplicaciones industriales de la **PL**. La primera publicación formal del algoritmo se encuentra en el capítulo *Maximization of a Linear Function of Variables Subject to Linear Inequalities*, incluido en el libro [Dan51]. Esta obra sentó las bases del análisis estructurado de problemas lineales en múltiples disciplinas. Años más tarde, *Dantzig* publicó un artículo en el que relata con detalle el contexto histórico y los desafíos que enfrentó durante el desarrollo del método, proporcionando una perspectiva única sobre su creación y sus primeras aplicaciones prácticas [Dan90].

3.1. Motivación del algoritmo

Hemos visto en la sección anterior que si existe una solución al **PPLC**, entonces existe en un punto extremo del conjunto factible, o *feasible set* en la literatura, [Bero3]. En este capítulo presentaremos los fundamentos del método Símplex, que es un algoritmo para resolver problemas de **PL** de manera eficiente.

Para entender y poder seguir los pasos lógicos de este algoritmo, consideraremos el siguiente ejemplo, en \mathbb{R}^2 . Pensemos en el siguiente problema:

Ejemplo 3.1. Maximizar $f(x) = -x_1 + 2x_2$ con las siguientes restricciones:

$$\begin{aligned} -x_1 - x_2 &\leq -1, \\ -x_1 + x_2 &\leq 0, \\ x_1 + 2x_2 &\leq 4, \\ x_1 \geq 0, \quad x_2 \geq 0 \end{aligned}$$

Podemos visualizar el conjunto factible en el área sombreada en la Figura 3.1. Se muestran en gris las líneas de nivel para la función en los puntos $x = 0$ y $x = 1$. La dirección del

¹Referirse a la *programación* en esta época es hacer referencia a todo aquel proceso de planificación o estructuración de problemas propuesto para solucionar tareas, logísticas mayoritariamente, que requerían mucho tiempo. El término de *programación* hoy en día hace referencia a la parte computacional de la resolución de problemas, desde un punto de vista mucho más general.

3. El método Símplex

gradiente para f iría hacia arriba. Es decir, f crece en ese sentido. Por tanto, el máximo se alcanzaría en el punto $(\frac{4}{3}, \frac{4}{3})$.

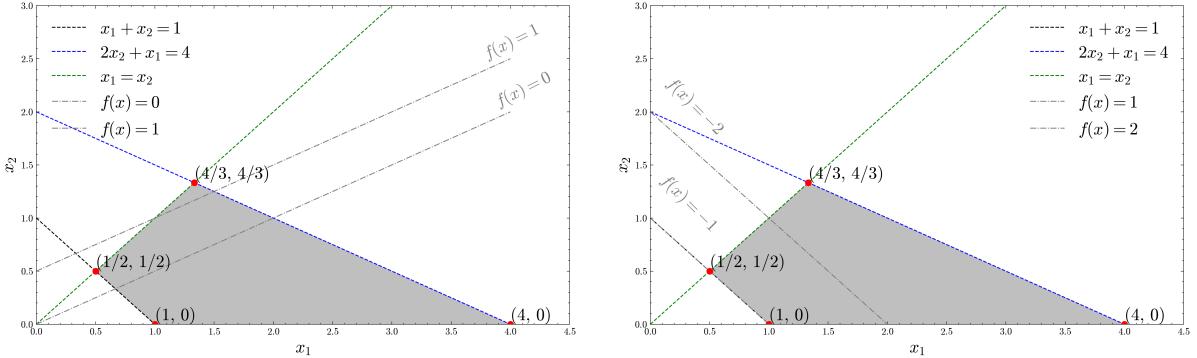


Figura 3.1.: Región factible en los ejemplos 3.1 (izquierda) y 3.2 (derecha). En el primero se muestran las líneas de nivel para $f(x) = 1$ y $f(x) = 0$, y en el segundo para $f(x) = -2$ y $f(x) = -1$.

Ejemplo 3.2. Este ejemplo difiere del Ejemplo 3.1 en que tomamos la función f como $f(x) = -x_1 - x_2$. Las curvas de nivel son líneas paralelas a la recta $x_1 + x_2 = 1$, un segmento de la cual forma parte de la frontera del conjunto factible. Dado que el gradiente de f es $(-1, -1)$, y esta es la dirección en la que f decrece, se sigue que f alcanza su máximo en todos los puntos del conjunto factible que caen sobre la recta $x_1 + x_2 = 1$. En particular, se alcanza el máximo en los puntos extremos $(\frac{1}{2}, \frac{1}{2})$ y $(1, 0)$. Ver Figura 3.1.

Ejemplo 3.3. Maximizar $f(x) = x_1 + x_2$ sujeto a:

$$\begin{aligned} -x_1 + x_2 &\leq 0, \\ x_1 - 2x_2 &\leq -2, \\ x_1 &\geq 0, \\ x_2 &\geq 0. \end{aligned}$$

El conjunto factible está representado por el área rayada en la Figura 3.2. La curva de nivel $f(x) = 6$ está indicada por una línea discontinua. Como el gradiente de f es $(1, 1)$, se deduce de la figura que el problema no tiene una solución en el conjunto factible, que no está acotado. Un conjunto no acotado, por otro lado, no implica que el problema no tenga solución. Si tomásemos $f(x) = -x_1 - x_2$ para el mismo ejemplo, tendríamos una solución en el punto extremo $(2, 2)$.

En los ejemplos que hemos considerado, el problema o bien no tenía solución, o bien tenía una solución alcanzada en un punto extremo del conjunto factible. Pero existe también una tercera posibilidad, que es que el conjunto factible sea vacío. En las siguientes secciones veremos que estas son las únicas posibilidades que se pueden encontrar en un problema de PL.

El método Símplex se aplica a problemas en forma canónica o PPLC. Para poner el problema del Ejemplo 3.1 en forma canónica, introducimos *variables de holgura* adicionales x_3, x_4, x_5 y escribimos el problema como:

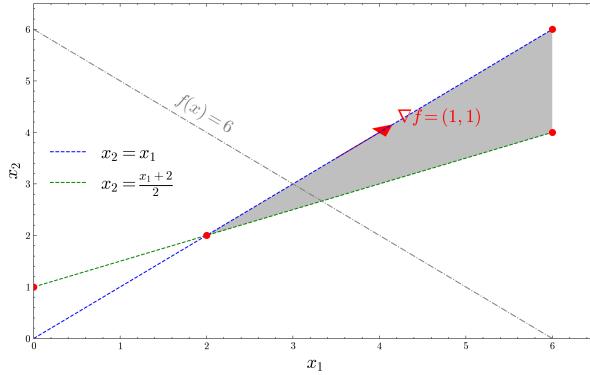


Figura 3.2.: Región factible sombreada, mostrándose la línea de nivel para $f(x) = 6$. El gradiente apunta hacia la derecha, pero el conjunto factible no está acotado en esa dirección, por lo que no hay solución para el problema.

$$\begin{aligned}
 & \text{Maximizar} && f(x) = -x_1 + 2x_2 \\
 & \text{sujeto a} && -x_1 - x_2 + x_3 = -1, \\
 & && -x_1 + x_2 + x_4 = 0, \\
 & && x_1 + 2x_2 + x_5 = 4.
 \end{aligned} \tag{3.1}$$

Dado que las variables x_3, x_4, x_5 no aparecen en la fórmula para f , el problema con el conjunto factible definido por (3.1) tiene su máximo alcanzado en

$$x_1 = \frac{4}{3}, \quad x_2 = \frac{4}{3}, \quad x_3 = \frac{5}{3}, \quad x_4 = 0, \quad x_5 = 0.$$

Se puede verificar fácilmente que este punto es un punto extremo del conjunto definido. Todo esto lo formalizaremos cuando expliquemos paso a paso el algoritmo.

Una vez ilustrados algunos ejemplos de problemas reales, pasamos al aspecto más importante que motiva el método: la **eficiencia**. En la última sección mostramos que si existe una solución para el problema de **PL** en forma canónica, entonces existe en un punto extremo del conjunto factible. También mostramos que hay como máximo $\frac{n!}{m!(n-m)!}$ de estos puntos. Para un sistema no muy grande, por ejemplo, de 5 ecuaciones de restricción y 25 variables, puede haber hasta

$$\binom{25}{5} = \frac{25!}{5!20!} = 53,130$$

puntos extremos. Por lo tanto, el esquema ingenuo de evaluar la función objetivo en todos los puntos extremos posibles y luego seleccionar aquel en el que se alcanza el mayor valor de la función objetivo no es práctico. Veremos ahora que justamente el método Símplex es un algoritmo usado para encontrar una solución pasando de un punto extremo del conjunto factible a otro, de manera que la función objetivo no disminuya.

3.2. Notación y definiciones previas

Ahora se introduce la notación y las definiciones que se utilizarán para la explicación detallada del algoritmo. Consideremos primero el sistema

$$Ax = c \quad (3.2)$$

que define el conjunto factible para un **PPLC**, un problema de **PL** en forma canónica. A es una matriz de orden $m \times n$ y de rango m . Sea B una submatriz de A de tamaño $m \times m$ obtenida al tomar m columnas linealmente independientes de A , y sea N la submatriz de A que consiste en las $n - m$ columnas restantes. Podemos escribir un vector x en \mathbb{R}^n como

$$x = (x_B, x_N),$$

donde x_B es el vector con aquellas componentes de x que corresponden con las columnas en B y análogamente para x_N con las columnas de N .

Definición 3.4. Las columnas de B se denominan una **base**. Las componentes de x_B se llaman **variables básicas**. Las componentes de x_N se llaman variables **no básicas** o **variables libres**.

Podemos escribir el sistema (3.2) como

$$(B, N) \begin{pmatrix} x_B \\ x_N \end{pmatrix} = c. \quad (3.3)$$

Cualquier vector de la forma (ξ_B, x_N) , donde x_N es arbitrario y

$$\xi_B = B^{-1}c - B^{-1}Nx_N,$$

será una solución de (3.3). Es por esto que las variables de x_N se llaman variables libres. Además, la única solución con $x_N = 0_N$ será

$$\xi = (\xi_B, 0_N), \quad \xi_B = B^{-1}c. \quad (3.4)$$

Definición 3.5. Existen tres tipos de soluciones:

- (I) Una solución de (3.2) de la forma (3.4) se llama una *solución básica*.
- (II) Si $\xi_B \geq 0_B$, entonces ξ se dice factible y se llama una *solución básica factible*.
- (III) Si algunas de las componentes de ξ_B son cero, entonces ξ se llama una *solución básica degenerada*.

Denotamos por ξ_B el valor de las variables básicas en una solución básica. Por ξ denotamos el valor de una solución de (3.2) y por ξ_N el valor de una variable x_N particular.

Nótense las dos observaciones siguientes, pues son cruciales para entender de manera íntegra y estructural el método Símplex.

Observación 3.6. En vista de lo anterior y del Teorema 2.7, se deduce que una solución básica factible es un punto extremo y viceversa.

Observación 3.7. Se deduce del Teorema 2.10 que, si existe una solución óptima, entonces existe una solución básica factible que es una solución óptima. Por lo tanto, solo necesitamos considerar soluciones básicas factibles en nuestra búsqueda de la solución óptima. Esta es la clave operativa para el método Simplex. En vez de revisar todas las infinitas soluciones posibles de $Ax = c, x \geq 0$, nos basta con revisar las soluciones básicas factibles, que son un número finito (aunque grande), y navegar entre ellas hasta encontrar la mejor.

Para aplicar el método Simplex, **reformulamos** el problema indicado en (3.2) introduciendo una variable escalar z y escribiendo el problema como:

$$\begin{aligned} & \text{Maximizar } z \\ & \text{sujeto a } Ax = c, \\ & \quad \langle b, x \rangle - z = 0, \\ & \quad x \geq 0. \end{aligned} \tag{3.5}$$

El método Simplex consta de dos fases. En la fase I, el método determina una solución básica factible o bien determina si el conjunto factible es vacío. En la fase II, el método parte de una solución básica factible y luego determina o bien que no existe una solución óptima o bien encuentra una solución óptima. En este último caso, el algoritmo lo hace moviéndose de una solución básica factible a otra de manera que el valor de z no decrece, siendo esta la principal ventaja que ofrece este algoritmo frente a una resolución del problema más directa.

3.3. Fase II del método Simplex

En la literatura, es común que se presente primero la Fase II del método Simplex antes de la Fase I. Esto se debe a que la Fase II constituye el núcleo del algoritmo y describe cómo se optimiza la función objetivo moviéndose de una solución básica factible a otra, mientras se garantiza que el valor de la función objetivo no disminuya. La Fase I, por otro lado, es un procedimiento auxiliar que se utiliza únicamente para encontrar una solución básica factible inicial, en caso de que no se disponga de una de manera directa.

La Fase II del método Simplex asume que ya se cuenta con una solución básica factible inicial. A partir de esta solución, el algoritmo procede iterativamente de la siguiente manera:

1. **Función objetivo:** Se expresa la función objetivo y las variables básicas en términos de las variables no básicas.
2. **Evaluación de la solución básica factible actual:** Se comprueba la optimalidad de la solución actual. Si resulta ser óptima, se termina el proceso.
3. **Determinación de la variable saliente:** Se identifica la variable básica que debe abandonar la base para mantener la factibilidad del sistema, de manera que no decrezca el valor de la función objetivo.
4. **Actualización de la base:** Se actualizan las variables básicas y no básicas, así como las matrices asociadas, para reflejar el cambio en la base, volviendo al paso 1 con las nuevas actualizaciones.

3. El método Símplex

Geométricamente, el proceso consiste en, partiendo de un punto extremo del conjunto factible, saltar a otro contiguo en el que la función objetivo aumenta su valor. Cuando se llega a un punto extremo de modo que en todos sus contiguos la función objetivo es menor, el proceso se detiene.

La razón por la que se presenta primero la Fase II es que esta contiene los fundamentos teóricos y prácticos del método Símplex. Una vez que se comprende cómo funciona la Fase II, es más sencillo entender la necesidad de la Fase I y cómo esta se utiliza para preparar el problema en caso de que no se disponga de una solución básica factible inicial. La Fase I, como se explicó anteriormente, introduce un problema auxiliar que permite encontrar dicha solución básica factible inicial o determinar que el conjunto factible es vacío.

Vamos ahora con la explicación en detalle y paso a paso de esta fase del algoritmo.

Paso 1. Expresar z y las variables básicas x_B en términos de las variables no básicas x_N . La ecuación en (3.5) de la sección anterior se puede escribir como:

$$\begin{pmatrix} B & N & 0_m \\ b_B^t & b_N^t & -1 \end{pmatrix} \begin{pmatrix} x_B \\ x_N \\ z \end{pmatrix} = \begin{pmatrix} c \\ 0 \end{pmatrix}. \quad (3.6)$$

Si multiplicamos la matriz aumentada correspondiente a las primeras m ecuaciones en (3.6) por B^{-1} a la izquierda, obtenemos una forma equivalente:

$$\begin{pmatrix} I_m & B^{-1}N & 0_m \\ b_B^t & b_N^t & -1 \end{pmatrix} \begin{pmatrix} x_B \\ x_N \\ z \end{pmatrix} = \begin{pmatrix} B^{-1}c \\ 0 \end{pmatrix}. \quad (3.7)$$

Si ahora tomamos $x_B = \xi_B$ y $x_N = 0_N$ en (3.7), obtenemos:

$$\xi_B = B^{-1}c. \quad (3.8)$$

A partir de (3.7) y (3.8) deducimos que:

$$x_B = \xi_B - B^{-1}N x_N. \quad (3.9)$$

Asimismo, de (3.7) y (3.9) obtenemos:

$$z = \langle b, \xi_B \rangle + \langle b_N - b_B^t B^{-1}N, x_N \rangle. \quad (3.10)$$

Denotamos:

$$\zeta = \langle b_B, \xi_B \rangle.$$

De esta manera, podemos escribir de nuevo z como:

$$z = \zeta + \langle b_N - b_B^t B^{-1}N, x_N \rangle. \quad (3.11)$$

Las ecuaciones (3.9) y (3.11) expresan x_B y z en términos de x_N .

Definimos el vector:

$$d_N = b_N - b_B^t B^{-1} N = (d_{j_{m+1}}, \dots, d_{j_n}), \quad (3.12)$$

donde j_{m+1}, \dots, j_n son los índices de las componentes de x que forman x_N . Entonces, podemos reescribir (3.11) como:

$$z = \zeta + \langle d_N, x_N \rangle = \zeta + \sum_{p=m+1}^n d_{j_p} x_{j_p}.$$

Observación 3.8. Nótese que, por la ecuación (3.8), podemos escribir la parte de la derecha de la igualdad en (3.7) como

$$\begin{pmatrix} B^{-1} c \\ 0 \end{pmatrix} = \begin{pmatrix} \xi \\ 0 \end{pmatrix}$$

El método tableau

El cálculo más complejo en (3.9) y (3.11) es la inversión de B . Los demás cálculos involucran operaciones matriciales y multiplicaciones de vectores y matrices más directas. Presentamos ahora un segundo método alternativo para expresar x_B y z en términos de x_N .

Desde el punto de vista computacional, la implementación de este método es esencialmente la misma que la del método anterior. Sin embargo, nuestro propósito aquí es didáctico. Para problemas de pequeña escala, resolver manualmente con este método es ventajoso y será el que usemos en el ejemplo que se muestra más adelante en el documento. El **método tableau** es una técnica que registra sistemáticamente la secuencia de pasos y cálculos que describiremos.

Comenzamos nuevamente con la matriz aumentada de (3.6). Aplicando una secuencia de operaciones elementales a las primeras m filas de la matriz, se transforma en una forma equivalente:

$$\begin{pmatrix} I_m & D & 0_m & c' \\ b_B^t & b_N^t & -1 & 0 \end{pmatrix}. \quad (3.13)$$

Cada una de estas operaciones se puede representar como una multiplicación por la izquierda de una matriz elemental E_i . Dado que esta secuencia de operaciones transforma B en la identidad I , el producto de estas matrices elementales es B^{-1} , como se demostró en la sección anterior. Así, la matriz (3.13) también se puede obtener multiplicando las primeras m filas de la matriz aumentada en (3.6) por B^{-1} . Por lo tanto, tenemos:

$$D = B^{-1} N, \quad c' = B^{-1} c.$$

Por lo tanto, (3.6) es equivalente al sistema con la matriz aumentada:

$$\begin{pmatrix} I_m & B^{-1} N & 0_m & c' \\ b_B^t & b_N^t & -1 & 0 \end{pmatrix}. \quad (3.14)$$

Dado que $c' = B^{-1} c$, obtenemos nuevamente (3.8) y (3.9). Supongamos ahora que las columnas de B son las columnas j_1, \dots, j_m de la matriz A . Si multiplicamos sucesivamente la fila 1 de (3.14) por $-b_{j_1}$ y sumamos el resultado a la última fila, luego multiplicamos la fila 2 por $-b_{j_2}$ y sumamos a la última fila, y así sucesivamente, transformamos (3.14) en:

3. El método Símplex

$$\begin{pmatrix} I_m & B^{-1}N & 0_m & c' \\ 0_B & b_N^t - b_B^t B^{-1}N & -1 & -\langle b_B, c' \rangle \end{pmatrix}$$

De la última fila de esta matriz y usando la relación $c' = B^{-1}c$, obtenemos inmediatamente:

$$z = \langle b_B, B^{-1}c \rangle + \langle b_N^t - b_B^t B^{-1}N, x_N \rangle. \quad (3.15)$$

Esto último se debe a que en la segunda fila se puede leer directamente la expresión de z , que corresponde a la función objetivo evaluada con respecto a las variables no básicas x_N . El primero de los sumandos (correspondiente a la última columna, segunda fila) lo podemos ver como un término constante resultado de evaluar la función objetivo en la solución básica actual, mientras que el segundo de los sumandos es dependiente de las variables no básicas.

Continuamos con el segundo paso de la fase II del método.

Paso 2. Comprobar si la solución básica factible actual es óptima es el siguiente paso lógico. Si cada componente del vector d_N definido en (3.12) es no positiva (es decir, ≤ 0), entonces la solución básica factible actual es óptima y el proceso termina.

Para ver por qué esto es cierto, recordemos que, según la Observación 3.6, solo necesitamos considerar soluciones básicas factibles en la búsqueda de la solución óptima. La solución básica factible actual es la única solución básica factible con $x_N = 0_N$. Por lo tanto, cualquier otra solución básica factible tendrá al menos una de las componentes de x_N positiva. Pero dado que $d_N \leq 0$, se deduce de (3.15) que el valor de la función objetivo disminuirá o permanecerá sin cambios. Por lo tanto, la solución básica factible actual es óptima.

Paso 3. Si la solución básica factible actual no es óptima, se debe o bien determinar una nueva base y la correspondiente solución básica factible que no disminuya el valor actual de z , o bien determinar que z es no acotado y, por lo tanto, concluir que el problema no tiene solución. Dividiremos este paso en dos pasos intermedios:

- (i) Se escoge una nueva variable no básica que va a ser la seleccionada para entrar en la base.

Supongamos que d_N es como en (3.12). Dado que la solución básica factible actual no es óptima, al menos una de las componentes de d_N es positiva. Se elige la componente de d_N que sea la mayor. Si hay más de una, se elige la de menor índice. Supongamos que la componente elegida corresponde a la q -ésima componente de $x = (x_1, \dots, x_n)$. La variable x_q entrará en la base. La columna de A correspondiente a x_q es A_q .

- (ii) Determinar si el problema no tiene solución o elegir la variable básica que saldrá de la base.

Sea v_q la columna de $B^{-1}N$ correspondiente a x_q , es decir, $v_q = B^{-1}A_q$. Definimos:

$$x_B = (x_{i_1}, \dots, x_{i_m}), \quad \xi_B = (\xi_{i_1}, \dots, \xi_{i_m}). \quad (3.16)$$

Sea $x_q = t$, donde $t \geq 0$, y las demás componentes de x_N permanecen en cero. Para que el vector resultante sea una solución factible, x_B debe cumplir las condiciones de (3.9) y ser no negativa, lo que nos da la ecuación:

$$x_B = \xi_B - tv_q. \quad (3.17)$$

La restricción $x_B \geq 0$ impone:

$$tv_{rq} \leq \xi_{i_r}, \quad r = 1, \dots, m. \quad (3.18)$$

De (3.10) obtenemos:

$$z = \zeta + d_q t. \quad (3.19)$$

Dado que ζ es factible, se cumple $\xi_B \geq 0$, y en particular $\xi_{i_r} \geq 0$ para todo $r = 1, \dots, m$. Si $v_{rq} \leq 0$ para todo r , entonces (3.17) y (3.18) se satisfacen para cualquier $t \geq 0$. Como además $d_q > 0$, se concluye que z no está acotado superiormente y el problema no tiene solución.

Si no todos los v_{rq} son negativos o cero, se define:

$$t^* = \min \left\{ \frac{\xi_{i_r}}{v_{rq}} : r = 1, \dots, m, \text{ y } v_{rq} > 0 \right\}. \quad (3.20)$$

Sea ρ el menor índice r en el que se alcanza este mínimo. Se define $x_q = t^*$ y se actualizan las variables:

$$\begin{aligned} \xi'_{i_r} &= \xi_{i_r} - t^* v_{rq}, \quad i_r \neq i_\rho, \\ \xi'_q &= t^*, \\ \xi'_{i_p} &= 0, \\ \xi'_i &= 0, \quad \text{para el resto de índices.} \end{aligned} \quad (3.21)$$

Sea ξ' el vector cuyas coordenadas están dadas por (3.21). Así, tenemos:

$$\xi' = (\xi'_{i_1}, \dots, \xi'_{i_{\rho-1}}, 0, \xi'_{i_{\rho+1}}, \dots, \xi'_{i_m}, 0, \dots, 0, t^*, 0, \dots, 0),$$

donde t^* es la q -ésima coordenada de ξ' . Dado que ζ fue construido como un vector factible, se deduce que ξ' también lo es. Podemos escribir ξ' como:

$$\xi'_B = (\xi'_{i_1}, \dots, \xi'_{i_{\rho-1}}, t^*, \xi'_{i_{\rho+1}}, \dots, \xi'_{i_m}), \quad \xi'_N = 0.$$

Mostraremos que las variables de coordenadas

$$x_{B'} = (x_{i_1}, \dots, x_{i_{\rho-1}}, x_q, x_{i_{\rho+1}}, \dots, x_{i_m}) \quad (3.22)$$

forman una base. Como $x_{B'}$ se obtiene de x_B al reemplazar x_{i_ρ} por x_q , esto también justifica que x_q entra en la base y x_{i_ρ} la abandona.

De (3.16), sabemos que la matriz I_m en (3.17) y (3.22) es:

3. El método Símplex

$$(e_{i_1} \dots e_{i_{\rho-1}} e_{i_\rho} e_{i_{\rho+1}} \dots e_m),$$

donde e_i es el vector con i -ésima componente igual a 1 y el resto igual a 0. Definimos:

$$B' = (e_{i_1} \dots e_{i_{\rho-1}} v_q e_{i_{\rho+1}} \dots e_{i_m})$$

Es decir, B' es la matriz obtenida de I_m al reemplazar la columna i_ρ por v_q . Sea N' la matriz obtenida de $B^{-1}N$ reemplazando la columna v_q por e_{i_ρ} .

Definimos $x_{B'}$ como en (3.22) y $x_{N'}$ como el vector obtenido de x_N al intercambiar x_q por x_{i_ρ} . Sea d_q la q -ésima coordenada del vector d_N definido en (3.12) y sea $d_{N'}$ el vector obtenido de d_N al reemplazar la q -ésima coordenada por 0. Tenemos que el sistema de ecuaciones

$$\begin{pmatrix} e_1 & \dots & e_{i_{\rho-1}} & v_q & e_{i_{\rho+1}} & \dots & e_{i_m} & N' & 0_m \\ 0 & \dots & 0 & d_q & 0 & \dots & 0 & d_{N'} & -1 \end{pmatrix} \begin{pmatrix} x_{B'} \\ x_{N'} \\ z \end{pmatrix} = \begin{pmatrix} B^{-1}c \\ -\zeta \end{pmatrix} \quad (3.23)$$

es equivalente al sistema cuya matriz aumentada es (3.17), ya que (3.23) se obtiene a partir de (3.17) con una reordenación de las variables.

Afirmamos que la matriz B' es una base y que las variables $x_{B'}$ son básicas. Para demostrar esto, observamos que todas las columnas de B' son linealmente independientes. Por la definición del índice ρ , la i_ρ -ésima componente de v_q es positiva, mientras que la misma componente en las demás columnas de B' es cero. Por lo tanto, v_q no puede ser una combinación lineal de las otras columnas de B' . Como las demás columnas son linealmente independientes, concluimos que todas las columnas de B' son linealmente independientes. Así, las coordenadas de x_B forman una base y el vector ζ' es una solución básica factible. A partir de (3.19), obtenemos:

$$\zeta' = \zeta + d_q t^*$$

Dado que $d_q \geq 0$ y $t \geq 0$, se deduce que el valor de la función objetivo no disminuye.

Decimos que en el Paso 3 hemos **actualizado** las cantidades sin prima a sus versiones con prima. Las cantidades primadas, o actualizadas, representan los nuevos valores con los que volveremos a iterar hasta dar con nuestro criterio de parada².

Observación 3.9. En el Paso 3-(I), elegimos la variable de entrada x_q como la variable correspondiente a la componente más grande de d_N . Al analizar el Paso 3, queda claro que la elección de la variable de entrada podría ser cualquier variable correspondiente a una componente positiva de d_N . La elección de la componente más grande se motiva por el hecho de que, en una iteración dada, esta elección parecería resultar en el mayor incremento en el valor de z .

²Se recuerda que este criterio consiste en comprobar las vecindades o puntos extremos contiguos y comprobar la optimidad de la solución actual. Si ninguno mejora, el proceso termina devolviendo como solución final la actual.

Paso 4. Volver al Paso 1 con los nuevos valores actualizados y empezar la siguiente iteración.

3.3.1. Terminación y ciclo

En la sección anterior se mostró que cada iteración del método Simplex da como resultado un valor de la función objetivo que es necesariamente mayor o igual que el valor de la función para la iteración anterior. Si ahora se mostrase que cada iteración tiene como resultado un valor de la función objetivo que es *estrictamente* mayor que el de la iteración anterior, el método terminaría de manera obligatoria en una solución óptima, en un número de pasos finito.

Sin embargo, es directamente falso que cada iteración produce un valor estrictamente mayor para la función objetivo. Se expone ahora un ejemplo de esta que muestra esta casuística, para después generalizar y formalizar cómo se tratan estos casos.

Supongamos que al final de la Fase 2 de una iteración hemos llegado a un sistema dado por

$$\left(\begin{array}{cccccc} x_1 & x_2 & x_3 & x_4 & x_5 & z \\ \hline 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 0 & 2 & -1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 & 3 \\ 0 & 3 & -1 & 0 & 0 & -1 & 1 \end{array} \right) \quad (3.24)$$

Entonces, la variable de entrada es x_2 y la columna de entrada v_q es la segunda columna. Recordando que la última columna da $(\xi_B, -\zeta)$, obtenemos que:

$$\frac{\xi_{i_1}}{v_1} = \frac{\xi_1}{v_1} = \frac{1}{1}, \quad \frac{\xi_{i_2}}{v_2} = \frac{\xi_4}{v_2} = \frac{0}{2} = 0, \quad \frac{\xi_{i_3}}{v_3} = \frac{\xi_5}{v_3} = \frac{3}{1}.$$

Por lo tanto, $t^* = 0$ y la variable saliente es x_4 . Se deduce entonces de (3.19) que $\zeta' = \zeta$ y el valor actualizado de z no cambia. De (3.21) obtenemos que la solución básica factible actualizada es:

$$\xi_{B'} = (\xi'_{i_1}, \xi'_{i_2}, \xi'_{i_3}) = (\xi'_1, \xi'_2, \xi'_5) = (1, 0, 3)$$

Observando (3.24), vemos que $t^* = 0$ porque $\xi_4 = 0$. En otras palabras, $t^* = 0$ es consecuencia de que la solución básica factible actual es degenerada. En este ejemplo, la solución básica actualizada también es degenerada. En el próximo ejemplo, mostramos que una solución básica degenerada no conduce necesariamente a otra solución degenerada sin aumento en el valor de z .

Supongamos ahora que al final del Paso 2 hemos llegado al sistema descrito por:

$$\left(\begin{array}{cccccc} x_1 & x_2 & x_3 & x_4 & x_5 & z \\ \hline 1 & 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & -2 & -1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 & 3 \\ 0 & 3 & -1 & 0 & 0 & -1 & 1 \end{array} \right)$$

La solución básica factible actual $(\xi_1, \xi_4, \xi_5) = (1, 0, 3)$ es de nuevo degenerada. La variable que entra a la base es x_2 también. Sin embargo, ahora los cocientes utilizados para determinar

3. El método Símplex

t^* son:

$$\frac{\xi_{i_1}}{v_1} = \frac{\xi_1}{v_1} = 1, \quad \frac{\xi_{i_3}}{v_3} = \frac{\xi_5}{v_3} = 3.$$

Así, $t^* = 1$ y la variable que se obtiene es x_1 . Dado que $d_q = 3$, $\zeta' = -1$, y $t^* = 1$, a partir de (3.19) obtenemos

$$\zeta' = -1 + 3(1) = 2.$$

Por lo tanto, el valor de z se incrementa. A partir de (3.21) calculamos la nueva solución básica factible como

$$\zeta'_B = (\zeta'_{i_1}, \zeta'_{i_2}, \zeta'_{i_3}) = (\zeta'_2, \zeta'_4, \zeta'_5) = (1, 2, 2),$$

la cual es no degenerada.

Ahora consideramos la situación general. Sea ζ_B una solución básica factible degenerada con una componente nula ζ_{i_p} . Si la componente correspondiente v_p de la columna saliente v es positiva, entonces $t^* = 0$, la solución básica factible actualizada será degenerada, y el valor actualizado de la función objetivo permanecerá sin cambios. Esto se deduce de (3.19), (3.20) y (3.21).

Surge entonces la cuestión de si puede existir una secuencia de iteraciones $I_k, I_{k+1}, \dots, I_{k+p}$ tal que las soluciones básicas factibles $\zeta_{B,k+j}$, $j = 0, 1, \dots, p$, sean todas degeneradas y tal que $\zeta_{B,k+p} = \zeta_{B,k}$. Si esto ocurre, entonces todos los valores actuales $\zeta_k, \zeta_{k+1}, \dots, \zeta_{k+p}$ de la función objetivo serán iguales y la secuencia de soluciones básicas degeneradas $\zeta_{B,k}, \zeta_{B,k+1}, \dots, \zeta_{B,k+p}$ se repetirá indefinidamente. Por lo tanto, el método simplex no terminará. El fenómeno recién descrito se denomina *ciclado* y se dice que el método Simplex *entra en ciclo*.

El aspecto negativo de este ciclado es obvio. Sin embargo, el aspecto positivo es que el ciclado aparece en muy raras ocasiones. Las buenas implementaciones del método Simplex contienen mecanismos para evitarlo.

De todo lo discutido en esta sección y la anterior, se obtiene el siguiente lema a modo de recopilación y resumen del comportamiento de la Fase II del método:

Lema 3.10. *La Fase II del método Simplex realiza una de las tres siguientes acciones:*

- (I) *Determina una solución optima en un número finito de pasos.*
- (II) *Determina que no existe una solución en un número finito de pasos.*
- (III) *Cicla indefinidamente.*

Se presenta ahora una regla muy simple para la selección de variables entrantes y salientes que hace imposible el ciclo. Esta regla es la *regla del índice más pequeño* o *regla de Bland*, nombrada así por su creador R. G. Bland [Bla77]. Otros métodos para prevenir el ciclo son el *método de perturbación* y la *regla de orden lexicográfico* relacionada, pero no se entrará a su discusión.

Antes de enunciar la regla del índice más pequeño, recordamos que señalamos en la observación 3.9 que la elección de la variable entrante puede ser cualquier variable correspondiente a una componente positiva de d_N . Asimismo, es posible elegir cualquier variable x_{i_t} , en la cual se alcanza t^* en (3.20) de la sección anterior.

Definición 3.11 (Regla del Índice Más Pequeño). De entre todas las posibles variables entrantes, se elige la de índice más pequeño. De entre todas las posibles variables salientes, se elige la de índice más pequeño.

Observación 3.12. De acuerdo con la regla del índice más pequeño, la elección de la variable entrante será, en general, diferente de la elección de aquella para la cual d_i es un máximo. La elección de la variable saliente es la misma que la utilizada anteriormente.

Observación 3.13. La experiencia computacional indica que usar la regla del subíndice más pequeño o la regla lexicográfica en lugar de la regla del $\max d_i$ incrementa considerablemente el tiempo requerido por el método Simplex para encontrar una solución óptima. Dado que se han reportado muy pocos casos de ciclos, algunos códigos computacionales no incorporan ninguna rutina anti-ciclos. Otros códigos introducen una rutina anti-ciclos solo después de que ocurre una secuencia de bases degeneradas de una longitud especificada. La rutina anti-ciclos conducirá entonces a una base no degenerada, momento en el cual se puede reanudar el uso de la regla del coeficiente más grande.

3.4. Fase I del método Simplex

Como se discutía al inicio de la explicación del método, en esta primera parte nos centraremos en determinar una solución básica factible o, en su defecto, determinaremos que no existe dicha solución, en cuyo caso diremos que el problema no tiene solución óptima.

Partimos ahora de un problema en forma canónica, **PPLC**. Añadimos las variables de holgura necesarias para nuestro problema, convirtiendo así las desigualdades en igualdades en nuestro problema. El vector de variables de holgura viene dado por:

$$x_s = (x_{n+1}, \dots, x_{n+m}), \quad x_{n+i} \geq 0, \quad i = 1, \dots, m,$$

y entonces escribimos el problema como:

$$\begin{aligned} & \text{Maximizar } \langle b, x \rangle \\ & \text{sujeto a } (A, I) \begin{pmatrix} x \\ x_s \end{pmatrix} = c, \quad x \geq 0, \quad x_s \geq 0 \end{aligned} \tag{3.25}$$

Si suponemos que $c \geq 0$, el vector $(x, x_s) = (0_n, c)$ es una solución básica factible para el problema en (3.25). Luego pasamos inmediatamente a la fase II con la solución básica factible inicial $(\tilde{x}, \tilde{x}_s) = (0_n, c)$.

Si algunas componentes de c son negativas, el vector $(0_n, c)$ sigue siendo una solución básica del sistema de ecuaciones en (3.25), pero claro, no es factible. Por lo tanto, recurrimos a otro procedimiento e introducimos un problema auxiliar, que discutiremos en la sección siguiente.

3.4.1. Problema Auxiliar

Se parte del siguiente problema:

Problema: Minimizar x_0 sujeto a

3. El método Simplex

$$(A, I, -e) \begin{pmatrix} x \\ x_s \\ x_0 \end{pmatrix} = c \quad (3.26)$$

$$x \geq 0, \quad x_s \geq 0, \quad x_0 \geq 0,$$

donde $e = (1, \dots, 1)$ y I es la matriz identidad de orden $m \times m$.

Si desglosamos por componentes, el sistema anterior queda como:

$$\sum_{j=1}^n a_{ij}x_j + x_{n+i} - x_0 = c_i, \quad i = 1, \dots, m,$$

$$x_j \geq 0, \quad j = 0, 1, \dots, n+m.$$

Un vector (ξ, ξ_s) es factible para (3.25) si y sólo si $(\xi, \xi_s, 0)$ es factible para el problema auxiliar. Por lo tanto, (ξ, ξ_s) es factible para la relación (3.25) si y sólo si $(\xi, \xi_s, 0)$ es óptimo para el problema auxiliar.

Si escribimos la función objetivo del problema auxiliar como “Maximizar $w = -x_0$ ”, entonces el problema auxiliar se convierte en un PPL en forma canónica al cual podemos aplicar la Fase II del método Simplex, siempre que podamos obtener una solución básica factible inicial. Una solución básica para el sistema (3.26) es $(\xi, \xi_s, \xi_0) = (0, c, 0)$. Esta solución, sin embargo, no es factible ya que algunas componentes de c son negativas. Podemos obtener una solución básica factible a partir de esta solución reemplazando una variable básica apropiada por x_0 .

Obtenemos ahora un sistema equivalente a (3.26) y que tenga una solución básica factible. Sea c_p la componente negativa menor de c . Esto es,

$$c_p = \min\{c_i : i = 1, \dots, m\}$$

Si hay más de una componente con esas características, se elige la de menor índice. A continuación, se pivotea en x_0 en la fila p del sistema (3.26). El resultado será un sistema equivalente dado por

$$(A', e_1, \dots, e_{p-1}, -e, e_{p+1}, \dots, e_m, e_p) \begin{pmatrix} x \\ x_s \\ x_0 \end{pmatrix} = c', \quad (3.27)$$

donde $e = (1, \dots, 1)$ y

$$c'_p = -c_p \geq 0, \quad c'_i = c_i - c_p \geq 0, \quad i = 1, \dots, p-1, p+1, \dots, m.$$

El sistema (3.27) lo desglosamos por componentes, queda como

$$\sum_{j=1}^n a'_{ij}x_j - x_{n+p} + x_{n+i} = c'_i, \quad i \neq p,$$

$$\sum_{j=1}^n a'_{pj}x_j - x_{n+p} + x_0 = c'_p.$$

Por lo tanto, una solución básica factible del sistema (3.27) es

$$\begin{aligned}\bar{\xi}_0 &= c'_p, \\ \bar{\xi}_{n+i} &= c'_i, \quad i = 1, \dots, p-1, p+1, \dots, m, \\ \bar{\xi}_i &= 0, \quad i = 1, \dots, n, n+p,\end{aligned}\tag{3.28}$$

y

$$w = -x_0 = -c'_p - x_{n+p} + \sum_{j=1}^n a'_{pj} x_j.\tag{3.29}$$

Las variables básicas son

$$x_{n+1}, \dots, x_{n+p-1}, x_0, x_{n+p+1}, \dots, x_{n+m}.\tag{3.30}$$

Hemos transformado el problema auxiliar (3.26) en el siguiente problema:

Maximizar w como se indica en (3.29), sujeto a (3.28) y $x_i \geq 0$, $i = 0, 1, \dots, n+m$.

Para este problema hemos determinado una solución básica factible dada por (3.28), con variables básicas dadas por (3.30). Ahora podemos aplicar la fase II al problema en esta forma. Si incorporamos una subrutina anti-ciclos en nuestra implementación de la Fase II, entonces, de acuerdo con el Lema 3.10, determinaremos, en un número finito de pasos, que el problema auxiliar o bien no tiene solución o bien encontraremos una solución óptima al problema auxiliar.

Si el problema auxiliar no tiene solución, entonces el PPL original (3.25) no tiene solución factible. Esto se debe a que, como vimos, un vector (ξ, ξ_s) es factible para el problema original si y solo si $(\xi, \xi_s, 0)$ es óptimo para el problema auxiliar.

La alternativa restante es que el problema auxiliar tenga solución. Ahora exploramos las posibilidades para esta alternativa. La variable x_0 es básica inicialmente. Al aplicar la fase II del método Simplex al problema auxiliar, observamos que nuestra regla para elegir la variable saliente es tal que, si en alguna iteración x_0 es candidata a salir de la base, entonces x_0 será elegida como la variable saliente. Si en alguna iteración I_k la variable x_0 sale de la base, entonces en la iteración I_{k+1} , la variable x_0 tendrá valor cero. La solución (ξ, ξ_s, ξ_0) de las ecuaciones de restricción en I_{k+1} también será solución de (3.26). Por lo tanto, si $\xi_0 = 0$, entonces $w = 0$ y (ξ, ξ_s) es óptima para el problema inicial. Así, si x_0 sale de la base en alguna iteración I_k , en la siguiente iteración I_{k+1} tendremos una solución para el problema auxiliar y por tanto una solución básica factible para la relación (3.25). Entonces podemos comenzar la fase II con esta solución básica factible.

Una solución óptima también podría ocurrir en las siguientes circunstancias:

- (i) x_0 básica, $w = 0$, y
- (ii) x_0 básica, $w \neq 0$.

Si ocurre (ii), entonces el problema original no tiene solución factible. Ahora mostramos que (i) es imposible. Si (i) ocurriera, entonces $x_0 = -w = 0$. Como la iteración anterior no terminó con una solución óptima, $x_0 \neq 0$ al comienzo de la iteración anterior y se volvió cero en la iteración previa. Si esto ocurrió, entonces x_0 fue candidata a salir de la base pero no lo hizo. Esto contradice la regla para determinar si x_0 deja la base.

El siguiente lema se deduce de la discusión en esta sección.

3. El método Símplex

Lema 3.14. Si se incluye una rutina anti-ciclos, entonces la fase I del método Símplex determina o bien que no existe una solución factible, o bien una solución básica factible.

Los lemas (3.10) y (3.14) nos llevan al siguiente teorema.

Teorema 3.15. El método Símplex que incorpora una rutina anti-ciclos, aplicado a un problema de PL , determina una de las siguientes opciones en un número finito de iteraciones:

- (i) No existe solución factible.
- (ii) El problema no tiene una solución óptima.
- (iii) Se encuentra una solución óptima.

4. Programación convexa

La programación convexa constituye una rama fundamental de la parte de optimización matemática, centrada esta vez en el estudio de problemas donde la función objetivo y el conjunto de restricciones poseen una estructura convexa. El desarrollo de la programación convexa está profundamente vinculado a la evolución de la [PL](#) y al crecimiento del Análisis Funcional en el siglo XX. Aunque los primeros trabajos en optimización se centraban en problemas lineales (como el método Simplex), pronto surgió la necesidad de tratar funciones objetivo y restricciones no lineales. En [\[Tik96\]](#) se explora la historia y orígenes de este tipo de problemas.

En el capítulo anterior estudiamos problemas donde tanto las restricciones como la función objetivo eran lineales, pero ahora asumiremos que ambas son convexas. También se comentaba en la Sección 2 la utilidad de técnicas más sencillas para abordar problemas de menor complicación. Sin embargo, como se demuestra en [\[ASOM12\]](#), cuando la complejidad del problema aumenta, es necesario hacer uso de la [PC](#), pues la lineal no basta para hacer frente a estos¹.

En los últimos años, el estudio de la optimización convexa y la teoría de la convexidad ha experimentado un notable avance, tanto en el ámbito teórico como en sus aplicaciones prácticas. Este progreso se ha visto impulsado por el rápido desarrollo de áreas como la ciencia de datos y el [ML](#), que emplean ampliamente estas técnicas para abordar problemas de clasificación y regresión.

En el estudio de la optimización, una vez abordados los problemas lineales, los [PPC](#) suelen representar el siguiente paso de interés antes de adentrarse en el extenso campo de la programación no lineal, al cual pertenecen. Por tanto, el análisis de [PPC](#) nos permite abordar una amplia gama de aplicaciones en distintas áreas, proporcionando un marco teórico sólido. En lo que sigue, nos adentraremos en el estudio formal de este tipo de problemas, introduciendo sus principales características y cómo abordarlos.

4.1. El problema de la optimización convexa

Como se ha ido adelantando, para el caso convexo se trata de la optimización de una función objetivo, con la particularidad de que esta debe ser convexa, sujeta a unas restricciones que han de ser, a su vez, convexas.

Hacer esta asunción acerca de las restricciones y la función objetivo, nos permite obtener información mucho más detallada sobre el comportamiento de las soluciones. De hecho, en [\[Roc93\]](#) se afirma: “*La gran línea divisoria en la optimización no está entre lo lineal y lo no lineal, sino entre lo convexo y lo no convexo*”.

Esta cita destaca la importancia fundamental de la convexidad en la optimización, ya que los problemas convexos permiten encontrar soluciones globales de manera eficiente, a

¹Un ejemplo es el propuesto en [\[ASOM12\]](#), en el que se demuestra que cuando los coeficientes son constantes, el método Simplex resuelve estos problemas sin dificultad. Sin embargo, cuando dejan de ser constantes, el método Simplex no es eficaz, y se propone una técnica que utiliza la convexidad de dichos coeficientes, haciendo uso de la teoría de aproximación por funciones lineales a trozos.

4. Programación convexa

diferencia de los problemas no convexos que pueden presentar múltiples óptimos locales.

En optimización convexa, aunque los problemas de minimización y maximización están estrechamente relacionados, se abordan de manera diferente debido a las propiedades matemáticas que definen las funciones convexas y cónicas. Mientras que una función convexa garantiza la existencia de un mínimo global, no sucede lo mismo con los máximos. Por ello, los problemas de maximización suelen reformularse como problemas de minimización del negativo de la función, permitiendo así el uso de métodos estándar de optimización convexa [BV04, NW99].

En el estudio de la convexidad, por claridad y diversidad del trabajo, seguiremos la línea de la **minimización**. Esto nos llevará a establecer un marco en el que trabajar sobre las funciones convexas. Como se ha mencionado anteriormente, los problemas de optimización convexa constituyen un caso particular dentro de la programación no lineal. La diferencia más significativa entre los problemas convexos y los no convexos radica en que, en los primeros, cualquier óptimo local es también un óptimo global, lo que brinda una garantía clave en su resolución. Más abajo se ve esto (Teorema 4.1), cuyo desarrollo se puede encontrar en [GR20].

Dentro de esta área, la **PC** se refiere a la formulación específica de problemas convexos, caracterizados por funciones objetivo convexas y restricciones convexas (o **afines**, en el caso de restricciones de igualdad). Para resolver este tipo de problemas, se pueden aplicar diversos métodos numéricos, entre los cuales destaca el *método del gradiente proyectado*, que se desarrollará en este capítulo. Este método resulta especialmente útil cuando el problema presenta restricciones lineales o no lineales que *limitan el espacio factible*, ya que permite proyectar el gradiente de la función objetivo sobre el subespacio de las variables independientes, reduciendo así la dimensionalidad efectiva del problema. De esta manera, el gradiente proyectado proporciona una dirección de descenso válida dentro del espacio factible, preservando la viabilidad de las soluciones iterativas y garantizando convergencia en contextos convexos bajo condiciones adecuadas.

Sea $f : \mathbb{R}^n \rightarrow \mathbb{R}$ diferenciable y continua y sea $X \subset \mathbb{R}^n$ un conjunto convexo y compacto. Se plantea el problema

$$\begin{aligned} & \min f(x) \\ & \text{sujeto a } x \in X. \end{aligned} \tag{4.1}$$

Teorema 4.1. En un problema de minimización convexa, todo óptimo local es un óptimo global.

Demostración. Sea x^* un óptimo local del problema, es decir,

$$f(x^*) = \inf\{f(x) : x \in X\}$$

y sea $r > 0$ tal que $B(x^*, r) \subset X$ es un entorno de x^* , para el que $f(x^*) \leq f(x)$ para todo x en $B(x^*, r)$, por ser un óptimo local.

Sea ahora $y \in X$ un punto del conjunto factible, tal que $y \notin B(x^*, r)$. Como X es convexo, se tiene que:

$$z = \alpha x^* + \beta y, \text{ donde } \alpha + \beta = 1 \text{ y } 0 \leq \alpha, \beta \leq 1, \text{ con } z \in X.$$

Ahora, al ser f una función convexa,

$$f(z) \leq \alpha f(x^*) + \beta f(y).$$

Por otra parte, la combinación convexa de α y β se puede elegir tal que $z \in B(x^*, r)$. Usando esto último se tiene que:

$$\begin{aligned} f(x^*) &\leq f(z) \leq \alpha f(x^*) + \beta f(y) \\ (1-\alpha)f(x^*) &\leq \beta f(y) \\ \beta f(x^*) &\leq \beta f(y) \\ f(x^*) &\leq f(y). \end{aligned}$$

Probando así que x^* es un óptimo global en todo X , por la arbitrariedad de y . \square

Ante un problema de este tipo, lo primero que nos planteamos es establecer condiciones de optimalidad que nos ayuden a detectar los puntos $x \in X$ que son candidatos a solución óptima.

Puesto que la condición de optimalidad en problemas sin restricciones ($\nabla f(x) = 0$) ya no es válida cuando se plantea un problema con restricciones, necesitamos encontrar otras condiciones de optimalidad alternativas aplicables a este nuevo problema.

Teorema 4.2. Si x^* es mínimo local del problema, entonces

$$\langle \nabla f(x^*), x - x^* \rangle \geq 0 \quad \forall x \in X.$$

Además, si f es convexa, la condición es suficiente.

Demostración. Sea x^* mínimo local, $0 < \epsilon < 1$.

$$f(x^* + \epsilon(x - x^*)) = f(x^*) + \langle \nabla f(x^* + \epsilon s(x - x^*)), x^* + \epsilon(x - x^*) - x^* \rangle, \quad 0 < s < 1.$$

Si $x^* \in X$ y $x \in X$, entonces, por la convexidad de X , toda combinación lineal de x y x^* también pertenece a X :

$$x^* + \epsilon(x - x^*) = (1 - \epsilon)x^* + \epsilon x, \quad x \in X.$$

Supongamos por reducción al absurdo que no se da la condición del enunciado; como ∇f es continua, para $\epsilon > 0$ suficientemente pequeño

$$\langle \nabla f(x^* + \epsilon s(x - x^*)), x - x^* \rangle < 0,$$

y puesto que $\epsilon s > 0$, entonces $\epsilon \langle \nabla f(x^* + \epsilon s(x - x^*)), x - x^* \rangle < 0$.

De donde se obtiene que

$$f(x^* + \epsilon(x - x^*)) < f(x^*),$$

luego se tiene una contradicción, pues x^* es mínimo local.

Veamos ahora qué ocurre si f es una función convexa. Supongamos que f es una función convexa; entonces ha de verificarse

4. Programación convexa

$$f(x) \geq f(x^*) + \langle \nabla f(x^*), x - x^* \rangle \quad \forall x \in X,$$

donde $f(x^*) + \langle \nabla f(x^*), x - x^* \rangle = 0$, $\forall x \in X$ es la ecuación del hiperplano que ha de quedar por debajo de cualquier punto de $f(x)$.

Si x^* verifica que $\langle \nabla f(x^*), x - x^* \rangle \geq 0 \quad \forall x \in X$, entonces

$$f(x) \geq f(x^*) + \langle \nabla f(x^*), x - x^* \rangle \geq f(x^*) \quad \forall x \in X.$$

Luego x^* es un mínimo global.

□

Observación 4.3. Dado un punto $x \in \text{int}(X)$, consideremos un entorno del punto x^* (mínimo local), tal que $x \in B(x^*, r)$, entonces el punto simétrico respecto al centro x^* también pertenece a la bola $B(x^*, r)$ y se tiene que:

$$\langle \nabla f(x^*), x - x^* \rangle \geq 0 \quad \text{y} \quad -\langle \nabla f(x^*), x - x^* \rangle \geq 0,$$

esto implica que

$$\nabla f(x^*) = 0.$$

El problema se presenta cuando x^* es un punto frontera de X . En este caso en concreto no se puede usar el criterio que se suele usar en el método del gradiente clásico, $\nabla f(x^*) = 0$. Como el conjunto X es cerrado, esto implica que en todo entorno centrado en un punto que se encuentra en la frontera existen puntos que no están contenidos en la bola original con centro en x^* ; en consecuencia, existen direcciones de descenso para el caso del problema de minimización que podrían no ser factibles.

La idea detrás de la condición es que si x^* es mínimo local, entonces en dicho punto no es posible encontrar ninguna dirección de descenso.

Se presentan a continuación resultados importantes sobre proyecciones en conjuntos convexos y compactos, que serán necesarios para el desarrollo del mencionado método de optimización.

Definición 4.4. Dado $z \notin X$, la **proyección** de z sobre X es el $x^* \in X$ tal que

$$\|z - x^*\| = \min_{x \in X} \|z - x\|. \quad (4.2)$$

Es decir, para calcular la proyección de z sobre X con una norma cualquiera, al fin y al cabo se ha de resolver un problema de optimización (en concreto, minimización) como se presenta en (4.2). Denotaremos como $\Pi_X(z)$ a la proyección sobre X del punto z .

Proposición 4.5. Sea $X \subset \mathbb{R}^n$ un conjunto con la norma euclídea, entonces se dan las siguientes propiedades:

(I) Si X es convexo y compacto la proyección es única.

(II) $x^* = \Pi_X(z)$ si $\langle z - x^*, x - x^* \rangle \leq 0, \forall x \in X$.

4.1.1. Método de direcciones factibles

En los problemas de optimización con restricciones, especialmente cuando el conjunto factible es convexo y compacto, resulta esencial desarrollar métodos que aseguren que las soluciones iterativas permanezcan dentro del conjunto permitido. Uno de los enfoques más fundamentales y ampliamente utilizados es el *método de direcciones factibles*, que servirá como base para el desarrollo del *método del gradiente proyectado*.

Este método propone una estrategia iterativa para minimizar una función diferenciable $f(x)$ sujeta a restricciones del tipo $x \in X$, donde $X \subset \mathbb{R}^n$ es un conjunto convexo. A cada iteración, el algoritmo genera una dirección factible y de descenso desde el punto actual, asegurando que la nueva iteración permanezca dentro del conjunto X y que el valor de la función objetivo disminuya.

Sea el problema formulado en (4.1). Los métodos de direcciones factibles son métodos de iteración tales que:

- (I) Dado un punto inicial x^0 , se generan los puntos $x^{k+1} = x^k + \alpha^k d^k$ siendo d^k una dirección factible en x^k (i.e., si existe $\mu > 0$ tal que $x^k + \mu d^k \in X$) y $\alpha^k \in (0, 1]$ es la longitud de paso (ahora se discutirá su elección), donde k representa el número o índice de la iteración.
- (II) Por ser X convexo, el conjunto de direcciones factibles en x^k viene dado por:

$$\gamma(\bar{x}^k - x^k) \quad \text{siendo } \bar{x}^k \in X, \gamma > 0.$$

- (III) Puesto que una dirección puede expresarse como la diferencia de sus extremos, esto hace que

$$x^k + \alpha^k d^k = x^k + \bar{\alpha}^k (\bar{x}^k - x^k), \quad \bar{\alpha}^k \in (0, 1], \quad \bar{x}^k \in X.$$

Por tanto:

$$x^{k+1} = x^k + \alpha^k (\bar{x}^k - x^k) = x^k (1 - \alpha^k) + \alpha^k \bar{x}^k \in X$$

pues X es un conjunto convexo y $\bar{x}^k, x^k \in X$.

Luego este método garantiza que, dados dos puntos de X , se puede generar otro punto del conjunto tomando α de forma conveniente. Además, nos interesa que el método sea de descenso; en otras palabras, buscamos direcciones factibles de descenso, es decir,

$$\langle \nabla f(x^k), d^k \rangle < 0$$

pues así se obtendrá que $f(x^{k+1}) < f(x^k)$.

Cabe preguntarse llegado este punto si justamente es posible encontrar direcciones de descenso; la respuesta es que existirán siempre que x^k no sea un punto **estacionario**, esto es:

$$\langle \nabla f(x^k), x - x^k \rangle \geq 0 \quad \forall x \in X.$$

Si no es estacionario existe $\bar{x} \in X$ tal que $\langle \nabla f(x^k), \bar{x} - x^k \rangle < 0$. Por tanto $\bar{x} - x^k$ es dirección factible y de descenso.

El siguiente paso natural y que se ha de decidir en los métodos basados en gradiente es el cómo elegir longitudes de paso para el descenso. Pues bien, las longitudes de paso se pueden elegir según los mismos criterios que para el caso de optimización sin restricciones. Por ejemplo, en el método de gradiente descendente clásico:

4. Programación convexa

- Longitud de paso constante: $\alpha^k = \alpha > 0$.
- Buscar un α^k que minimice

$$\phi(\alpha) = f(x^k - \alpha \nabla f(x^k)).$$

Esto se suele conseguir usando condiciones como la condición de *Armijo* o la de *Wolfe*, [NW99].

No obstante, para asegurar la factibilidad desde el inicio se debe imponer $\alpha \in (0, 1]$. Si $\alpha \leq 0$ ó $\alpha > 1$, entonces el punto generado no queda dentro del conjunto X .

Se enuncia a continuación un teorema que será importante para el desarrollo del método del gradiente proyectado. Sin embargo, no se incluye su demostración pues no es estrictamente necesaria, y puede encontrarse en [Bla25].

Teorema 4.6. *Sea $\{x^k\}$ una sucesión generada por el método de direcciones factibles $x^{k+1} = x^k + \alpha^k d^k$, tal que la sucesión de direcciones $\{d^k\}_{k \in K}$ verifica:*

1. $\limsup_{k \rightarrow \infty} \|\alpha^k d^k\| < +\infty$.
2. $\limsup_{k \rightarrow \infty} \langle \nabla f(x^k), d^k \rangle < 0$.

Además, si la longitud de paso se elige mediante las reglas de *Armijo* o *minimización*, entonces todo punto de acumulación de $\{x^k\}$ es estacionario.

4.1.2. Método del gradiente proyectado

Como se venía comentando y se explica en [Bla25], el *método del gradiente proyectado* es una técnica iterativa utilizada para resolver problemas de optimización con restricciones, particularmente útil cuando el conjunto factible es convexo y compacto. La idea principal consiste en avanzar en la dirección del gradiente negativo y luego proyectar el resultado sobre el conjunto factible, manteniendo así la viabilidad de la solución.

En particular, se trata de un método de direcciones factibles en los que la dirección d^k se elige usando la siguiente regla:

$$d^k = \Pi_X(x^k - s^k \nabla f(x^k)) - x^k, \quad s^k > 0.$$

Por tanto, se parte de un punto x^0 y se genera una sucesión $x^{k+1} = x^k + \alpha^k d^k$ con la dirección $d^k = \Pi_X(x^k - s^k \nabla f(x^k)) - x^k$. Para simplificar la notación, denotaremos

$$\bar{x}^k = \Pi_X(x^k - s^k \nabla f(x^k)).$$

Por tanto, si $\alpha^k = 1, \forall k$, se obtiene que $x^{k+1} = x^k + d^k = x^k + (\bar{x}^k - x^k) = \bar{x}^k$.

Condición de parada: El método se detiene cuando se obtiene o genera un punto \bar{x} tal que $\Pi_X(\bar{x} - s \nabla f(\bar{x})) = \bar{x}$. Esto se debe a que si \bar{x} es estacionario, entonces

$$\langle \nabla f(\bar{x}), x - \bar{x} \rangle \geq 0 \quad \forall x \in X.$$

Luego para $s > 0$ se tiene que

$$-s\langle \nabla f(\bar{x}), x - \bar{x} \rangle \leq 0, \forall x \in X,$$

$$\langle \bar{x} - s\nabla f(\bar{x}) - \bar{x}, x - \bar{x} \rangle \leq 0, \forall x \in X.$$

Que es justamente la condición de que \bar{x} sea la proyección sobre X de $\bar{x} - s\nabla f(\bar{x})$ (Proposición 4.5). Luego se concluye que

$$\bar{x} \text{ es estacionario} \iff \bar{x} = \Pi_X(\bar{x} - s\nabla f(\bar{x}))^T.$$

Para concluir con el desarrollo del método se han de describir también las **longitudes de paso**, como en todo método basado en el gradiente. Se pueden usar los siguientes métodos para obtener dichas longitudes:

- **Minimización restringida:** Sea $s^k = s > 0, \forall k$ y sea α^k tal que

$$f(x^k + \alpha^k(\bar{x}^k - x^k)) = \min_{\alpha \in (0,1)} f(x^k + \alpha^k(\bar{x}^k - x^k)).$$

No obstante, en ocasiones puede resultar interesante tomar un $\alpha > 1$, siempre que sigamos obteniendo puntos factibles y así lograr una convergencia más rápida a la solución óptima.

- **Armijo sobre la dirección factible:** Sea $s^k = s, \forall k$ y sean $\beta, \sigma \in (0,1)$ fijos. Se toma entonces $\alpha^k = \beta^{m_k}$, siendo m_k el menor entero $m > 0$ que cumple que

$$f(x^k) - f(x^k + \beta^m(\bar{x}^k - x^k)) > -\sigma\beta^m \langle \nabla f(x^k), \bar{x}^k - x^k \rangle.$$

Teorema 4.7. *Sea $\{x^k\}$ una sucesión generada por el método del gradiente proyectado con α^k elegido mediante minimización restringida o Armijo sobre la dirección factible. Entonces, todo punto de acumulación es estacionario.*

Demostración. Haciendo uso del Teorema 4.6, si \bar{x} es un punto de acumulación no estacionario, entonces basta probar que:

1. $\limsup_{k \rightarrow \infty} \|\bar{x}^k - x^k\| < +\infty$
2. $\limsup_{k \rightarrow \infty} \langle \nabla f(x^k), \bar{x}^k - x^k \rangle < 0$

Si esto se verifica, entonces podemos concluir que todo punto de acumulación de $\{x^k\}$ es estacionario. En 1) se tiene que $\bar{x}^k = \Pi_X(x^k - s\nabla f(x^k))$.

Dado que la proyección es continua y lo es la norma, entonces se tiene que

$$\limsup_{k \rightarrow \infty} \|\bar{x}^k - x^k\| = \|\Pi_X(\bar{x} - s\nabla f(\bar{x})) - \bar{x}\| < \infty,$$

que está acotado pues X es compacto y además $\Pi_X(\bar{x} - s\nabla f(\bar{x}))$ y \bar{x} son elementos de X . Para probar 2), se tiene que, según la condición de que \bar{x}^k es la proyección de $x^k - s\nabla f(x^k)$ sobre X (Proposición 4.5),

$$\langle x^k - s\nabla f(x^k) - \bar{x}^k, x - \bar{x}^k \rangle \leq 0 \quad \forall x \in X.$$

Ahora, si $x = x^k$:

$$\langle x^k - s\nabla f(x^k) - \bar{x}^k, x^k - \bar{x}^k \rangle \leq 0,$$

$$\|x^k - \bar{x}^k\|^2 - s\langle \nabla f(x^k), x^k - \bar{x}^k \rangle \leq 0$$

$$\Rightarrow \langle \nabla f(x^k), \bar{x}^k - x^k \rangle \leq -\frac{1}{s}\|x^k - \bar{x}^k\|^2.$$

Por último, tomando límite en k :

$$\begin{aligned} \lim_{k \rightarrow \infty} \sup_{k \in K} \langle \nabla f(x^k), \bar{x}^k - x^k \rangle &\leq -\frac{1}{s} \lim_{k \rightarrow \infty} \sup_{k \in K} \|x^k - \bar{x}^k\|^2 \\ &= -\frac{1}{s} \|\bar{x} - \Pi_X(\bar{x} - s\nabla f(\bar{x}))\|^2 < 0. \end{aligned}$$

Así pues, por el resultado anterior, se tiene que \bar{x} debe coincidir con $\Pi_X(\bar{x} - s\nabla f(\bar{x}))$ si y sólo si, el punto \bar{x} es estacionario, y como por hipótesis el punto \bar{x} no es estacionario, entonces $\bar{x} \neq \Pi_X(\bar{x} - s\nabla f(\bar{x}))$. Luego $\|\bar{x} - \Pi_X(\bar{x} - s\nabla f(\bar{x}))\|^2 > 0$. En consecuencia, tras probar 1) y 2), por el Teorema 4.6, se tiene que todo punto de acumulación es estacionario. \square

Obtenemos finalmente un método derivado de los clásicos de descenso de gradiente para el caso de la optimización sin restricciones, que es la **PC**. Como se hacía para la **PL**, el desarrollo hecho busca sentar unas bases teóricas, para luego ofrecer un método de resolución y búsqueda de estos problemas de optimización. Se reitera que el caso convexo es la generalización del lineal, por lo tanto, los métodos y técnicas desarrollados han de ser algo más complejos y elaborados. Se establece de esta manera una analogía entre la **PL** y la **PC**. En el caso de la **PL**, era Simplex, ahora, el método de gradiente presentado.

En esta línea, en la siguiente sección, se vuelve a dar un paso más allá; en lugar de trabajar en dimensión finita, se presentan dos resultados muy importantes para el caso de dimensión infinita.

4.2. El teorema de Krein-Milman

Una vez se desarrolla la optimización convexa en el ámbito de la dimensión finita, tiene sentido la generalización y el paso a la dimensión infinita. Primeramente, es crucial entender correcta e íntegramente cómo se trabaja con las funciones y restricciones convexas. Es por esto que se presentan a continuación resultados imprescindibles para la construcción del marco teórico del *Teorema de Krein-Milman* y el *Principio del máximo de Bauer*. Se seguirá el desarrollo hecho en [Su1].

Será necesario revisar el concepto de *punto extremo*, explicado en secciones anteriores, así como recordar conceptos fundamentales relativos al Análisis Funcional. En concreto, se desarrollará la teoría necesaria para probar el Teorema de *Krein-Milman*, resultado de gran importancia para entender la estructura de los conjuntos convexos, sobre los cuales se desarrollan los problemas de optimización.

Se recuerdan los conceptos más importantes acerca de las topologías débiles de un espacio normado y la topología débil de su dual. La principal razón del estudio de estos conceptos es la búsqueda de subconjuntos compactos en espacios normados de *dimensión infinita*.

En general, la topología asociada a la norma tiene demasiados abiertos, por lo que se intenta buscar topologías más pequeñas y donde se puedan encontrar un mayor número de compactos. Al estudiar las propiedades de estas nuevas topologías que definiremos, podremos demostrar importantes resultados que ayudarán a la demostración del teorema de Krein-Milman.

Definición 4.8. Sea X un espacio normado. La topología débil de X , a menudo denotada por $\sigma(X, X^*)$, es la topología inicial en X para los elementos de su dual, X^* , es decir, la topología con menos abiertos en X que hace continuos los funcionales de X^* .

Análogamente, se define la topología débil-* de X^* , denotada por $\sigma(X^*, X)$, como la topología inicial en X^* para los elementos de X , es decir, la menor topología de X^* que hace continuos los elementos de X .

Se recordarán las propiedades necesarias acerca de estas topologías en el desarrollo de lo siguiente. También se recuerda al lector que denotaremos las propiedades de los conjuntos con respecto a las topologías débiles con el adverbio *débilmente*. Por ejemplo, un conjunto que sea cerrado para la topología $\sigma(X, X^*)$ lo llamaremos un conjunto débilmente cerrado.

Como se hacía en el caso de los problemas de características lineales, se pondrá como eje de estudio principal los puntos extremos. Ya se motivó el uso y análisis de dichos puntos en los problemas de optimización. Es por ello que también lo haremos en el caso de los resultados más importantes de la optimización convexa.

Observación 4.9. Nótese que todo punto extremo de un conjunto X es en sí un subconjunto extremal minimal (en el sentido del más pequeño, no puede tener subconjuntos). En la búsqueda de los puntos extremos de X se deben buscar subconjuntos extremales cada vez más pequeños. De hecho, en virtud del Lema 1.10 en el que se habla de subconjuntos extremales, podemos intuir en qué dirección se ha de buscar para encontrar otro subconjunto más pequeño, bajo ciertas hipótesis. Si X tiene más de un punto, cabe preguntarse si existe un funcional lineal f tal que no sea constante en X , en cuyo caso se necesita la certeza de que alcance su máximo en X . Para conseguirlo, solo se necesita que X sea compacto, en una topología que haga continuo a f . Justamente, bastará por tanto que los funcionales lineales continuos para una cierta topología separen puntos y que A sea compacto en esa topología.

El siguiente teorema es consecuencia del Teorema de Hahn-Banach 1.13. De hecho, es dar una versión de los famosos teoremas de separación del Análisis Funcional pero para las topologías débil y débil-*.

Teorema 4.10. *Sea X un espacio normado. Si A es un subconjunto convexo y débilmente cerrado de X y $x_0 \notin A$, entonces existe un funcional $f \in X^*$ tal que*

$$f(x_0) > \sup_{a \in A} f(a), \forall a \in A.$$

Demostración. La demostración del teorema es trivial, pues si A es débilmente cerrado, entonces también lo es para la norma. \square

Teorema 4.11 (Mazur). *Si A es un conjunto convexo del espacio normado X , entonces la clausura débil de A coincide con su clausura para la norma.*

4. Programación convexa

Se recuerda al lector que se denota por $J_X(x) \in X^{**}$, $\forall x \in X$, a la aplicación $J_X : X \rightarrow X^{**}$ y recibe el nombre de *inyección canónica* del espacio normado X en su bidual. Esto es, el dual del dual de X . De hecho, se recuerda también que cuando el bidual coincide con el espacio de partida, el espacio se dice reflexivo ($X = X^{**}$).

Teorema 4.12. *Sea X un espacio normado. Si A es un subconjunto débilmente compacto de X , entonces todo subconjunto extremal cerrado para la topología débil de X contiene un punto extremal de A . En particular, A tiene puntos extremos. Análogamente, si A es un subconjunto débilmente-* compacto de X^* , entonces todo subconjunto extremal cerrado para la topología débil de A^* contiene un punto extremal. En particular, A tiene puntos extremos.*

Demostración. Basta hacer la prueba para la topología débil-*, pues si A es débilmente compacto en X entonces $J_X(A)$ es débilmente-* compacto en el bidual y además sus subconjuntos débilmente cerrados serán débilmente-* cerrados en X^{**} .

Por tanto, si partimos de B un subconjunto extremal débilmente-* cerrado en A , la familia Ω de los subconjuntos extremales débilmente-* cerrados no vacíos de B deberá tener un elemento minimal, que además se reduce a un punto. Este será un punto extremal, justamente.

La idea de la demostración se basa en que la familia de subconjuntos extremales Ω está ordenada por inclusión, pues al ser B débilmente-* compacto, la intersección de subconjuntos de débilmente-* cerrados de B seguirá siendo débilmente-* cerrado y no vacío, mientras que la intersección de subconjuntos extremales de B es otro subconjunto extremal de B , siempre y cuando no sea vacía. Por tanto, como se ha mencionado, todo elemento de Ω contiene un elemento minimal, es decir, el subconjunto extremal más pequeño. Faltaría comprobar únicamente que además, ese subconjunto se reduce a un solo punto.

Probar esto último implicaría que dicho punto es un punto extremal de B y consecuentemente de A (Lema 1.9). Se prueba por contradicción.

Si $F \in \Omega$ es minimal y $x^*, y^* \in F$, con $x^* \neq y^*$, se toma $u \in X$ tal que $y^*(u) \neq x^*(u)$, por lo que, usando el Lema 1.10, el conjunto

$$E = \{z^* \in F : z^*(u) = \max[J_X(u)](F)\}$$

(que se sabe no vacío pues F es débilmente-* compacto) es un subconjunto extremal de F . Es fácil ver que $E \in \Omega$ a la vez que $E \subset F$, pues no puede ocurrir que $x^*, y^* \in E$. Por la minimalidad de F , se llega a contradicción. \square

Como consecuencia de este importante teorema, se llega al teorema de *Krein-Milman*, que da nombre a esta subsección. Se establece ahora, para la demostración, toda la notación que va a ser necesaria. Denotamos por $\overline{\text{conv}(A)}$ a la evolvente *convexo-cerrada* de A , que es el menor conjunto convexo y cerrado (tanto para la norma como para la topología débil, Teorema 4.11) de X que contiene a A . A su vez, si $B \subset X^*$, entonces $\overline{\text{conv}^*(B)}$ es la *envolvente convexa y débil-* cerrada*, que es el menor conjunto convexo y débil-* cerrado de X^* que contiene a B .

Teorema 4.13 (Krein-Milman). *Sea X un espacio normado. Entonces:*

- (1) *Si $A \subset X$ es un subconjunto débilmente compacto, entonces*

$$A \subseteq \overline{\text{conv}(ex(A))},$$

y, si A es convexo, entonces se tiene la igualdad.

(II) Si A es un subconjunto débilmente-* compacto de X^* , entonces

$$A \subseteq \overline{\text{conv}^*(\text{ex}(A))},$$

y, si A es convexo, entonces se da la igualdad.

Demostración. Comenzamos probando (II). Tenemos que el conjunto $B = \overline{\text{conv}^*(\text{ex}(A))}$ es convexo y débil-* cerrado en X^* . Si existiera $a^* \in A \setminus B$, el Teorema 4.10 nos da un elemento $x \in X$ tal que

$$\sup[J_X(x)](B) < a^*(x).$$

Por lo tanto, el conjunto $E = \{x^* \in A : x^*(x) = \max[J_X(x)](A)\}$ sería un subconjunto extremal, débilmente-* cerrado y además no vacío de A tal que $E \cap B = \emptyset$. En particular, $E \cap \text{ex}(A) = \emptyset$, contradiciendo esto el teorema anterior. Por tanto, no puede existir dicho a^* y ocurre que $A \subseteq B = \overline{\text{conv}(\text{ex}(A))}$.

La demostración de I es bastante más directa. Si A es débilmente compacto, entonces $J_X(A)$ será débilmente-* compacto en X^{**} , por lo que el apartado II nos dice que $J_X(A) \subseteq \overline{\text{conv}^*(\text{ex}(A))}$, por tanto si desarrollamos:

$$A = J_X(A) \cap X \subseteq \overline{\text{conv}^*(\text{ex}(A))} \cap X = \overline{\text{conv}(\text{ex}(A))}.$$

□

Este resultado no solo proporciona una caracterización estructural profunda de los conjuntos convexos, sino que también resalta la importancia de los puntos extremos interpretados como *generadores* del conjunto. La potencia del teorema radica en su aplicabilidad en la teoría de optimización, pues la comprensión de la geometría de conjuntos convexos es fundamental. Así, el teorema sirve como un puente entre propiedades geométricas y topológicas, consolidando la relevancia de los puntos extremos en el estudio de la convexidad.

4.3. Principio del máximo de Bauer

El principio del máximo de *Bauer* es un resultado fundamental en Análisis Funcional y teoría de la convexidad, que profundiza en la relación entre las funciones continuas y los puntos extremos de conjuntos convexos. En términos generales, el principio establece que existe un conjunto con determinadas características para el que toda función continua real sobre ese conjunto alcanza su máximo en al menos un punto extremo del mismo. Esta afirmación refuerza la idea, también presente en el teorema de *Krein–Milman*, de que los puntos extremos desempeñan un papel esencial en la estructura de los conjuntos convexos, actuando como *soporte* para diversas propiedades.

Definición 4.14. Sea X un espacio vectorial y $A \subset X$ convexo. Se dice que $f : A \rightarrow \mathbb{R}$ es *casi-convexa* si

$$f((1-t)x + ty) \leq \max\{f(x), f(y)\}, \quad x, y \in A, t \in [0, 1].$$

Se recuerda también al lector el concepto de *semicontinuidad*. Una función f es semicontinua superiormente (resp. inferiormente) en x_0 si $\forall \epsilon \in \mathbb{R}^+$, existe un entorno U de x_0 en X tal que $f(U) \subset (f(x_0) - \epsilon, +\infty)$ (resp. $(-\infty, f(x_0) + \epsilon)$).

4. Programación convexa

Será necesario, adicionalmente, hacer uso del siguiente lema de Topología elemental, recordando algunos conceptos básicos de paso:

Lema 4.15. *Si K es un espacio topológico T_2 (Hausdorff) compacto y $f : K \rightarrow \mathbb{R}$ es una función semicontinua superiormente, entonces f alcanza su máximo en K .*

El problema que se plantea es buscar aquellos puntos del conjunto K donde f alcanza su máximo y, haciendo uso del Teorema 4.13, podremos obtener esta información. Supondremos que K es un subconjunto convexo y que f es casi-convexa. Estamos pues, en condiciones de enunciar el Principio del máximo de Bauer como corolario.

Corolario 4.16 (Principio del máximo de Bauer). *Sea X un espacio normado, A un subconjunto no vacío convexo y débilmente compacto de X . Sea f una función semicontinua superiormente y casi-convexa. Entonces f alcanza su máximo en un punto extremo de A , es decir, existe $x_0 \in \text{ex}(A)$ tal que $f(x) \leq f(x_0), \forall x \in A$.*

Demostración. Utilizaremos como anticipábamos el Teorema de Krein-Milman. Para ello, se define el conjunto

$$E = \{x \in A : f(x) = \max f(A)\},$$

esto es, el conjunto de los elementos de A donde f alcanza sus máximos. Si comprobamos que el conjunto definido es un conjunto extremal de A , habremos terminado, pues E contendrá un punto extremo de A .

Supongamos que $x \in E$ y que $x = (1 - t)y + tz$, con $y, z \in A$ y $t \in [0, 1]$. Queremos probar que $y, z \in E$, es decir, que también alcanzan el máximo de f .

Dado que f es semicontinua superiormente, casi-convexa y $f(x) = \max f(A)$, se sigue que:

$$\max f(A) = f(x) \leq \max\{f(y), f(z)\} \leq \max f(A).$$

Por tanto debe cumplirse la igualdad

$$\max\{f(y), f(z)\} = \max f(A).$$

Ahora imaginemos que uno de los puntos alcanza el máximo y el otro no, pongamos $f(y) = \max f(A)$ y $f(z) < f(y)$. Sin embargo, se tenía que $x = (1 - t)y + tz$ y también que $f(x) = \max f(A)$. Esto implicaría que $\max f(A)$ es combinación convexa de $f(y)$ y de otro valor menor, $f(z)$, que sabemos que no es posible.

Esto último implica que ambos deben cumplir que

$$f(y) = f(z) = \max f(A) \implies y, z \in E.$$

□

Para terminar, podemos extender naturalmente a las funciones convexas:

Corolario 4.17 (Principio del máximo de Bauer para funciones convexas). *Toda función convexa y continua definida sobre un conjunto convexo y compacto alcanza su máximo en algún punto extremo del conjunto.*

El principio del máximo de Bauer, junto con el teorema de Krein–Milman, muestra cómo la geometría de los conjuntos convexos permite encontrar soluciones óptimas en puntos

particularmente relevantes, como son los puntos extremos. La utilidad de estos resultados reside en la posibilidad de abordar problemas de optimización en espacios de dimensión infinita, como por ejemplo, espacios de sucesiones o de funciones (ℓ_1, c_0, \dots).

Con esta última sección se da por finalizada la primera parte del trabajo. El desarrollo hecho muestra la importancia y la utilidad de la teoría de la **PL** y **PC**. En concreto, por su fácil implementación práctica y uso en escenarios reales, como se verá en la segunda parte de esta memoria. En particular, la parte lineal será la que evidenciará la utilidad del método Símplex. El desarrollo y puesta en práctica de la parte convexa, junto a estos últimos resultados, quedan como generalizaciones que podrían interesar utilizar en un trabajo futuro. Se comenta sobre esto en la Sección 11.

Parte II.

Aplicación práctica de la optimización en la privacidad en Aprendizaje Federado

5. El concepto de Aprendizaje

El **ML** y el **DL** son subcampos fundamentales de la **IA** que han transformado diversas disciplinas científicas y aplicaciones tecnológicas. Estas técnicas permiten a los sistemas computacionales mejorar su rendimiento en tareas específicas a través de la experiencia, sin depender de reglas programadas explícitamente [Mit97, LBH15]. La capacidad de extraer patrones significativos de grandes volúmenes de datos ha posicionado al **ML** como una herramienta clave en áreas como la visión por computador, el procesamiento del lenguaje natural, la biomedicina y la economía.

El aprendizaje no solo se limita al ámbito académico o científico, sino que también tiene aplicaciones prácticas en una amplia variedad de campos. Por ejemplo, en el sector de la salud, los algoritmos de aprendizaje automático se utilizan para predecir enfermedades, personalizar tratamientos y analizar imágenes médicas [GACW21]. En el ámbito financiero, se emplean para detectar fraudes, predecir tendencias del mercado y evaluar riesgos crediticios [NDZY21].

En el sector de la tecnología, el aprendizaje profundo ha permitido avances significativos en áreas como el reconocimiento de voz, la traducción automática y la conducción autónoma [SS18]. Además, en el comercio electrónico, los sistemas de recomendación basados en **ML** han transformado la experiencia del usuario al personalizar sugerencias de productos y servicios [LMY⁺12].

Otro caso de uso relevante es el análisis de datos en redes sociales, donde los modelos de aprendizaje automático ayudan a identificar patrones de comportamiento, detectar contenido inapropiado y mejorar la interacción con los usuarios [BVT⁺24].

Solo con estas aplicaciones, se pone en evidencia el impacto transformador de la **IA** y todas sus disciplinas en nuestra vida cotidiana y su potencial para resolver problemas complejos en diversos dominios.

5.1. Aprendizaje Automático

El **ML** se define como “el estudio de algoritmos que mejoran automáticamente a través de la experiencia” [Mit97]. Este campo se basa en el uso de datos y modelos matemáticos para realizar tareas como la clasificación, la regresión, el agrupamiento y la reducción de dimensionalidad para problemas específicos.

Los principales paradigmas del **ML** son:

- **Aprendizaje Supervisado:** Se basa en datos etiquetados, donde el modelo aprende a mapear entradas a salidas mediante un conjunto de ejemplos. Algoritmos populares incluyen regresión lineal, máquinas de soporte vectorial (SVM o *Support Vector Machine*) y árboles de decisión [HTFFo4].
- **Aprendizaje No Supervisado:** No requiere etiquetas y se enfoca en identificar patrones o estructuras ocultas en los datos, como en el caso del agrupamiento (k-means, DBSCAN) o la reducción de dimensionalidad (PCA) [Biso6].

5. El concepto de Aprendizaje

- **Aprendizaje por refuerzo:** Este paradigma se centra en la interacción de un agente con un entorno, donde el agente aprende a tomar decisiones secuenciales para maximizar una recompensa acumulada. A diferencia del aprendizaje supervisado, no se proporcionan etiquetas explícitas, sino que el agente recibe retroalimentación en forma de recompensas o penalizaciones. Algoritmos como Q-learning y Deep Q-Networks (DQN) han demostrado ser efectivos en tareas como juegos, robótica y control autónomo [DAHGHS18].

Un problema clásico en el aprendizaje automático es el de la **clasificación**, que consiste en asignar una etiqueta o categoría a una instancia de datos basada en sus características. Este problema se encuentra en el núcleo de muchas aplicaciones prácticas, como el reconocimiento de imágenes, la detección de spam en correos electrónicos y la predicción de enfermedades en el ámbito médico.

En el caso de la **clasificación multiclase**, el objetivo es asignar una de varias categorías posibles a cada instancia. Por ejemplo, en el reconocimiento de dígitos escritos a mano, el modelo debe clasificar cada imagen como un número del 0 al 9. Este tipo de problemas requiere algoritmos capaces de manejar múltiples clases y de aprender patrones complejos en los datos.

Entre los desafíos más comunes en los problemas de clasificación se encuentran:

- **Desequilibrio de clases:** Cuando algunas clases están representadas por un número significativamente menor de ejemplos en comparación con otras, el modelo puede sesgarse hacia las clases más frecuentes [BMM17].
- **Sobreajuste:** Si el modelo es demasiado complejo, puede ajustarse demasiado a los datos de entrenamiento y no generalizar bien a datos nuevos [Unk10].
- **Datos ruidosos o irrelevantes:** La presencia de características irrelevantes o ruido en los datos puede dificultar el aprendizaje del modelo [M⁺10, M⁺19].
- **Dimensionalidad alta:** En problemas con un gran número de características, el modelo puede enfrentar dificultades para identificar patrones relevantes sin un preprocesamiento adecuado [XJKo1].

El tercero de estos es el que más trataremos, pues en la experimentación se verá que se pretende introducir ruido como mecanismo de privacidad, intentando preservar la utilidad y efectividad del algoritmo.

En este trabajo, sin embargo, no se utilizarán técnicas de clasificación básicas del ámbito del **ML**. En su defecto, usaremos redes neuronales complejas, explicadas más adelante. Se partirá de un problema clásico de clasificación, dado que los conjuntos de datos más utilizados en la literatura suelen ser de este tipo. Como base comparativa para nuestra experimentación será más que suficiente. Esto además facilita la implementación y permite enfocar la investigación en el eje central del estudio, que es la privacidad de los datos.

5.2. Aprendizaje Profundo

El Aprendizaje Profundo o **DL** es una subdisciplina del **ML** que se centra en el uso de **redes neuronales artificiales** con múltiples capas para modelar y aprender representaciones jerárquicas de los datos. Este enfoque ha ganado popularidad en los últimos años debido a

su capacidad para resolver problemas complejos en áreas como la visión por computador, el procesamiento del lenguaje natural y los sistemas de recomendación.

Cuando se habla de redes neuronales profundas, se hace referencia a modelos que constan de varias capas ocultas entre la entrada y la salida. Estas capas permiten que el modelo aprenda características de alto nivel a partir de los datos brutos, lo que distingue al aprendizaje profundo de los métodos tradicionales de aprendizaje automático, que a menudo requieren ingeniería manual de características. Es la propia red la que se encarga de esta extracción de características.

Desde un punto de vista matemático, una red neuronal es una función parametrizada que transforma un vector de entrada $\mathbf{x} \in \mathbb{R}^n$ en una salida $\mathbf{y} \in \mathbb{R}^m$, mediante una serie de operaciones lineales y no lineales. Este elemento mencionado de no linealidad es lo que convierte a las redes neuronales en una herramienta tan poderosa, pues permite extraer conocimiento complejo y profundo que, de hecho, no siempre es interpretable a ojos de un ser humano. Formalmente, una red neuronal con L capas puede representarse como:

$$\mathbf{y} = f_L(f_{L-1}(\dots f_1(\mathbf{x}))),$$

donde cada capa $f_i, i \in \{1, \dots, L\}$ realiza una transformación de la forma:

$$f_i(\mathbf{z}) = \sigma(\mathbf{W}_i \mathbf{z} + \mathbf{b}_i),$$

con \mathbf{W}_i y \mathbf{b}_i representando los pesos y sesgos de la capa i , respectivamente, y σ siendo una función de activación no lineal, como la ReLU (Rectified Linear Unit) o la sigmoide, que se discutirán más adelante.

Los elementos principales de una red neuronal son:

- **Capas de entrada y salida:** La capa de entrada recibe los datos brutos, mientras que la capa de salida produce las predicciones del modelo.
- **Capas ocultas:** Estas capas intermedias procesan y transforman los datos, aprendiendo representaciones jerárquicas.
- **Pesos y sesgos:** Los parámetros ajustables o entrenables del modelo que se optimizan durante el entrenamiento para minimizar una función de pérdida.
- **Función de activación:** Introduce no linealidad en el modelo, permitiendo que la red aprenda relaciones complejas en los datos.
- **Función de pérdida:** Mide el error entre las predicciones del modelo y las etiquetas reales, guiando el proceso de optimización.

Introduciremos y explicaremos brevemente todos estos elementos (haciendo hincapié en los que se usarán en este trabajo), así como en qué consiste el entrenamiento de una red como esta. En este trabajo, se explorarán las capacidades del aprendizaje profundo en el contexto de problemas de clasificación, destacando su potencial para extraer patrones complejos y su aplicabilidad en escenarios donde la privacidad de los datos es la preocupación central.

Funciones de activación

Las **funciones de activación** son componentes esenciales en las redes neuronales, ya que introducen no linealidad en el modelo. Sin ellas, la red neuronal no sería capaz de aprender

5. El concepto de Aprendizaje

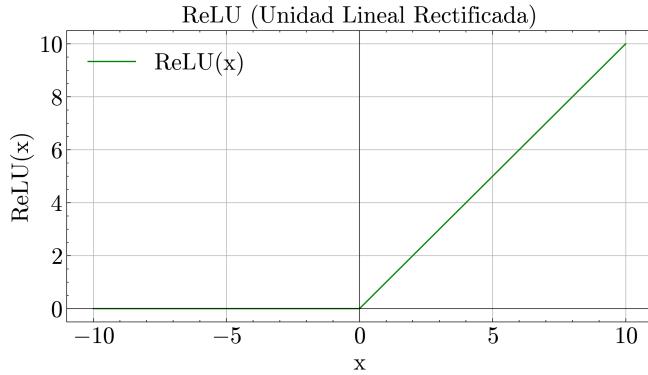


Figura 5.1.: Gráfica de la función de activación ReLU. Para valores de entrada negativos el resultado es 0. Para aquellos mayores que 0, su imagen es la identidad.

representaciones complejas y, en consecuencia, no podría resolver tareas que impliquen patrones no lineales. En términos generales, las funciones de activación transforman la salida de cada neurona antes de pasarla a la siguiente capa.

Matemáticamente, una función de activación $\phi(\cdot)$ se aplica sobre la salida lineal de una neurona, que es el producto de los pesos y las entradas más un sesgo:

$$\mathbf{h} = \phi(\mathbf{Wx} + \mathbf{b}),$$

donde \mathbf{x} es el vector de entradas, \mathbf{W} es el vector de pesos, \mathbf{b} es el sesgo y \mathbf{h} es la salida de la neurona.

Existen varias funciones de activación comunes, cada una con propiedades y aplicaciones específicas. Una de las más conocidas es la **ReLU (Rectified Linear Unit)**. La función ReLU es una de las más utilizadas debido a su simplicidad y eficiencia computacional. Se define como:

$$\text{ReLU}(x) = \max(0, x),$$

lo que significa que cualquier valor negativo se reemplaza por cero, mientras que los valores positivos permanecen sin cambios, ver Figura 5.1. La principal ventaja de ReLU es que reduce el problema del desvanecimiento del gradiente (*vanishing gradient*¹), aunque puede sufrir del problema de "*neuronas muertas*", donde algunas neuronas dejan de aprender si sus salidas siempre son negativas.

Otras de las funciones de activación más utilizadas en el **DL** son la **sigmoide** o la **tangente hiperbólica**. Cada una de ellas tiene características específicas que las hacen apropiadas para distintas tareas.

Es por esto que la elección de la función de activación depende del tipo de tarea y la arquitectura de la red. Mientras que funciones como ReLU y sus variantes han demostrado ser muy efectivas en redes profundas, funciones como la sigmoide y la tangente hiperbólica aún tienen aplicaciones en redes más simples y en tareas específicas, como la clasificación bi-

¹El *vanishing gradient* es un fenómeno que ocurre cuando, durante la retropropagación o *back propagation*, los gradientes de los errores con respecto a los pesos disminuyen exponencialmente a medida que se retropropagan a través de las capas de la red, lo que hace que las actualizaciones de los pesos en las primeras capas sean muy pequeñas y lentas, afectando negativamente el proceso de aprendizaje. Suele ocurrir con la sigmoide o con la tangente hiperbólica como funciones de activación.

naria. La correcta selección de la función de activación es crucial para asegurar una adecuada propagación del gradiente y un entrenamiento eficiente.

Capas ocultas

Las **capas ocultas** son un componente fundamental en la arquitectura de una red neuronal artificial. Se definen como aquellas capas intermedias situadas entre la capa de entrada (donde entran los datos) y la capa de salida, y son las responsables de la transformación progresiva de los datos a lo largo de la red. Cada capa oculta está formada por un conjunto de neuronas, y cada una de ellas aplica una transformación lineal a sus entradas seguida de una función de activación no lineal (explicadas en la siguiente subsección).

Formalmente, una capa oculta puede representarse como:

$$\mathbf{h}^{(i)} = \phi\left(\mathbf{W}^{(i)}\mathbf{h}^{(i-1)} + \mathbf{b}^{(i)}\right),$$

donde $\mathbf{h}^{(i-1)}$ es la salida de la capa anterior, $\mathbf{W}^{(i)}$ y $\mathbf{b}^{(i)}$ son los pesos y sesgos de la capa i -ésima, respectivamente, y $\phi(\cdot)$ es la función de activación.

El papel de las capas ocultas es extraer y componer representaciones cada vez más abstractas y complejas de los datos de entrada. El número de neuronas y capas ocultas es un parámetro de diseño importante, que afecta directamente a la capacidad de generalización del modelo. Mientras que redes con pocas capas pueden ser insuficientes para capturar la complejidad de ciertos problemas, un número excesivo puede conducir a sobreajuste si no se aplican técnicas adecuadas de regularización.

Funciones de pérdida

Las funciones de pérdida desempeñan un papel central en el entrenamiento de redes neuronales, ya que permiten cuantificar el error cometido por el modelo al comparar sus predicciones con las salidas reales. Durante el aprendizaje, el objetivo es minimizar esta función con respecto a los parámetros del modelo, para mejorar la precisión de sus predicciones. Dependiendo del tipo de problema que se aborde, se deberá usar una función acorde.

Formalmente, sea $\hat{\mathbf{y}} \in \mathbb{R}^K$ el vector de salida de la red (puede estar normalizado, por ejemplo mediante softmax), e $\mathbf{y} \in \{0,1\}^K$ el vector objetivo codificado en formato *one-hot*², donde K es el número de clases. La función de pérdida se diseña para penalizar discrepancias entre $\hat{\mathbf{y}}$ e \mathbf{y} , de forma que una mayor distancia implique un mayor error. Para problemas de clasificación multiclase, que son los que trabajaremos, la función de pérdida más comúnmente utilizada es la *entropía cruzada*, definida como:

$$\mathcal{L}(\hat{\mathbf{y}}, \mathbf{y}) = - \sum_{k=1}^K y_k \log(\hat{y}_k),$$

donde \hat{y}_k representa la probabilidad estimada por el modelo para la clase k , y $y_k \in \{0,1\}$ es el valor real asociado a dicha clase.

En términos generales, cualquier función que permita cuantificar de manera coherente la discrepancia entre las predicciones generadas por el modelo y los valores reales asociados

²Una codificación *one-hot* representa una clase entre K posibles como un vector binario de longitud K , donde exactamente una componente toma el valor 1 (indicando la clase activa) y las demás son 0. Por ejemplo, si $K = 4$ y la clase verdadera es la número 2, el vector sería $[0, 1, 0, 0]$.

5. El concepto de Aprendizaje

a las muestras del conjunto de entrenamiento puede considerarse una función de pérdida válida para el proceso de optimización.

5.2.1. Entrenamiento de la red

Una vez definida la arquitectura de la red neuronal, incluyendo sus capas ocultas, funciones de activación y función de pérdida, el siguiente paso fundamental es el proceso de entrenamiento. El objetivo principal del entrenamiento es ajustar los parámetros internos de la red (pesos y sesgos) de modo que la salida producida por el modelo se aproxime lo más posible a la salida esperada, minimizando así la función de pérdida. El entrenamiento de los pesos de la red es un proceso puramente algorítmico y sigue los siguientes pasos:

1. El entrenamiento comienza con la **propagación hacia adelante** o (*forward propagation*), donde se introduce un conjunto de entrada \mathbf{x} en la red. Cada neurona aplica su función de activación a una combinación lineal de sus entradas, generando así una salida que se propaga a la siguiente capa. Este proceso continúa hasta la capa de salida, donde se obtiene la predicción final $\hat{\mathbf{y}}$.
2. A partir de la predicción $\hat{\mathbf{y}}$ y del valor real \mathbf{y} , se calcula el error mediante la función de pérdida $\mathcal{L}(\hat{\mathbf{y}}, \mathbf{y})$.
3. Una vez calculado el error, se procede a la fase de **retropropagación** o *backpropagation*. Este algoritmo aplica la regla de la cadena para calcular el gradiente de la función de pérdida con respecto a cada parámetro de la red. En otras palabras, se determina cómo pequeños cambios en los pesos afectan el valor de la pérdida.

El objetivo es computar el gradiente $\nabla_{\theta}\mathcal{L}$, donde θ representa el conjunto de todos los parámetros de la red. Este cálculo se realiza propagando los errores hacia atrás, comenzando en la capa de salida. Se define el error en la capa i como:

$$\delta^{(i)} = \frac{\partial \mathcal{L}}{\partial \mathbf{z}^{(i)}} = \left(\frac{\partial \mathcal{L}}{\partial \mathbf{a}^{(i)}} \right) \circ \phi'(\mathbf{z}^{(i)}),$$

donde \circ denota el producto elemento a elemento, $\mathbf{a}^{(i)}$ es el vector de activaciones³ de la capa i y $\mathbf{z}^{(i)}$ es el vector de entradas de la capa i . Luego, se actualizan los gradientes:

$$\frac{\partial \mathcal{L}}{\partial \mathbf{W}^{(i)}} = \delta^{(i)} \cdot (\mathbf{a}^{(i-1)})^\top, \quad \frac{\partial \mathcal{L}}{\partial \mathbf{b}^{(i)}} = \delta^{(i)}.$$

Para propagar el error hacia capas anteriores, se utiliza la siguiente relación recursiva:

$$\delta^{(i-1)} = ((\mathbf{W}^{(i)})^\top \delta^{(i)}) \circ \phi'(\mathbf{z}^{(i-1)}).$$

4. Se parte del gradiente anterior, el cual proporciona la dirección en la que deben ajustarse los parámetros para minimizar el error. Entonces se actualizan los parámetros de la red mediante un algoritmo de optimización. El más común es el descenso por el gradiente (*gradient descent*), en el cual los parámetros se modifican según la siguiente regla:

³Nos referimos por vector de activaciones a la salida resultado de aplicar la función de activación en dicha capa.

$$\theta \leftarrow \theta - \eta \nabla_{\theta} \mathcal{L},$$

donde a $\eta > 0$ se le llama tasa de aprendizaje (*learning rate*). Este procedimiento se repite iterativamente para un número determinado de épocas, utilizando todo el conjunto de datos, hasta alcanzar una convergencia adecuada.

5. Generalización y sobreajuste: Durante el entrenamiento, es fundamental evitar el sobreajuste (*overfitting*), fenómeno por el cual la red se ajusta demasiado a los datos de entrenamiento y pierde capacidad de generalización a datos no vistos. Para mitigar este problema, se suelen emplear técnicas como la regularización, el uso de conjuntos de validación y el *early stopping*.

En resumen, el entrenamiento de una red neuronal consiste en un proceso iterativo que ajusta sus parámetros internos mediante la retropropagación del error y técnicas de optimización, con el objetivo de minimizar la función de pérdida y mejorar el rendimiento del modelo en tareas de predicción.

5.2.2. Redes y capas convolucionales

En la experimentación de este trabajo, se usarán conjuntos de datos de imágenes, pues suele ser un estándar en escenarios de investigación y además aporta un elemento visual a los experimentos, que ayuda a entenderlos.

Las imágenes digitales, que se representan y almacenan como una matriz de píxeles. Cada celda de la matriz contiene el valor del color a representar, con unas intensidades. Es aquí donde surge el principal problema de las redes neuronales; su complejidad, pues cada conexión representa un peso a entrenar, y en una red neuronal artificial, cuantos más pesos se hayan de entrenar, más costoso resultará su entrenamiento. La idea general por la que surgen las redes convolucionales es esta idea, pues se pretenden cubrir problemas en los que los datos se presentan en forma de matriz y se ha de tratar con muchos pesos. El uso de filtros y la reducción de la dimensionalidad conforme se avanza en la red que explicaremos ahora, hace que el número de pesos total, se vea altamente reducido.

De esta manera, se consigue que el número de operaciones sea mucho menor (lo que se conoce como conectividad dispersa) y que el número de parámetros a aprender también se reduzca (conocido como *weight sharing*). El paper original de Yann LeCun en [LBBH98], publicado en el año 1998 habla de estos conceptos, entre otros, y forma las bases sobre las innovadoras redes convolucionales.

Los elementos clave que se han de comprender para poder utilizar las redes neuronales convolucionales son:

- **Filtros:** En una red neuronal convolucional (RNC), un **filtro** (o *kernel*) es un tensor de pesos (entrenables) que se utiliza para extraer características locales de una entrada multidimensional, típicamente una imagen. Formalmente, un filtro se representa como un tensor de dimensiones $k_h \times k_w \times C_{in}$, donde k_h y k_w corresponden a la altura y el ancho del filtro, respectivamente, y C_{in} representa el número de canales de entrada, o el número de filtros a aprender.
- **Operación de convolución y capa convolucional:** Durante la operación de convolución, el filtro se desliza sobre la entrada realizando una operación de producto punto (celda a

$$\begin{array}{|c|c|c|c|c|} \hline 2 & 4 & 9 & 1 & 4 \\ \hline 2 & 1 & 4 & 4 & 6 \\ \hline 1 & 1 & 2 & 9 & 2 \\ \hline 7 & 3 & 5 & 1 & 3 \\ \hline 2 & 3 & 4 & 8 & 5 \\ \hline \end{array} \otimes \begin{array}{|c|c|c|} \hline 1 & 2 & 3 \\ \hline -4 & 7 & 4 \\ \hline 2 & -5 & 1 \\ \hline \end{array} = \begin{array}{|c|c|c|} \hline 51 & & \\ \hline & & \\ \hline & & \\ \hline \end{array}$$

K H

Figura 5.2.: Filtro 3×3 sobre una imagen de dimensión 5×5 , mostrando el resultado de la operación de convolución para la primera celda. ($k_h = 3, k_w = 3, C_{in} = 1$)

celda) entre sus valores y los valores locales de la entrada en cada posición. El resultado de esta operación en cada posición es un único valor escalar, y al recorrer toda la entrada se obtiene un nuevo mapa bidimensional denominado *mapa de activación (feature map)*, que representa la presencia de ciertos patrones aprendidos en distintas regiones de la entrada. Visualmente, en la Figura 5.2, podemos ver el resultado de aplicar el filtro en una de las celdas.

Formalmente, sea $\mathbf{X} \in \mathbb{R}^{H \times W}$ una entrada bidimensional (por ejemplo, una imagen de una sola canal), y sea $\mathbf{K} \in \mathbb{R}^{k_h \times k_w}$ un filtro o *kernel*. La convolución de \mathbf{X} con \mathbf{K} , denotada por $\mathbf{X} * \mathbf{K}$, está definida como:

$$(\mathbf{X} * \mathbf{K})_{i,j} = \sum_{m=0}^{k_h-1} \sum_{n=0}^{k_w-1} \mathbf{X}_{i+m, j+n} \cdot \mathbf{K}_{m,n}$$

donde (i, j) recorre las posiciones válidas de la salida convolucional. Si la entrada tiene múltiples canales, la operación se extiende sumando sobre todos los canales y utilizando un filtro por cada canal:

$$(\mathbf{X} * \mathbf{K})_{i,j} = \sum_{c=1}^{C_{in}} \sum_{m=0}^{k_h-1} \sum_{n=0}^{k_w-1} \mathbf{X}_{i+m, j+n, c} \cdot \mathbf{K}_{m,n,c}$$

Esta operación se puede generalizar a convoluciones con *stride* y *padding*, controlando la cantidad de desplazamiento del filtro y la forma de tratar los bordes, respectivamente.

Desde una perspectiva funcional, cada filtro actúa como un detector de patrones espaciales específicos, tales como bordes, texturas o esquinas. Gracias al principio de *weight sharing*, el mismo conjunto de pesos del filtro se aplica a toda la entrada, lo cual no solo reduce significativamente el número de parámetros, sino que también introduce invariancia espacial en la red.

En una **capa convolucional** suelen utilizarse múltiples filtros simultáneamente, cada uno generando su propio mapa de activación. Estos mapas se apilan y sirven como entrada a las siguientes capas, lo que permite construir representaciones jerárquicas de la información contenida en los datos de entrada. Son estos pesos generados para

cada filtro los pesos que entrenará la red, usando el algoritmo discutido en la anterior sección.

- **Capas de reducción de dimensionalidad o *downsampling*:** Estas capas se utilizan para reducir la resolución espacial de los mapas de activación generados por las capas convolucionales, manteniendo las características más relevantes y disminuyendo la complejidad computacional de la red. La operación más común para este propósito es la de *max pooling*, que consiste en dividir el mapa de activación en regiones no superpuestas (por ejemplo, de tamaño 2×2) y tomar el valor máximo dentro de cada región. Este proceso introduce cierta invariancia traslacional y ayuda a evitar el sobreajuste al reducir el número de parámetros y la sensibilidad a pequeñas variaciones espaciales.
- **Capas totalmente conectadas o *fully connected*:** Estas capas aparecen comúnmente en la parte final de una red neuronal convolucional, una vez que las representaciones espaciales han sido suficientemente procesadas y condensadas por las capas anteriores. En una capa totalmente conectada, cada neurona está conectada con todas las salidas de la capa anterior. Formalmente, la operación se expresa como una combinación lineal de las entradas seguida de una función de activación: $\mathbf{a}^{(l)} = \phi(\mathbf{W}^{(l)} \mathbf{a}^{(l-1)} + \mathbf{b}^{(l)})$, donde $\mathbf{W}^{(l)}$ representa la matriz de pesos, $\mathbf{b}^{(l)}$ el vector de sesgos, y ϕ la función de activación. Estas capas permiten realizar tareas de clasificación o regresión a partir de las características extraídas previamente. Es en estas últimas capas donde se acumula el mayor número de pesos y dota a la red de una gran complejidad y poder de predicción.

Cualquier red que tenga, al menos, una capa convolucional, se dice que es una red neuronal convolucional. Los pesos a aprender serán los filtros usados en las convoluciones. Esto implica que, en lugar de aprender un peso por cada conexión entre píxeles de entrada y neuronas, como ocurre en las redes totalmente conectadas (*fully connected*), una red convolucional aprende un conjunto reducido de filtros que se aplican de manera local y repetida sobre toda la imagen.

Este enfoque ha demostrado ser especialmente eficaz en tareas de visión por computador como la clasificación de imágenes, la detección de objetos y el reconocimiento facial. A lo largo de los últimos años, múltiples arquitecturas basadas en convoluciones -como LeNet (ver Figura 5.3, AlexNet, VGG o ResNet- han establecido nuevos estándares en precisión y rendimiento en *benchmarks* internacionales, consolidando las redes convolucionales como una de las herramientas fundamentales en el aprendizaje profundo.

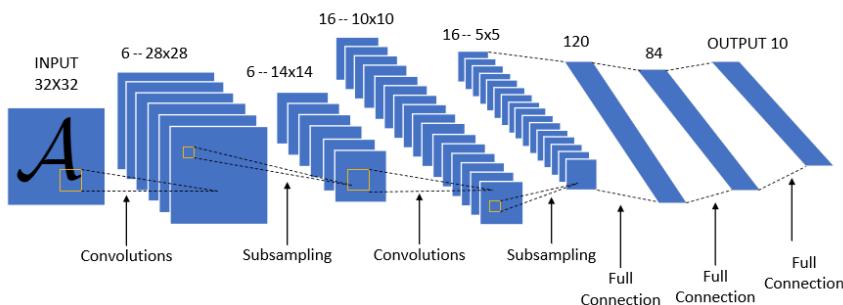


Figura 5.3.: Arquitectura de LeNet5 para reconocimiento de caracteres manuscritos, presentada en [LBBH98], con dos capas convolucionales.

6. Aprendizaje Federado

En los últimos años, el crecimiento exponencial en la generación de datos y la adopción masiva de dispositivos inteligentes ha planteado nuevos desafíos en cuanto a la privacidad, la seguridad y la eficiencia del **ML**. Como se comenta en la introducción del trabajo, en esta línea surgen los escenarios de riesgo propuestos en [Eur24] o el reglamento GDPR (*General Data Protection Regulation*) [God17], en vigor desde el año 2018.

En este contexto, el Aprendizaje Federado o **FL** ha emergido como un paradigma innovador que permite entrenar modelos de **ML** de forma colaborativa entre múltiples clientes o nodos, sin necesidad de centralizar sus datos. Esta aproximación resulta especialmente útil en escenarios sensibles, como los entornos médicos, financieros o móviles, donde la transferencia de información personal a un servidor central puede ser inviable o incluso ilegal.

Como se hace en [RBJLL⁺23], el **FL** se propone como una arquitectura descentralizada en la que cada cliente entrena localmente una copia del modelo utilizando sus propios datos [LSTS20], y únicamente comparte con el servidor central los parámetros o gradientes actualizados. A través de un proceso iterativo de agregación, el servidor actualiza el modelo global sin tener acceso directo a los datos originales. Esta metodología preserva la privacidad de los usuarios al tiempo que permite el aprovechamiento colectivo del conocimiento distribuido. Repasaremos los principales elementos y el funcionamiento de este paradigma.

El **FL** es un paradigma de **ML** propuesto en respuesta a tres desafíos principales:

- **Preservación de la privacidad:** Los datos más relevantes y de mayor riqueza en cuanto a posible aprendizaje suelen ser de una gran sensibilidad [WYJ⁺22]. Consecuentemente, urge que se preserve en todo momento la privacidad del usuario.
- **Costes de comunicación y latencia:** En el enfoque de **ML** centralizado, los datos sin procesar son enviados a un servidor central para su procesamiento y posterior uso en el entrenamiento del modelo. Este intercambio de información puede implicar un alto coste computacional y de comunicación [MMR⁺17b], especialmente cuando se trabaja con conjuntos de datos de gran volumen.
- **Acceso a los datos,** [ABHKS17]: El **ML** aborda esta problemática mediante la implementación de enfoques colaborativos que permiten el entrenamiento de modelos a partir de datos distribuidos entre diversas instituciones y organizaciones. Este enfoque no solo favorece el desarrollo de modelos más robustos y generalizables, sino que también garantiza la preservación de la privacidad y la seguridad de la información al evitar la necesidad de compartir los datos de manera directa.

El primer uso exitoso de este paradigma fue de la mano de la gigantesca tecnológica *Google*, que utilizó los datos de miles de dispositivos *Android*, manteniendo los datos de manera local en los dispositivos y manteniendo así la privacidad de los mismos [KMY⁺16]. Desde entonces, el **FL** se ha aplicado a una gran variedad de campos con distintas aplicaciones, como puede ser la medicina, con la denominada *Internet of Health-care Things* (*ioHT*) [CNV⁺23, CLL⁺23]. De esta manera, la combinación de conocimiento de distintas áreas o partes del escenario de **FL** se usa para determinar de una forma más precisa el estado de salud de un

6. Aprendizaje Federado

paciente, para tomar acciones más rápidamente. Es curioso saber que también se usó en la detección del COVID-19 [NPC22].

Para más contexto, también es notable la utilidad que ha tenido el FL en el ámbito industrial, como por ejemplo detectar defectos en tareas de producción [HYG19] o detección de ataques maliciosos en sistemas de comunicación de vehículos aéreos sin tripulación humana [MTDC19].

Se expone a continuación y en la siguiente sección un marco teórico de definición del paradigma central explicado en este TFG, con objeto de poder formalizar la teoría acerca de la materia.

Un escenario de FL consiste en una red de clientes C_1, \dots, C_n que participan en dos procesos principales:

1. **Fase de entrenamiento del modelo:** Cada uno de los clientes intercambia información de manera anónima, esto es, sin compartir de manera explícita datos de entrenamiento. Estos datos se usan para entrenar un modelo \mathcal{M}_f , que puede residir en un cliente o ser compartido por varios de ellos.
2. **Fase de inferencia:** Los clientes aplican de manera colaborativa el modelo entrenado \mathcal{M}_f a una nueva instancia de los datos.

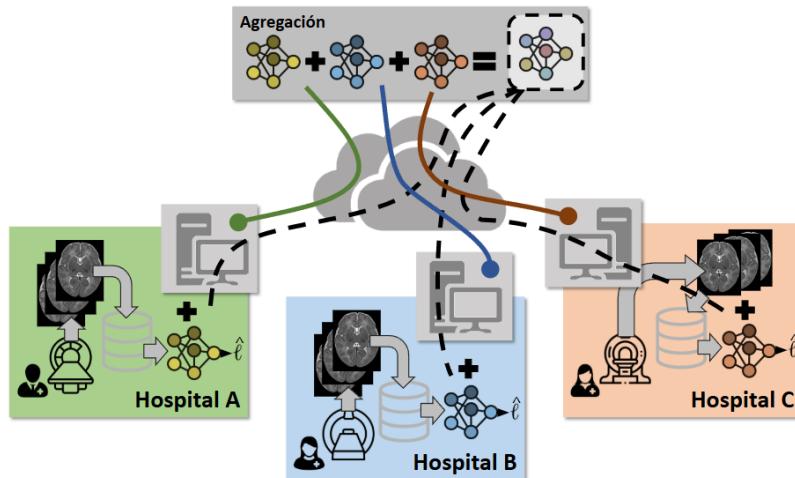


Figura 6.1.: Caso de uso genérico donde datos de resonancias magnéticas son usados para entrenar tres modelos locales de tres hospitales distintos, para luego ser agregados y obtener un modelo final. Fuente de la imagen: [LRBAG⁺24].

Estos procesos pueden ser tanto síncronos como asíncronos, dependiendo de la disponibilidad de los datos y del modelo entrenado. Es relevante tener en cuenta que no solo se está tratando la privacidad de esta manera, sino que además debe haber un sistema de justa repartición para compartir los beneficios obtenidos en el proceso colaborativo de entrenamiento.

6.1. Formalización del Aprendizaje Federado

Una vez introducido el **FL** desde un punto de vista general, es posible formalizar de manera teórica un escenario del mismo, con todos sus componentes y características generales, como se hace en [LRBAG⁺²⁴].

Se considera un conjunto de clientes o nodos $\{C_1, \dots, C_n\}$, cada uno con su respectivo conjunto de datos de entrenamiento local $\{D_1, \dots, D_n\}$ (recordemos que estos nunca se comparten). Cada cliente C_i posee un modelo de aprendizaje local L_i , expresado mediante los parámetros $\{L_1, \dots, L_n\}$. FL tiene como objetivo aprender un modelo global G utilizando los datos de todos los clientes, a través de un proceso de aprendizaje iterativo conocido como *ronda de entrenamiento*.

En particular, durante cada ronda de entrenamiento t , cada cliente entrena su modelo local sobre su conjunto de datos local D_i^t , lo que da lugar a una actualización de sus parámetros locales de L_i^t a \hat{L}_i^t . Posteriormente, se calculan los parámetros globales G^t agregando los parámetros locales entrenados $\{\hat{L}_1^t, \dots, \hat{L}_n^t\}$ mediante un operador de agregación Δ , que definiremos más tarde. Finalmente, los modelos locales se actualizan con los parámetros globales agregados:

$$G^t = \Delta(\hat{L}_1^t, \hat{L}_2^t, \dots, \hat{L}_n^t), \quad L_i^{t+1} \leftarrow G^t, \quad \forall i \in \{1, \dots, n\}.$$

Las actualizaciones entre los clientes y el servidor se repiten hasta alcanzar un criterio de parada definido. De este modo, el modelo final G recoge el conocimiento aprendido por todos los clientes, habiendo usado todos los datos para ello sin haberlos comprometido.

La motivación principal del diseño de este paradigma de aprendizaje distribuido permite entrenar modelos sobre datos que no pueden ser recolectados o centralizados fácilmente, proporcionando así una solución a los desafíos importantes comentados en la sección anterior:

- **Privacidad de los datos:** El paradigma aborda la posible fuga de datos permitiendo que el modelo sea entrenado donde residen los datos, sin compartir ninguna información sensible con un servidor central. De esta forma, los datos permanecen seguros en los dispositivos de los usuarios en todo momento.
- **Costes de comunicación y latencia:** Mitiga estos problemas permitiendo que solo se intercambien las actualizaciones del modelo, en lugar de los propios datos. Este enfoque acaba con los problemas de latencia y altos anchos de banda con los que tienen que lidiar otros tipos de procesos de entrenamiento distribuidos. Es un paradigma más escalable y eficiente.
- **Acceso a los datos:** Se resuelve este problema mediante la implementación de un entrenamiento colaborativo. Este enfoque colectivo relaja la necesidad de disponer directamente de los datos para entrenar un modelo de aprendizaje profundo. Se elimina también la necesidad de un servidor central, haciendo los datos sensibles más seguros.

En resumen, el paradigma del **FL** presenta una solución robusta a los desafíos asociados con la centralización de datos. Su enfoque en la privacidad de los datos asegura que la información sensible nunca se comparta, sino que el entrenamiento se realiza directamente en los dispositivos de los usuarios. Además, la reducción de costes de comunicación y latencia, al intercambiar únicamente actualizaciones del modelo, lo convierte en una alternativa más eficiente y escalable frente a enfoques tradicionales. Finalmente, la colaboración entre dispositivos elimina la necesidad de centralizar los datos, proporcionando un acceso más flexible

6. Aprendizaje Federado

y seguro a los mismos. Este enfoque tiene el potencial de transformar cómo se entrena los modelos de aprendizaje profundo en entornos donde la privacidad y la eficiencia son consideraciones clave.

6.2. Flujo de trabajo en Aprendizaje Federado

Una vez se ha introducido teóricamente el concepto de FL, se procede a describir el flujo de trabajo principal de un proceso de aprendizaje en estas condiciones. En la Figura 6.2 se presentan las diferentes etapas que componen el entrenamiento. A continuación, se detallan dichas etapas y se especifican los elementos clave que emergen en cada una de ellas. Estos pasos se detallan a fondo en [BSJ⁺20].



Figura 6.2.: Flujo y pasos que componen el aprendizaje en un escenario de FL. Imagen inspirada en [LRBAG⁺24].

- **Entrenamiento Local:** El proceso comienza con el entrenamiento local de cada uno de los modelos de ML en los distintos nodos o clientes propietarios de datos. Generalmente, estos modelos comparten una misma arquitectura, aunque los parámetros de entrenamiento, como el número de épocas, el tamaño del lote y la tasa de aprendizaje, pueden variar entre clientes. En esta fase surgen los primeros elementos clave del proceso:

- **Datos Descentralizados:** A diferencia de los enfoques centralizados, en FL los datos permanecen distribuidos en distintos dispositivos o nodos, lo que resulta especialmente beneficioso cuando la privacidad y seguridad de la información son una preocupación. Además, estos datos no se comparten con terceros ni son accesibles externamente. La distribución de los datos entre los clientes puede clasificarse en dos categorías principales:

- (i) Distribución Homogénea o Independiente e Idénticamente Distribuida (*IID*): En este caso, se asume que la distribución de los datos entre los clientes es *IID*, lo que implica que todos los clientes poseen datos con una distribución subyacente similar.
- (ii) Distribución Heterogénea o No Independiente e Idénticamente Distribuida (*non-IID*): En este escenario, los datos de cada cliente presentan distribuciones diferentes. Formalmente, se pueden distinguir tres tipos de heterogeneidad en la distribución de datos, según se describe en [CCP²²]:
 1. Diferencias en el espacio de características entre los clientes, aunque comparten un mismo objetivo común de aprendizaje.
 2. Espacio de entrada similar, pero con diferencias en el espacio de etiquetas dependiendo de la naturaleza de los datos.
 3. Diferencias tanto en el espacio de características como en el espacio de etiquetas entre los distintos clientes.
- **Modelo de entrenamiento:** El entrenamiento del modelo se lleva a cabo haciendo uso de los datos descentralizados, donde cada dispositivo cliente entrena su propio modelo y contribuye activamente al proceso de entrenamiento, compartiendo sus parámetros entrenados. De manera *simbótica*, el modelo también es capaz de hacer predicciones de mayor calidad pues posee una mayor capacidad de generalización al haber sido entrenado en una amplia variedad de datos.
- **Clientes:** Son los mencionados nodos donde reside el modelo a entrenar, así como los datos individuales de cada uno de ellos.
- **Comunicación:** Existe también un factor de *comunicación* importante, que permite la coordinación y agregación de las actualizaciones de los modelos. Es clave que, en este paso, se introduzcan técnicas de privacidad en los datos como la que es el eje central de este trabajo, la privacidad diferencial.
 - **Esquema de comunicación:** Puede ser tanto síncrona como asíncrona, dependiendo de la configuración y disponibilidad de los clientes y sus datos. También puede existir un servidor central que se encargue de la recolección de los modelos locales, o puede ser que esta recolección se haga de manera distribuida a lo largo de múltiples nodos.
 - **Protocolos de privacidad:** Como se ha comentado con anterioridad, los datos nunca son compartidos directamente en las comunicaciones en aprendizaje federado. Sin embargo, esto no lo hace *per sé* totalmente fiable [RBJLL²³]. Por tanto, es importante que se introduzcan técnicas de privacidad en este paso del flujo de trabajo.
- **Agregación:** Los cambios generados por los clientes en sus modelos locales se combinan mediante un operador de agregación, incorporando el resultado al modelo final entrenado. No es trivial la implementación del operador de agregación mencionado, pues depende en gran medida de la tarea que se esté tratando de abordar. El más común de los operadores usados es el conocido *Federated Averaging* (*FedAvg*), el cual se describe en [MMR^{17b}].
- **Actualización local:** El último paso consiste en actualizar los modelos locales almacenados en los diferentes nodos con el nuevo modelo global. El caso más simple es

6. Aprendizaje Federado

actualizar todos los modelos locales con este nuevo modelo global. Sin embargo, existen diferentes estrategias de actualización que consisten en combinar los modelos locales y globales en lugar de reemplazarlos directamente. Estos enfoques se utilizan para lograr características como la personalización de los clientes a sus datos locales.

6.3. Arquitecturas de aprendizaje

Todos los elementos descritos anteriormente por sí solos no proporcionan ninguna utilidad. Es la colaboración entre ellos y la comunicación entre las partes la que hace que el proceso de Aprendizaje tenga sentido. Por ello, si partimos de los elementos descritos y explicados en la sección anterior, podemos diferenciar claramente dos maneras de relacionarlos entre ellos (modelos clásicos de diseño de *software*). Estas son:

- **Arquitectura cliente-servidor:** Existe un nodo maestro o *servidor* que se encarga de la coordinación y agregación de las actualizaciones del modelo y el resto de nodos (donde se alojan sus respectivos datos) son los responsables de entrenar de manera local sus modelos. Esto últimos serán los *clientes*. Pese a ser una estructura bien conocida, usada y fácil de implementar, requiere un gran nivel de confianza de parte de los clientes hacia el servidor. Este es su principal debilidad. Podemos ver esta arquitectura representada en la Figura 6.3.

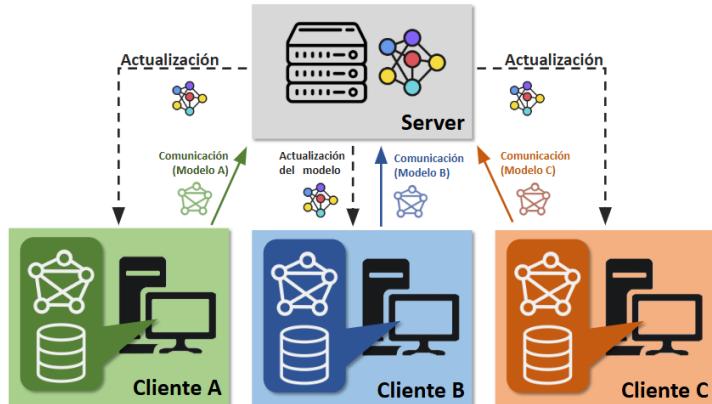


Figura 6.3.: Estructura de cliente-servidor en un escenario de Aprendizaje Federado. Concretamente, uno con tres nodos o clientes. Fuente de la imagen: [LRBAG⁺24].

- **Arquitectura peer to peer:** En este escenario, todos y cada uno de los nodos poseen los datos a entrenar y además se encargan de la agregación de las actualizaciones de los modelos. De esta manera, la figura de maestro o coordinador se convierte prescindible. Es más complicado de implementar que la arquitectura cliente-servidor pero como contrapartida ofrece la ventaja de que es más seguro y mantiene mayor privacidad.

En la mayoría de la literatura, así como en la mayoría de los escenarios reales de Aprendizaje Federado, es más común el uso de arquitecturas cliente-servidor. Por ello mismo, para este trabajo será el estándar. De hecho, todos los experimentos se realizan bajo esta estructura.

6.4. Categorías de Aprendizaje Federado

Antes de dividir por categorías los distintos escenarios, hemos de definir cuáles serán los criterios para hacerlo [RBJLL⁺²³]. Por tanto, empezamos definiendo cuáles serán los elementos clave comunes a escenarios descentralizados que generarán las categorías de FL más relevantes. Estos serán el espacio de características (X), el espacio de clases o etiquetas (Y) y el espacio muestral (I). La clasificación hecha a continuación se puede ver en la Figura 6.4.

- **Aprendizaje Federado Horizontal u *Horizontal Federated Learning (HFL)***: Cuando los datos se partitionan entre los clientes según las muestras, lo que significa que cada cliente posee diferentes muestras del conjunto total de datos de entrenamiento. Formalmente, se puede definir como:

$$X_i = X_j, \quad Y_i = Y_j, \quad I_i \neq I_j, \quad \forall D_i, D_j, \quad i \neq j,$$

donde el espacio de características y etiquetas de los clientes (i, j) se representa por (X_i, Y_i) y (X_j, Y_j) , y se asume que es el mismo, mientras que las muestras I_i y I_j no son iguales. D_i y D_j representan los datos de los clientes i y j . Es adecuado para entrenar modelos en datos recolectados de numerosos dispositivos similares, como *smartphones* o dispositivos IoT.

- **Aprendizaje Federado Vertical o *Vertical Federated Learning (VFL)***: Cuando los datos se partitionan entre los clientes según las características, lo que significa que cada cliente posee el mismo conjunto de muestras, pero con diferentes conjuntos de características. Formalmente, se puede definir como:

$$X_i \neq X_j, \quad Y_i \neq Y_j, \quad I_i = I_j, \quad \forall D_i, D_j, \quad i \neq j,$$

Es adecuado para entrenar modelos en datos recolectados de un pequeño número de dispositivos con diferentes espacios de características. Por ejemplo, puede utilizarse para predecir resultados médicos basados en datos recolectados de múltiples hospitales, donde cada hospital tiene un conjunto diferente de registros médicos.

- **Aprendizaje Federado por Transferencia o *Transfer Federated Learning (TFL)***: Cuando el conocimiento se transfiere entre múltiples dominios sin que exista solapamiento entre muestras o características [YLCT19]. Formalmente, se puede definir como:

$$X_i \neq X_j, \quad Y_i \neq Y_j, \quad I_i \neq I_j, \quad \forall D_i, D_j, \quad i \neq j,$$

En esta arquitectura no se asume que la distribución de los conjuntos de entrenamiento y validación sean las mismas y estén definidas en el mismo espacio de características.

6.5. Privacidad Diferencial

La **PD**, propuesta en [DMNS06], proporciona un marco matemático para garantizar que la contribución de cualquier usuario individual al entrenamiento del modelo no pueda ser

6. Aprendizaje Federado

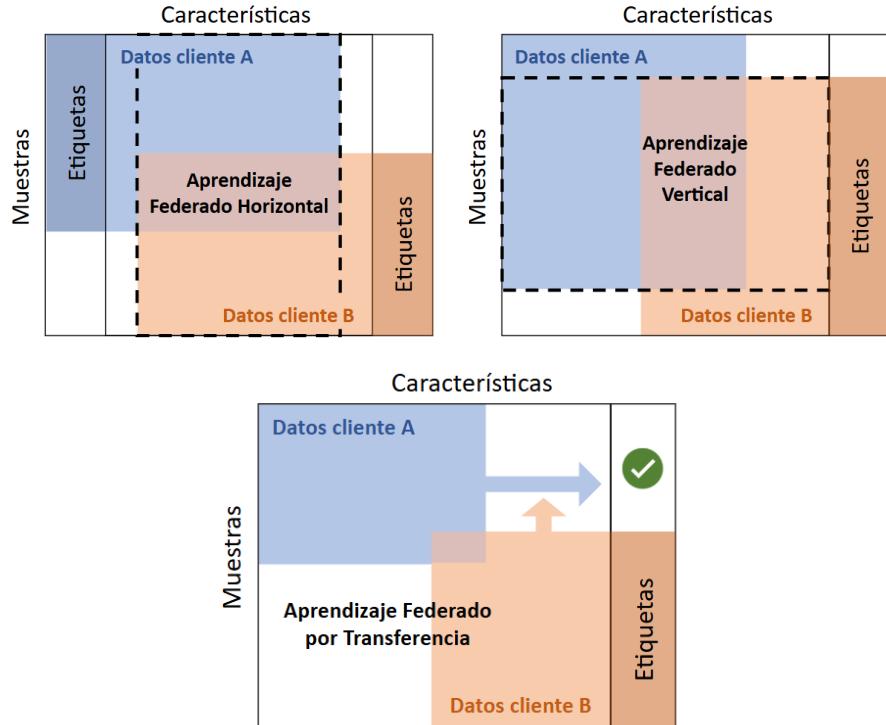


Figura 6.4.: Categorización de los escenarios del Aprendizaje Federado. Esquema extraído de [RBJLL⁺23].

detectada o explotada, incluso si un adversario¹ observa todas las comunicaciones entre los dispositivos y el servidor (ver Figura 6.5).

Para lograrlo, se introducen mecanismos de aleatorización controlada, como la adición de ruido a los gradientes o a las actualizaciones de los modelos locales antes de ser enviados al servidor. Esta perturbación protege la información personal sin eliminar la posibilidad de construir modelos útiles. Además, la PD permite ofrecer garantías formales sobre el nivel de protección alcanzado. Esto último es crucial cuando se pretende que el usuario confíe en el proceso de aprendizaje colaborativo, pues se necesitan garantías reales de seguridad.

Ejemplo de privacidad diferencial

Imaginemos que se quiere recolectar información acerca de cierta población, pongamos, de personas de una determinada edad. Se quiere saber cuántas personas toman café por la mañana.

Para ello, se hace una encuesta a la población, con la pregunta correspondiente, y la posibilidad de responder Sí o No. Las respuestas son almacenadas, pero con la peculiaridad de no mandar las respuestas reales. Se introduce ruido aleatorio en las respuestas de la siguiente manera. Imaginemos un usuario que toma café por la mañana y que ha respondido Sí. Antes de recopilar su respuesta, un mecanismo ingenuo de PD lanza una moneda:

¹Llamaremos adversario a cualquier entidad que pretenda o trate de atacar de cualquier manera la integridad de los datos de los clientes, calidad del entrenamiento u cualquier otro elemento del escenario de FL.

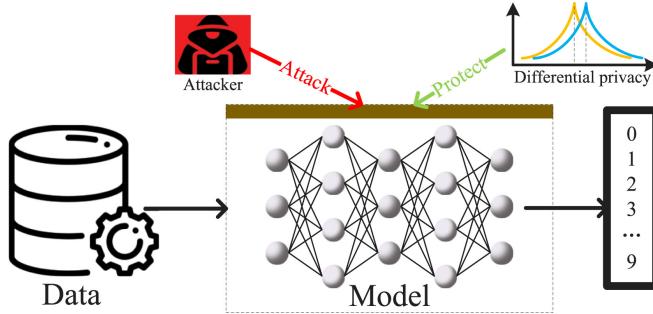


Figura 6.5.: PD en el DL. Escenario donde se utiliza la privacidad diferencial para lidiar con un posible atacante. Imagen extraída de [POG⁺24].

- (i) Si el resultado es *cara*, se envía la respuesta real del usuario.
- (ii) Si el resultado es *cruz*, se lanza una segunda moneda; si vuelve a ser *cara*, se manda *Sí*. Si es *cruz*, se manda *No*.

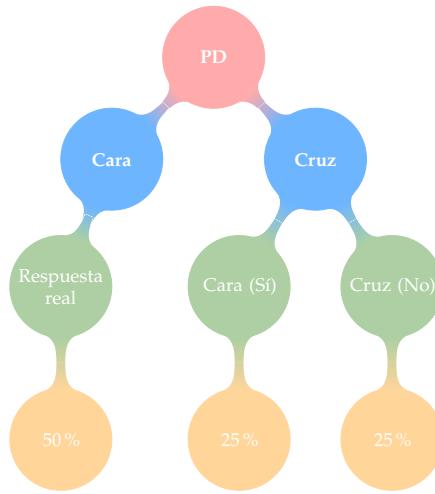


Figura 6.6.: Ejemplo de mecanismo ingenuo de PD. Se añade ruido controlado a los datos reales. Existe una posibilidad entre cuatro de que la respuesta de un usuario que introdujo *Sí*, acabe siendo *No*. De esta manera, no se podrá afirmar con total seguridad cuál fue su respuesta.

En el servidor, se puede ver individualmente los datos de los usuarios, pero debido al ruido añadido, no se puede confiar plenamente en la veracidad de las respuestas de cada uno.

En el caso del usuario en cuestión, aunque haya respondido que *No*, hay *una posibilidad entre cuatro* de que en realidad sí tome café por la mañana (ver Figura 6.6). Puede ser que su respuesta sea el resultado de una *cruz* tanto en la primera moneda como en la segunda. Es la llamada *negación plausible*.

Ahora bien, como se conoce cómo se está añadiendo este *ruido*, se puede compensar y así hacerse una idea bastante precisa de cuántas personas de las encuestadas realmente

6. Aprendizaje Federado

toman café por la mañana. En escenarios reales, como se verá más adelante, se utilizan distribuciones conocidas y estudiadas (Normal, Laplaciana...) de probabilidad para añadir una mayor aleatoriedad.

Si se abstrae este ejemplo a una casuística común en la que se pretende recoger información de relevancia muy sensible acerca de usuarios reales, la tarea de hacer privada esta información es crucial. Se procede entonces a establecer correctamente los fundamentos de este mecanismo para profundizar en su utilidad.

Así, la combinación de **FL** y **PD** establece un marco potente para desarrollar sistemas de **IA** respetuosos con la privacidad, donde los datos de los usuarios permanecen tanto localizados, como *matemáticamente* protegidos contra posibles ataques de reconstrucción o inferencia. En esta sección se dedica un esfuerzo considerable a la explicación detallada del concepto de **PD**, dado que comprender el marco teórico que lo sustenta resulta fundamental para el correcto entendimiento de su aplicación práctica.

6.5.1. Definición matemática

Como se mencionaba, la eficacia de la **PD** se demuestra de forma teórica por su propia definición. Es medible ya que cada acceso a datos privados tiene un coste de privacidad, expresado en términos de ϵ , o en términos de ϵ y δ , dos parámetros que se discuten ahora y serán el eje principal en la experimentación.

Esta interpretación conduce naturalmente a definir la distancia entre bases de datos [RBJLL⁺23].

Definición 6.1. Dos bases de datos x e y se dicen *n*-vecinas si difieren únicamente en n entradas. En particular, si las bases de datos difieren únicamente en un solo elemento de datos ($n = 1$), se consideran simplemente *vecinas*.

Definición 6.2. Se dice que un mecanismo de acceso a bases de datos, \mathcal{M} , preserva la ϵ -PD si para todas las bases de datos vecinas x y y , y para cada posible salida de \mathcal{M} , representada por \mathcal{S} , se cumple que:

$$P[\mathcal{M}(x) \in \mathcal{S}] \leq e^\epsilon P[\mathcal{M}(y) \in \mathcal{S}].$$

Si, por otro lado, para $0 < \delta < 1$ se cumple que:

$$P[\mathcal{M}(x) \in \mathcal{S}] \leq e^\epsilon P[\mathcal{M}(y) \in \mathcal{S}] + \delta,$$

entonces el mecanismo posee la propiedad de (ϵ, δ) -PD, también conocida como privacidad diferencial aproximada.

En otras palabras, la PD especifica un “*presupuesto de privacidad*” dado por ϵ y δ . La forma en que este presupuesto se gasta está dominada por el concepto de *pérdida de privacidad*. La pérdida de privacidad nos permite reinterpretar ϵ y δ de manera mucho más intuitiva:

- ϵ limita la cantidad de pérdida de privacidad permitida, es decir, nuestro presupuesto de privacidad.
- δ representa la probabilidad de exceder el presupuesto de privacidad dado por ϵ , asegurando que, con probabilidad $1 - \delta$, la pérdida de privacidad no sea mayor que ϵ .

Estos conceptos son de gran relevancia en este trabajo, pues nos permitirán controlar, monitorear y asegurar la privacidad mantenida en cualquier proceso de aprendizaje colaborativo.

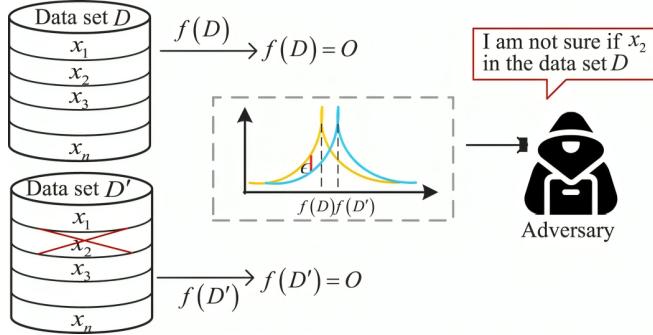


Figura 6.7.: Definición de PD. El atacante no tiene la certeza de que las muestras pertenezcan a los conjuntos de datos. Imagen extraída de [POG⁺24].

En la Figura 6.7 se muestra de manera gráfica el concepto de . Se pretende hacer *invisibles* a individuos, ocultando su presencia. Aplicado al FL, se puede usar en dos momentos:

- (i) **PD local:** Consiste en un mecanismo de aplicación de PD a nivel de cliente, es decir, en el momento en que se entrena localmente los modelos de cada cliente es cuando se añade o aplica la PD.
- (ii) **PD en la agregación:** Se trata de la aplicación de la PD en el paso de agregación de los pesos, explicada en 6.2. De esta manera se consigue un operador de agregación que cumple con la propiedad de (ϵ, δ) -PD².

Dependiendo del tipo de ataque que esté teniendo lugar y la confianza que se tenga en el servidor, se elegirá una u otra aplicación de la PD. Como se explica en [BRG⁺21], el enfoque *local* es adecuado cuando no se puede confiar en el servidor, pero puede ser vulnerable a ciertos ataques de inferencia. El enfoque de PD en la agregación ofrece mejor utilidad del modelo y protección contra ataques más pasivos, pero requiere confiar en el servidor para aplicar correctamente la PD.

6.5.2. Mecanismos de privacidad diferencial

En un escenario práctico, los mecanismos de privacidad diferencial son técnicas diseñadas para garantizar que el acceso a resultados derivados de datos sensibles no comprometa la privacidad de los individuos. Entre los mecanismos más relevantes definidos en [POG⁺24] se encuentran el **mecanismo Laplaciano** y el **mecanismo Gaussiano**, los cuales introducen ruido controlado en las respuestas de funciones de consulta para limitar la cantidad de información que un adversario podría inferir.

El mecanismo Laplaciano es más adecuado para lograr estrictamente ϵ -privacidad diferencial, mientras que el mecanismo Gaussiano se emplea generalmente para garantizar (ϵ, δ) -privacidad diferencial, o privacidad diferencial aproximada [POG⁺24]. Esto se debe a

²En el caso del operador FedAvg, si se añade privacidad diferencial se conoce como *Central Differential Privacy (CDP)* en oposición a la PD local.

6. Aprendizaje Federado

la propia definición de ambos mecanismos y sus distribuciones. El uso de teoría de colas, como se hace en [Coo17], conduce a introducir un pequeño margen de error (nuestro δ) para mantener buenas garantías en el caso del mecanismo Gaussiano. Sin embargo, en la distribución de se controlan de manera mucho más estricta las desviaciones, lo que permite utilizar solo el parámetro ϵ .

El **mecanismo Laplaciano** logra ϵ -privacidad diferencial mediante la adición de ruido extraído de una distribución Laplaciana a la salida de una función de consulta. Se define formalmente como sigue:

Definición 6.3 (Mecanismo Laplaciano). Dado un conjunto de datos D y una función de consulta $f : \mathcal{D} \rightarrow \mathbb{R}^d$, un algoritmo aleatorizado \mathcal{M} satisface ϵ -PD si:

$$\mathcal{M}(D) = f(D) + \text{Lap}\left(\frac{\Delta f}{\epsilon}\right),$$

donde $\Delta f = \max_{D,D'} \|f(D) - f(D')\|_1$ representa la *sensibilidad* de la función f en norma l_1 , es decir, la máxima variación posible entre las salidas de f al evaluar conjuntos de datos vecinos D y D' . $\text{Lap}\left(\frac{\Delta f}{\epsilon}\right)$ indica una variable aleatoria extraída de una distribución Laplaciana con parámetro de escala $\frac{\Delta f}{\epsilon}$.

El **mecanismo Gaussiano** extiende el concepto de privacidad diferencial para situaciones donde es permisible una pequeña probabilidad de error δ , proporcionando así (ϵ, δ) -privacidad diferencial. Su definición es la siguiente:

Definición 6.4 (Mecanismo Gaussiano). Dado un conjunto de datos D y una función de consulta $f : \mathcal{D} \rightarrow \mathbb{R}^d$, un algoritmo aleatorizado $\mathcal{M}(D) = f(D) + \mathcal{N}(0, \sigma^2 I)$ satisface (ϵ, δ) -PD si:

$$\sigma > \frac{\Delta_2 f}{\epsilon} \sqrt{2 \ln(1.25/\delta)},$$

donde $\Delta_2 f = \max_{D,D'} \|f(D) - f(D')\|_2$ es la sensibilidad y $\mathcal{N}(0, \sigma^2 I)$ representa una variable aleatoria distribuida según una distribución normal multivariada centrada en cero con varianza σ^2 por componente.

6.5.3. Propiedades clave

Se introducen tres propiedades fundamentales de la privacidad diferencial que definen la pérdida de privacidad a través de múltiples aplicaciones, y que a menudo sirven como principios guía en el diseño de algoritmos apropiados para cumplir las garantías de privacidad diferencial.

- **Composición secuencial:** La composición secuencial [MT07] logra la preservación de la privacidad para una composición de algoritmo, es decir, una secuencia de algoritmos aleatorizados cuando estos se ejecutan secuencialmente sobre todo un conjunto de datos. El nivel de preservación de la privacidad ofrecido por el algoritmo de composición es la suma de los presupuestos de privacidad en cada paso.

Teorema 6.5 (Composición Secuencial). Supongamos que un algoritmo aleatorizado \mathcal{M}_i satisface ϵ_i -DP sobre un conjunto de datos completo D . Entonces, el conjunto de algoritmos \mathcal{M}_i proporciona $(\sum \epsilon_i)$ -DP.

- **Composición paralela**, [McSo9]: Significa que, si los subconjuntos de datos procesados por un conjunto de algoritmos de privacidad diferencial son disjuntos, entonces el nivel de preservación de la privacidad ofrecido por el algoritmo de composición depende de la peor garantía de privacidad, es decir, del mayor presupuesto de privacidad proporcionado por cualquiera de los algoritmos en la composición.

Teorema 6.6 (Composición Paralela). *Supongamos que un algoritmo aleatorizado \mathcal{M}_i satisface ϵ_i -DP sobre subconjuntos disjuntos del conjunto de datos completo D , respectivamente. Entonces, el conjunto de algoritmos \mathcal{M}_i proporciona $(\max \epsilon_i)$ -DP.*

- **Post-procesamiento:** La propiedad de post-procesamiento [DR⁺14] implica que no existe riesgo de fuga de privacidad, incluso si se aplican cálculos arbitrarios sobre la salida de un algoritmo que cumple privacidad diferencial. Además, la cantidad de ruido disminuirá cuando se aplique correctamente esta propiedad.

Proposición 6.7 (Post-procesamiento). *Supongamos que un algoritmo aleatorizado \mathcal{M} cumple (ϵ, δ) -DP. Sea $f : \mathcal{O} \rightarrow \mathcal{O}'$ una función arbitraria. Entonces, $f \circ \mathcal{M}$ también cumple (ϵ, δ) -DP.*

6.6. Ataques en el Aprendizaje Federado

Aunque el **FL** proporciona una capa de protección en comparación con el aprendizaje centralizado tradicional, como cualquier paradigma de **ML**, no está exento de vulnerabilidades. De hecho, el **FL** introduce nuevos elementos de ataque que pueden comprometer tanto la privacidad de los datos como la integridad del modelo. Dependiendo de la definición y ámbito en que se definen los ataques, se podrán clasificar cada uno de ellos. Es por ello que para entender correctamente su funcionamiento y posibles rangos de ataque, es crucial establecer una taxonomía de los mismos. En esta línea, en [RBJLL⁺23] se da una definición de términos mutuamente excluyentes para poder clasificar los ataques correctamente.

- **Atacante interno/externo:** Uno de los elementos clave de cualquier sistema distribuido es la comunicación entre las diferentes partes. La comunicación es muy vulnerable, ya que puede ser comprometida por agentes externos al sistema de aprendizaje, conocidos como atacantes externos (*outsider attackers*). Cuando el ataque es realizado por uno de los participantes del sistema distribuido, ya sea uno o varios clientes o el propio servidor, se conoce como un atacante interno (*insider attacker*).

Claramente, el alcance de los dos tipos de ataques es muy diferente: los ataques internos son más dañinos y pueden estar dirigidos a modificar el comportamiento del modelo o a inferir información valiosa de otros clientes, mientras que los realizados por atacantes externos usualmente se centran únicamente en inferir información sobre los datos o sobre el modelo de aprendizaje resultante.

Nos enfocamos en los ataques internos, dentro de los cuales destacamos las siguientes categorías:

- **Ataques bizantinos:** Consisten en enviar actualizaciones arbitrarias al servidor, comprometiendo así el rendimiento del modelo global de aprendizaje [LSP19, HLWZ].
- **Ataques sibilinos:** Consisten en ataques colaborativos, ya sea mediante varios atacantes que actúan conjuntamente o mediante la simulación de clientes ficticios, con el objetivo de ser más disruptivos.

6. Aprendizaje Federado

- **Cliente/servidor:** Dentro de los ataques internos, en concreto en un escenario de aprendizaje **HFL**, es natural diferenciar si dichos ataques vienen desde el servidor o desde uno de los clientes. Los ataques desde los clientes tendrán información acerca de uno o varios clientes distintos, mientras que el servidor dispondrá de información sobre la arquitectura del modelo y las actualizaciones de los clientes, entre otras.
- **Información del adversario:** En entornos centralizados, un atacante *white-box* tiene acceso completo al modelo objetivo, incluyendo su arquitectura, parámetros y estado interno. En cambio, un atacante *black-box* no tiene acceso directo al modelo y puede conocer solo información parcial sobre su arquitectura o proceso de entrenamiento. Estas categorías son muy generales; por ello, se introdujo el atacante *grey-box* en [TLG⁺19], que posee conocimientos estadísticos específicos sobre la víctima.

Como hemos visto, en escenarios de **FL**, los atacantes pueden ser clientes o el propio servidor, donde el área de ataque es mayor debido al acceso a datos y comunicaciones. Por esta razón, se requiere una clasificación más precisa del conocimiento de los atacantes en **FL**, particularmente en **HFL** y **VFL**.

En un sistema **HFL** estándar, un atacante que controla un cliente tiene **conocimiento del lado del cliente**, incluyendo:

- Acceso *white-box* al modelo agregado.
- Acceso *white-box* al modelo localmente entrenado del cliente.
- Acceso al conjunto de datos del cliente que controla.

Si un atacante accede a los datos locales de otros clientes o a sus etiquetas, posee *extra client-side knowledge*.

Un atacante que controla el servidor federado tiene **conocimiento del lado del servidor**:

- Acceso *white-box* al modelo agregado tras cada ronda de comunicación.
- Acceso *white-box* a los modelos entrenados compartidos por los clientes o, alternativamente, acceso a sus gradientes.
- Identificadores de los clientes agregados en cada ronda de comunicación.
- Etiquetas y, opcionalmente, el tamaño del conjunto de datos de cada cliente.

En un sistema **VFL** estándar, un atacante que controla un cliente posee **conocimiento del lado del grupo**:

- Acceso *white-box* a los parámetros relacionados con las características del cliente controlado.
- Acceso al conjunto de datos privado del cliente.
- Acceso a la salida parcial de los parámetros al realizar inferencias.

Si además el atacante accede a información sobre las características de otros clientes, posee *conocimiento extra de la parte del grupo*.

Un atacante que controla el coordinador de aprendizaje en un sistema **VFL** posee **conocimiento de terceras partes** (*third party-side knowledge*), incluyendo:

- Gradientes compartidos por cada cliente.
- La pérdida (*loss*) calculada.
- La salida parcial de cada cliente al realizar inferencias.

Si el atacante solo tiene acceso a un subconjunto de la información especificada, se considera que tiene **conocimiento parcial** (*partial knowledge*). Además, las defensas están diseñadas para reducir el conocimiento del atacante, por lo que, en presencia de defensas, se espera que los atacantes tengan conocimiento parcial.

En sistemas **HFL** y **VFL**, si el atacante únicamente accede a las salidas del modelo federado, se dice que posee **conocimiento externo** (*outsider-side knowledge*).

Se resalta que los tipos de conocimiento no son mutuamente excluyentes; un atacante puede poseer varios tipos simultáneamente. Escenarios de ataque realistas requieren generalmente menor conocimiento, mientras que ataques complejos requieren información de múltiples participantes.

- **Pasivo/Malicioso:** Un adversario malicioso o activo intenta hacerse con información del proceso de entrenamiento del modelo, intentando corromperlo. Por el contrario, un atacante pasivo (u *honesto pero curioso*) no interfiere en el proceso de aprendizaje, siguiendo los protocolos establecidos por todas las partes, pero intenta obtener o inferir información privada de otros clientes.
- **Colusión/No colusión:** De forma general, una colusión es un acuerdo entre dos o más partes para limitar la competencia. En el aprendizaje federado, el problema reside en el hecho de que es el atacante que controla más clientes el que tendrá más poder. Existen dos tipos de colusiones:
 - **Servidor-Participantes:** El atacante controla solo algunos de los participantes y el servidor, pretendiendo inferir información de los nodos que no controla.
 - **Participante-Participante:** El adversario controla algunos clientes, y pretende inferir información del servidor u otros clientes, o directamente dañar el rendimiento del modelo.

Los denominados *ataques adversarios* son el principal problema del aprendizaje federado, pues existen una gran diversidad de ellos, lo que los convierte en un reto individual del que defenderse. La propia naturaleza del paradigma federado lo convierte en vulnerable ante un mar de diferentes ataques. Esto es suficiente para motivar la clasificación y taxonomía que se propone en [RBJLL⁺₂₃].

La principal diferencia que divide a los posibles ataques es:

- **Ataques al modelo:** Pretenden modificar el modelo y el comportamiento del modelo aprendido en común.
- **Ataques a la privacidad:** Su propósito es el de inferir información sensible de los participantes en el proceso de aprendizaje.

Esta primera clasificación se puede representar como se hace en la Figura 6.8. En este trabajo se tendrá el foco sobre ambos tipos de ataques, pues en el proceso de experimentación se simulan los dos tipos de ataques, evidenciando así que la privacidad diferencial es útil en ambos casos.

6.6.1. Ataques al modelo

En un escenario donde los participantes comparten el mismo espacio de características y etiquetas (**HFL**), existe el riesgo de que uno de ellos intente dañar el rendimiento del modelo.

6. Aprendizaje Federado

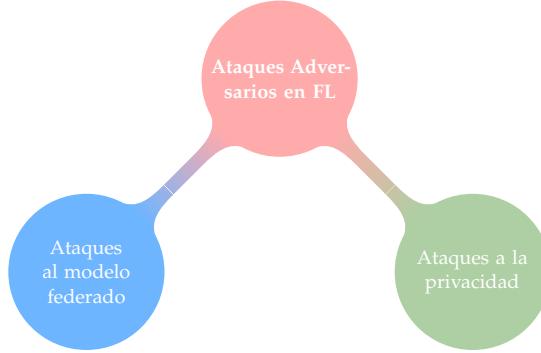


Figura 6.8.: Clasificación de los ataques según su objetivo en **FL**. Imagen extraída de [RBJLL⁺23].

Para ello, se pueden enviar actualizaciones *envenenadas* al servidor, sin que el mismo servidor sea consciente, pues no puede tener acceso a los datos de ninguno de los clientes.

En términos generales, estos ataques son realizados por los propios clientes, y la característica de *white-box* de dichos ataques hace referencia a situaciones en las que el atacante posee conocimiento del lado del cliente. Esto implica que uno o varios clientes pueden ser adversarios o atacantes. En ciertos casos, se asume que los atacantes tienen acceso a información adicional, como detalles del mecanismo de agregación utilizado por el servidor. Sin embargo, este escenario no suele ser realista. Por esta razón, se pone énfasis en aquellos ataques que únicamente requieren información del cliente adversario, como se hace en [RBJLL⁺23].

En la Figura 6.9 se propone una taxonomía o clasificación de los distintos tipos de ataques al modelo, según cuatro criterios distintos. Se exponen a continuación todos ellos, haciendo especial énfasis en aquellos que han sido simulados en la experimentación del trabajo.

Según el momento del ataque

Dependiendo del momento en que tenga lugar el ataque, la influencia que tendrá el mismo en el modelo federado tendrá un alcance u otro. En concreto, se distinguen dos tipos de ataques:

- **Durante el entrenamiento:** Se considera tiempo de entrenamiento desde la recolección de los datos y su preparación como el entrenamiento del modelo. Puede ocurrir que los ataques se hagan de manera continua o de manera aislada en un momento en particular. Este tipo de ataques son los más comunes en la literatura, pues pueden interrumpir y modificar el comportamiento del modelo cuando todavía está siendo entrenado, pudiendo así inferir información de los datos de entrenamiento [BCMC18, NSH19].
- **Durante la inferencia:** Una vez el modelo ya ha sido entrenado, se pueden llevar a cabo ataques en la inferencia del mismo. También se denominan *ataques de evasión*, como se hace en [BCM⁺13]. En general, el objetivo no es modificar el comportamiento del modelo entrenado, si no generar predicciones erróneas del mismo y/o recabar información acerca de sus características.

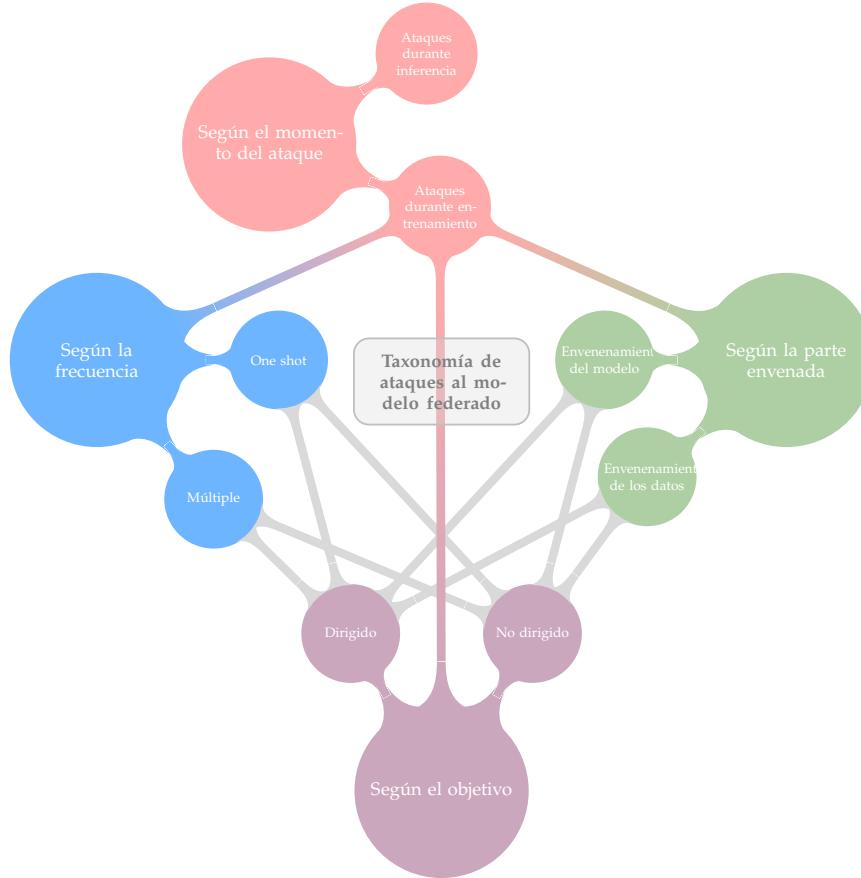


Figura 6.9.: Taxonomía para los ataques al modelo, según el momento de ataque, la parte envenenada, la frecuencia y el objetivo del ataque. Imagen extraída de [RBJLL⁺23].

Según el objetivo

Se trata de la clasificación más frecuentemente usada en la literatura, lo que la convierte en la más relevante de las cuatro que se ven aquí. Aunque todos los ataques de este tipo tienen como objetivo modificar el modelo federado, las modificaciones pueden variar bastante. Distinguimos dos tipos principales de ataques, dependiendo del objetivo:

- **Ataques dirigidos o *backdoor*:** Tienen como tarea principal *inyectar* una tarea secundaria al modelo, es decir, el modelo aprenderá su tarea principal pero a la vez estará asimilando una tarea que no estaba definida en un principio. Es por esto último que este tipo de ataques son tan difíciles de detectar [BCMC19].

No representan un peligro para la tarea principal del escenario federado, sin embargo, sí que lo son para la integridad del sistema global, pues el atacante se aprovecha de la estructura federada para realizar una acción secundaria.

Existen diversos ataques posibles, pues las tareas secundarias que se pueden inyectar son a su vez diversas. Es por existe también una taxonomía de las mismas, basándose

6. Aprendizaje Federado

en diferentes criterios. No es el objetivo de este trabajo desglosar y explicar cada uno de ellos, pero sí que se explicarán, al menos, los usados en la experimentación.

En concreto, se tratan las *estrategias patrón-clave* [BVH⁺20]. El objetivo de estos ataques es que el modelo asocie un patrón específico dentro de una muestra, con una etiqueta particular. Por ejemplo, si se están tratando con imágenes, se pueden alterar de manera que presenten patrones comunes en imágenes con etiquetas distintas, y que el modelo las clasifique dentro de la misma clase. Se dará algún ejemplo de esto cuando se presente la experimentación realizada.

- **Ataques no dirigidos:** En oposición a los anteriores, estos ataques tienen como única tarea u objetivo comprometer el rendimiento del modelo en su tarea principal, cualquiera que fuese. El caso más extremo es el conocido *ataque bizantino*, presentandos con anterioridad en la memoria, en los cuales los clientes adversarios comparten actualizaciones generadas de manera arbitraria, o entrenando en datos generados aleatoriamente. Estos ataques son, claramente, menos *sigilosos* o más detectables que los ataques dirigidos, pues con un mero análisis de los modelos locales, se puede averiguar la identidad de los adversarios.

Según la parte envenenada

La mayoría de los ataques en tiempo de entrenamiento se basan en *envenenar* información de los clientes para poder corromper así el modelo global entrenado. Entonces, dependiendo de qué parte de la información del cliente esté corrupta, podemos diferenciar entre ataques de envenenamiento de datos o del modelo, como se hace en la Figura 6.9.

- **Ataques de envenenamiento de datos:** Se asume que el atacante tiene acceso a los datos de entrenamiento de uno o más clientes y que puede modificarlos a su antojo. Dependiendo de las características de la corrupción de los datos, se pueden diferenciar los siguientes ataques:
 - *Label flipping*, [TTGL20]: Consiste en modificar las etiquetas de una parte de los datos de entrenamiento. Puede ser un ataque dirigido, si se cambia alguna etiqueta en específico, o no dirigido, si el intercambio es totalmente aleatorio.
En la experimentación se comentará más sobre este tipo de ataques pues se realiza una simulación del mismo.
 - Envenenamiento de muestras: En este caso se modifican directamente las muestras, no como en el anterior que se modificaban las etiquetas de los datos de entrenamiento. Existe muchas maneras de corromper los datos, como incluir patrones en las muestras o introducir ruido uniforme con el objetivo de reducir el rendimiento del modelo.
 - Ataques *out-of-distribution*: Exactamente igual que el ataque anterior, solo que en lugar de modificar las muestras de entrenamiento, los adversarios inyectan muestras que no estaban dentro de la distribución inicial de datos [FRL21].
- **Ataques de envenenamiento del modelo:** Este tipo de ataques consisten en enviar directamente actualizaciones envenenadas del modelo de los clientes al servidor. Aunque naturalmente el envenenamiento de los datos dé como resultado envenenamientos del modelo, estos ataques se caracterizan por enviar directamente actualizaciones envenenadas.

También nos centraremos en un tipo específico de ataque de este tipo en la experimentación, generando ruido *Gaussiano* aleatorio en las actualizaciones de algún cliente.

Según la frecuencia del ataque

Como la fase de entrenamiento se mantiene en largos períodos de tiempo en muchos escenarios, los ataques pueden tener lugar en cualquier momento del entrenamiento y en una o muchas ocasiones, [BVH⁺20].

- Ataques *one-shot*: Se realiza el ataque en un único momento del entrenamiento, en una ronda de aprendizaje determinada.
- Ataques múltiples o adaptativos: Se realizan ataques de manera continua durante el proceso de entrenamiento, o bien durante todas las rondas o bien durante una parte de ellas.

Se comentan brevemente este tipo de ataques por completitud del trabajo, aunque no serán objeto de estudio aquí.

6.6.2. Ataques a la privacidad

Los ataques a la privacidad están diseñados para revelar información sobre los participantes de una tarea de **ML**. No solo representan una amenaza para la privacidad de los datos utilizados para entrenar los modelos, sino también para las personas que han aceptado compartir sus datos privados. Este proceso de aprendizaje expone una amplia superficie de ataque. Aunque los datos privados nunca abandonan al propietario, los modelos intercambiados pueden memorizar información del conjunto de datos privado. En esta sección, presentamos una taxonomía que pretende facilitar la comprensión de la diversidad de ataques a la privacidad. Esta taxonomía está organizada en torno al objetivo del atacante, y un resumen de la misma se muestra en la Figura 6.10. De nuevo, seguiremos el esquema de presentación de [RBJLL⁺23].

Ataques de inferencia de características

También conocidos como *ataques de reconstrucción* en escenarios de **HFL**. Tienen como objetivo recuperar muestras del conjunto de datos de un cliente que esté participando en un escenario de **FL**. Generalmente, estos datos suelen ser texto o imágenes (en la experimentación serán imágenes). Se pueden dividir estos ataques en dos categorías, dependiendo de la información compartida entre los clientes y el servidor:

- **Basados en gradiente**: Los clientes comparten sus gradientes con el servidor. Así, se proponen ataques basados en la información que se puede extraer de estos gradientes. Hay diversos ataques de este tipo [ZLH19], pero no serán objeto de estudio aquí.
- **Basados en parámetros**: Los clientes comparten sus parámetros del modelo local con el servidor. A partir de estos, se pueden reconstruir directamente muestras de los clientes. En concreto, en [HAPC17], se pone el foco en la reconstrucción de imágenes de los clientes. Para ello se usa una *Generative Adversarial Networks (GAN)*. Esto se explicará más a fondo en la experimentación pues este ataque se simula en un escenario ficticio de **FL**.

6. Aprendizaje Federado

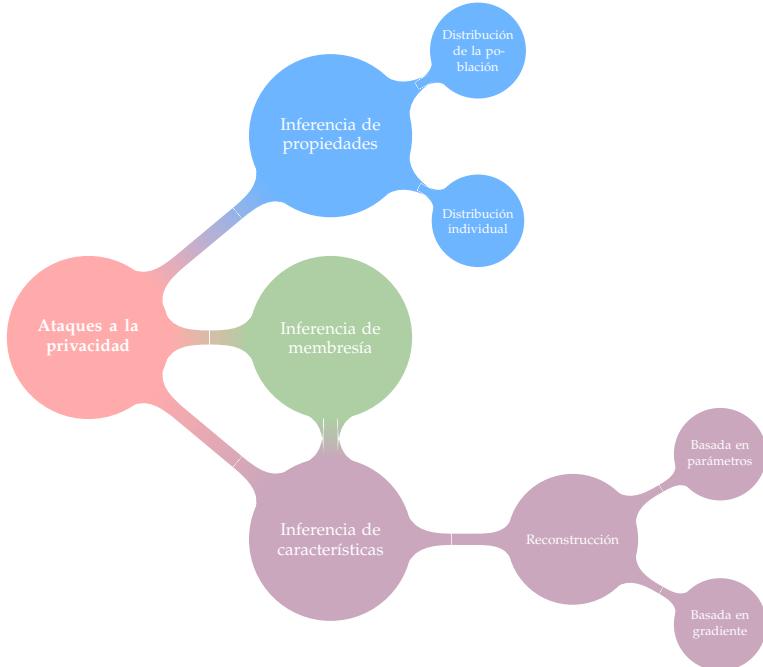


Figura 6.10.: Taxonomía para los ataques a la privacidad, según el momento de ataque, la parte envenenada, la frecuencia y el objetivo del ataque. Imagen extraída de [RBJLL⁺23].

Eso sí, el atacante necesita información de parte de cliente y conocimiento de fuera del cliente también para poder llevar a cabo el ataque. Este conocimiento que mencionamos se refiere a que el adversario necesita asumir que el cliente objetivo del ataque y el mismo comparte una etiqueta, para que el ataque pueda ocurrir dirigido a una etiqueta no compartida.

A parte del mencionado, existe otros tipos de ataques basados en parámetros, haciendo uso de GAN's y otras técnicas que no procede su explicación.

Se ha comentado que estos ataques son para escenarios HFL y hacen uso de modelos profundos de aprendizaje. Sin embargo, el VFL ofrece una mayor variedad de posibles modelos de aprendizaje que se pueden beneficiar de estas comunicaciones entre clientes y servidores [LWXO21], como modelos de árboles de decisión o regresión logística, entre otros.

Ataques de inferencia de membresía

El principal objetivo de estos ataques es determinar si una muestra en concreto ha sido usada para entrenar el modelo de la víctima, dado el modelo de este cliente y algunos datos. Generalmente, tienen lugar en la fase de entrenamiento.

Ataques de inferencia de propiedades

Este tipo de ataques, también conocidos como *ataques de inferencia de atributos*, buscan extraer si cierta propiedad de un cliente o de la población entera de clientes (que podría no estar relacionada con la tarea principal del modelo) está presente o no en el modelo federado. Es decir, se pretende inferir una propiedad de un individuo o de todos que, en un principio, no había intención de compartir.

7. Optimización en la aplicación de Privacidad Diferencial

Como se venía adelantando en varias ocasiones en esta memoria, el objetivo de este TFG es poner a prueba los métodos descritos en la parte de matemáticas para la optimización de los parámetros ϵ y δ en la aplicación de la PD como método de defensa para diversos ataques. En concreto, se usará el método al que se ha dedicado más esfuerzo y extensión: el método Simplex.

Para ello, en esta sección se describe el desarrollo seguido para poder definir correctamente el problema, y así poder optimizar, en tiempo real de entrenamiento en un escenario simulado de , los parámetros de la con un objetivo claro: **maximizar rendimiento y minimizar gasto de presupuesto de privacidad**, brindando así una buena defensa contra los ataques y tratando de mitigar al máximo los efectos de la PD en el rendimiento de los modelos.

Para esto, se decide implementar un PPL con las restricciones necesarias para modelar el problema. Se opta por los PPL en lugar de los problemas convexos, pues, como se comentaba en la parte de matemáticas, es bastante difícil cumplir con las restricciones de convexidad en problemas reales. Cualquier restricción mal formulada o que no cumpla ser convexa no permite una correcta definición del problema y , por tanto, una correspondiente solución. Por tanto, lo que se propone aquí es usar siempre el método Simplex, haciendo una aproximación lineal sencilla para las restricciones que hagan falta. Se describe a continuación este proceso.

Linealización de la relación entre Ruido y Privacidad

En el mecanismo *Gaussiano de PD* que se comentaba en la Definición 6.4, el nivel de ruido σ requerido para garantizar privacidad (ϵ, δ) se define como:

$$\sigma = \frac{s \cdot \sqrt{2 \log(1.25/\delta)}}{\epsilon}$$

donde s es la sensibilidad del mecanismo. Esta expresión muestra que σ depende de manera inversamente proporcional a ϵ , esto es, $\sigma(\epsilon) \propto \frac{1}{\epsilon}$.

Esta relación es no lineal, lo que implica que pequeños cambios en ϵ pueden tener efectos no uniformes sobre el ruido, especialmente cuando ϵ es pequeño. Por tanto, se lleva a cabo una sencilla aproximación lineal local, de la que no se desarrollará en profundidad la teoría necesaria, pues escapa del ámbito de este trabajo.

Para facilitar la optimización, se propone una linealización de esta relación en un intervalo acotado alrededor de los posibles valores de ϵ . Esto se hace evaluando σ en dos puntos extremos del intervalo:

$$\sigma_{\text{lower}} = \frac{s \cdot \sqrt{2 \log(1.25/\delta_t)}}{\epsilon_{\text{lower}}}, \quad \sigma_{\text{upper}} = \frac{s \cdot \sqrt{2 \log(1.25/\delta_t)}}{\epsilon_{\text{upper}}}.$$

Luego, se calcula la recta que aproxima la función en ese intervalo:

$$a = \frac{\sigma_{\text{upper}} - \sigma_{\text{lower}}}{\epsilon_{\text{upper}} - \epsilon_{\text{lower}}}, \quad b = \sigma_{\text{lower}} - a \cdot \epsilon_{\text{lower}}.$$

7. Optimización en la aplicación de Privacidad Diferencial

De este modo, se obtiene una aproximación afín de la forma:

$$\sigma(\epsilon) \approx a \cdot \epsilon + b.$$

Este método permite utilizar técnicas de **PL** (Símplex en nuestro caso) sencillas y eficientes computacionalmente. En un escenario donde las restricciones no cumplen con la convexidad requerida por los métodos de **PC**, y hay que hacer de manera obligatoria alguna aproximación, se opta por usar directamente el caso particular de optimización lineal de la **PL**. Este tipo de aproximaciones son razonablemente precisas si el intervalo de variación de ϵ es relativamente pequeño, que es justamente nuestro caso.

En el Algoritmo 1, descrito a continuación, se propone una forma de optimizar tanto ϵ como δ para aplicar **PD** en un escenario federado iterativo por rondas. Para ello, se utiliza la aproximación descrita previamente para obtener unos nuevos parámetros, definiendo el siguiente problema:

$$\begin{aligned} \min_{\Delta\epsilon, \Delta\delta} \quad & c_1 \cdot \Delta\epsilon + c_2 \cdot \Delta\delta \\ \text{sujeto a: } \quad & \Delta\epsilon + \Delta\delta \leq \text{presupuesto}_{\text{total}} - \text{presupuesto}_{\text{usado}} \\ & \Delta\epsilon \in [\max(-0.2, -\epsilon_t + 0.5), 1.0] \\ & \Delta\delta \in [\max(-10^{-7}, -\delta_t + 10^{-4}), 10^{-7}] \end{aligned}$$

Donde:

- $\Delta\epsilon$ es el incremento propuesto sobre el parámetro de privacidad ϵ_t (ϵ para la ronda t).
- $\Delta\delta$ es el incremento propuesto sobre el parámetro de privacidad δ_t (δ para la ronda t).
- c_1, c_2 representan los coeficientes de coste, que penalizan los cambios en ϵ y δ considerando la pérdida y el ruido estimado.
- ϵ_t, δ_t son los valores actuales de los parámetros de privacidad en la iteración o ronda de entrenamiento t .
- $\text{presupuesto}_{\text{total}}$ es la cantidad total de presupuesto de privacidad disponible.
- $\text{presupuesto}_{\text{usado}}$ es la cantidad de presupuesto ya consumido hasta la iteración actual.

El primer término en las restricciones garantiza que el cambio total propuesto no exceda el presupuesto de privacidad restante. Los intervalos de $\Delta\epsilon$ y $\Delta\delta$ aseguran que los nuevos valores se mantengan dentro de rangos numéricamente estables y razonables¹.

Por tanto y a modo de resumen de lo expuesto, el problema a resolver por el algoritmo Símplex consiste en minimizar la suma de las variaciones de ϵ y δ , con dos restricciones: mantenerse siempre dentro de unos rangos establecidos para los dos parámetros y no superar el límite del presupuesto total de privacidad para todas las rondas. Como se explica antes, los coeficientes c_1 y c_2 permiten al algoritmo penalizar o relajar el incremento de los parámetros dependiendo de la pérdida en entrenamiento del modelo global en la ronda anterior. De nuevo, en el Algoritmo 1 se explica cómo se realiza esto.

¹Los valores mostrados para los dos parámetros no son aleatorios, han sido escogidos siguiendo los valores que se dan en [DR⁺14] y tras diversas pruebas llevadas a cabo.

Algorithm 1 Optimización de (ϵ, δ) en una iteración de **FL**

Require: $loss_t, loss_{min}, \epsilon_t, \delta_t, sensibilidad, presupuesto_total, presupuesto_usado, \gamma$

Ensure: Nuevos parámetros $\epsilon_{t+1}, \delta_{t+1}$

```

1:  $\Delta loss \leftarrow loss_t - loss_{min}$ 
2:  $\alpha \leftarrow 1 + \max(0, \Delta loss), \beta \leftarrow 1 - \min(0, \Delta loss)$ 
3:  $\epsilon_{lower} \leftarrow \max(0.1, \epsilon_t - 0.1), \epsilon_{upper} \leftarrow \min(1.5, \epsilon_t + 0.1)$ 

4: if  $\epsilon_{upper} = \epsilon_{lower}$  then
5:    $\epsilon_{upper} \leftarrow \epsilon_{upper} + 10^{-8}$ 

6: end if
7:  $\sigma_{lower} \leftarrow \frac{sensibilidad \cdot \sqrt{2 \log(1.25/\delta_t)}}{\epsilon_{lower}}, \sigma_{upper} \leftarrow \frac{sensibilidad \cdot \sqrt{2 \log(1.25/\delta_t)}}{\epsilon_{upper}}$ 
8:  $a \leftarrow \frac{\sigma_{upper} - \sigma_{lower}}{\epsilon_{upper} - \epsilon_{lower}}, b \leftarrow \sigma_{lower} - a \cdot \epsilon_{lower}$ 

9: if  $\Delta loss > 0$  then
10:    $c \leftarrow [\alpha + \gamma \cdot a + 1.5, \beta + 1.5]$ 
11: else
12:    $c \leftarrow [-(\alpha + \gamma \cdot a - 1.5), -(\beta - 1.5)]$ 
13: end if

14:  $presupuesto\_efectivo \leftarrow presupuesto\_total - presupuesto\_usado$ 
15: Restricciones:  $A \leftarrow [[1, 1]], b \leftarrow [presupuesto\_efectivo]$ 
16: Límites de búsqueda:
17:    $\Delta \epsilon \in [\max(-0.2, -\epsilon_t + 0.5), 1.0]$ 
18:    $\Delta \delta \in [\max(-10^{-7}, -\delta_t + 0.0001), 10^{-7}]$ 
19: Resolver usando Simplex:
20:    $\min c^\top x$  sujeto a  $Ax \leq b$  y  $x$  en las restricciones
21: if solución then
22:    $[\Delta \epsilon, \Delta \delta] \leftarrow$  solución óptima
23:    $\epsilon_{t+1} \leftarrow \epsilon_t + \Delta \epsilon$ 
24:    $\delta_{t+1} \leftarrow \delta_t + \Delta \delta$ 
25: else
26:    $\epsilon_{t+1} \leftarrow \epsilon_t$ 
27:    $\delta_{t+1} \leftarrow \delta_t$ 
28: end if
29: return  $\epsilon_{t+1}, \delta_{t+1}$ 

```

Se permite entonces, según lo expuesto, al parámetro de presupuesto de privacidad, ϵ , estar en el rango de valores $[0.5, 1.5]$. Como se veía en el desarrollo de la **PD**, cuanto menor sea este valor, por definición, la privacidad será mayor, a cambio de una dificultad añadida al entrenamiento del modelo, resultando esto en un peor rendimiento, en general.

Este algoritmo propuesto se basa en la idea de que algoritmos clásicos como puede ser Simplex, si se define correctamente el problema a optimizar, pueden dar buenos resultados. Se demostrará esto en la experimentación, pero a modo de introducción, los algoritmos clásicos de **PL**, pese a haber sido desarrollados hace más de 70 años, ofrecen una solución sencilla y elegante a problemas aparentemente complejos de optimización.

A modo de resumen, en esta sección, el ajuste dinámico de los parámetros de **PD** se

7. Optimización en la aplicación de Privacidad Diferencial

aborda mediante formulaciones de , lo que implica una linealización implícita del problema subyacente. Este enfoque simplifica la relación, potencialmente no lineal, entre el presupuesto de privacidad (ϵ, δ) y el rendimiento del modelo.

Se adopta esta estrategia principalmente por motivos de dificultad computacional y claridad en la implementación. No obstante, el estudio detallado de las posibles no linealidades y su impacto en la optimización se considera fuera del alcance de este trabajo, ya que no forma parte de los objetivos establecidos.

Trabajos como los de Dvorkin et al.[DR⁺14] y Yuan et al.[YYZH16] exploran formulaciones más complejas mediante programación convexa o estocástica, ofreciendo alternativas más generales pero también más costosas computacionalmente. O también, por ejemplo, en [SLW25] se sigue un enfoque de optimización de la privacidad donde el problema a optimizar es de una complejidad mucho mayor (incluyendo duales del problema y otras técnicas avanzadas) que uno como el lineal propuesto en esta sección.

Por simplicidad y por el carácter demostrativo de este TFG, se ha optado por un modelo lineal que permite capturar adecuadamente la dinámica de las rondas de entrenamiento bajo restricciones de privacidad. Usar versiones más simples del problema parece tener sentido en escenarios donde las restricciones no son demasiado complicadas. Parece razonable adaptar la complejidad de las soluciones a la del problema.

8. Entorno experimental

En esta sección se especifica la configuración de cada experimento realizado en este trabajo, con el fin de poder replicar los resultados. Se detalla a continuación la configuración escogida, incluyendo datos de entrenamiento, ataques simulados, métricas empleadas para la evaluación y otros parámetros de relevancia.

8.1. Conjuntos de datos

Para la evaluación experimental de los distintos escenarios de ataques y defensas considerados en este trabajo, se utilizaron tres conjuntos de datos ampliamente conocidos en el ámbito del aprendizaje automático: MNIST, CIFAR-10 y FashionMNIST. Se ha elegido trabajar con imágenes por intentar ser lo más visual posible, pues se pretende poner a prueba la privacidad de los datos. Además, estos conjuntos de datos han sido de referencia en numerosos ensayos y simulaciones. A continuación, se describe brevemente cada uno de ellos, así como su uso específico en los diferentes experimentos:

- **MNIST**, [Den12]: Este conjunto de datos contiene imágenes en escala de grises de dígitos manuscritos del 0 al 9, con una resolución de 28×28 píxeles (Figura 8.1). Consta de 60,000 ejemplos de entrenamiento y 10,000 de prueba. Su simplicidad y alta calidad lo hacen adecuado para pruebas rápidas y claras de distintos métodos de ataque y defensa. En este trabajo, MNIST se ha utilizado en los cuatro experimentos: en el experimento de *label flipping*, donde se evalúa la sensibilidad del modelo ante etiquetas manipuladas en el conjunto de entrenamiento, en el ataque *gaussiano*, en el *backdoor* y en el experimento final de ataque mediante reconstrucción de una **GAN**, donde se simula la reconstrucción de una muestra que no pertenece al conjunto de entrenamiento del adversario, en concreto de la clase 0.



Figura 8.1.: 5 muestras escogidas aleatoriamente del conjunto MNIST.

- **CIFAR-10**, [Krio9]: Este conjunto de datos contiene imágenes a color de 32×32 píxeles distribuidas en 10 clases diferentes que incluyen animales y vehículos (Figura 8.2). Cada clase cuenta con 6,000 imágenes, divididas en 50,000 ejemplos de entrenamiento y 10,000 de prueba. Debido a su mayor complejidad visual, CIFAR-10 fue empleado en todos los experimentos salvo en la reconstrucción mediante una **GAN**. Esto se debe a la gran complejidad de entrenar una red de este tipo, es altamente sensible y, sobre todo, para este tipo de imágenes mucho más complejas que el caso de MNIST.

8. Entorno experimental



Figura 8.2.: 5 muestras aleatorias del conjunto CIFAR10, junto a su etiqueta real.

- **FashionMNIST, [XRV17]:** Este conjunto es una alternativa moderna a MNIST, compuesto por imágenes en escala de grises de 28×28 píxeles de artículos de ropa, organizados en 10 categorías (Figura 8.3). Presenta un mayor desafío visual que MNIST, manteniendo una estructura de datos similar. De nuevo, este conjunto de datos ha sido utilizado para los tres primeros ataques, los que no involucran el entrenamiento de una **GAN**, por complejidad.

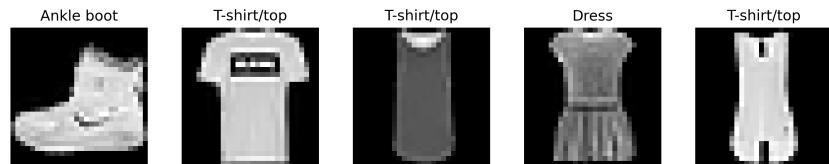


Figura 8.3.: 5 muestras aleatorias del conjunto FashionMNIST, junto a su etiqueta real.

Conjuntos	Tipo	Train	Test	Etiquetas	Ataque
MNIST	Escala de grises	60,000	10,000	10	Label Flipping Gaussian Backdoor DMUG ¹
FashionMNIST	Escala de grises	60,000	10,000	10	Label Flipping Gaussian Backdoor
CIFAR-10	RGB	50,000	10,000	10	Label Flipping Gaussian Backdoor

Tabla 8.1.: Resumen de los tres conjuntos de datos usados en la experimentación y para los ataques que han sido usados.

La elección de estos conjuntos de datos permite una evaluación diversa, abarcando desde escenarios simples hasta otros más complejos, y ofrece una base sólida para analizar la eficacia de las técnicas de optimización de la privacidad propuestas. Los ataques realizados en estos conjuntos de datos se explican a continuación.

¹La alta sensibilidad durante el entrenamiento de la **GAN**, junto a grandes tiempos de ejecución para el caso de MNIST, nos lleva a prescindir de este ataque en FashionMNIST y CIFAR-10.

8.2. Escenarios de ataque

Se detallan a continuación las simulaciones llevadas a cabo, junto a la explicación de los ataques ya mencionados. Dado que el objetivo de este trabajo es hacer una comparativa exhaustiva entre una aplicación de privacidad dinámica y otra estática, no se profundiza demasiado en sus implementaciones o funcionamiento, pues se sale del marco de este trabajo. Para encontrar la implementación de los ataques se puede acudir a las referencias dadas.

El flujo es similar en todos los ataques: (1) se realiza un entrenamiento de una red de una época por ronda, tras la cual, (2) los clientes (tanto benignos como malignos) envían sus actualizaciones al servidor. (3) Este las agrega y (4) comparte los pesos agregados con los clientes. En la siguiente ronda, los clientes partirán su entrenamiento del modelo agregado compartido por el servidor. Este proceso se repite durante 100 rondas. Los elementos **comunes** en todos los experimentos realizados son los siguientes:

- Se simulan 100 rondas de comunicación entre servidor (que actuará en todos los casos como agregador de las actualizaciones) y clientes.
- Todos los clientes comparten mismo modelo (en estructura) y datos de un mismo conjunto de datos, repartidos de sin reemplazo, es decir, cada cliente tendrá una porción exclusiva del conjunto de datos.
- En los tres ataques al modelo, se usa **CDP** (con sensibilidad 0.01, para menor *agresividad* del ruido aplicado) como mecanismo de defensa, pues se pretende evitar la corrupción y el peor rendimiento del modelo federado. En cambio, en el ataque a la privacidad se usa **PD-local** en el entrenamiento de los clientes benignos.
- La optimización de la **PD** se hace exactamente igual en los cuatro escenarios. Se usan los mismos rangos para los parámetros, al igual que las constantes en la definición del problema de **PL**.
- En la simulación de los cuatro ataques, se hace una **comparación** de los mismos cuando se aplica una **optimización de la PD** con una aplicación **estática**, manteniendo los mismos valores de ϵ y δ durante todas las rondas.
- En cada ronda, los clientes entran una sola época su modelo local, para después enviar las actualizaciones al servidor.
- En la aplicación de **PD** estática, se usa la media de los extremos del rango de valores que puede tomar cuando se realiza la optimización. Esto es $\epsilon = 1.0$ y $\delta = 0.001$. Es justamente por esto que se propone como presupuesto global un valor de 100, agotando así el mecanismo estático todo el disponible, mientras que la aplicación dinámica veremos que consigue quedar por debajo de este valor en el uso de ϵ .
- Por último, como análisis adicional, se incluyen experimentos variando el número de atacantes presentes en el escenario simulado. Esto se hace con el objetivo de comprobar cómo de resiliente es el método de optimización propuesto frente a un ataque más agresivo.

Se detallan a continuación los ataques simulados. Siguiendo la práctica más común en la literatura sobre ataques en **FL**, se construyen e implementan tres ataques al modelo y un ataque a la privacidad.

8. Entorno experimental

Durante la revisión sistemática de la literatura sobre amenazas en FL, se observó una clara predominancia de estudios centrados en ataques contra la integridad del modelo (envenenamiento, *backdoor*, manipulación de gradientes, ...) frente a aquellos dirigidos a comprometer la privacidad de los datos. Por ejemplo, Bagdasaryan et al. [BVH⁺20] demuestran que los ataques de *backdoor* afectan gravemente el comportamiento del modelo con una sola ronda de entrenamiento.

Por ende, la prevalencia en la bibliografía de tres ataques al modelo frente a uno sobre la privacidad resulta coherente con la tendencia investigadora.

Ataque *Label Flipping*

Como se venía comentando, para evaluar la robustez del método de optimización de PD frente a ataques maliciosos, se ha diseñado una simulación controlada en la que participan diversos clientes, de los cuales alguno de ellos actuará de manera maliciosa mediante un ataque al modelo agregado de *label flipping*, [TTGL20].

En cada ronda, los clientes actualizan sus modelos utilizando sus propios subconjuntos de datos locales, y posteriormente las actualizaciones son agregadas en un servidor central mediante la estrategia de agregación discutida, CDP.

Esto es común para todos los participantes, salvo para el adversario, cuyos datos han sido manipulados; todas las etiquetas de sus datos de entrenamiento las reemplaza por etiquetas distintas de la original de manera aleatoria. Es decir, en lugar de usar las etiquetas verdaderas de sus datos, asigna una etiqueta aleatoria a cada muestra. Un ejemplo es la simulación mostrada en la Figura 8.4.

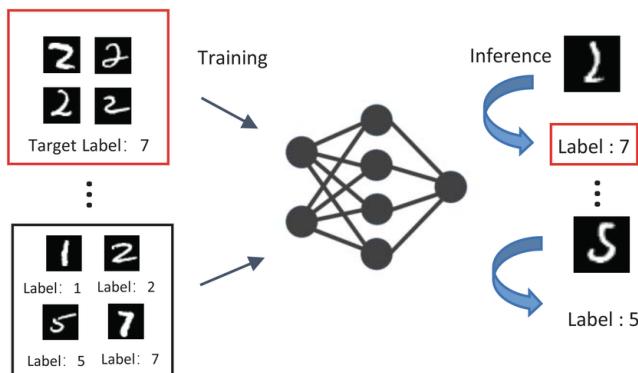


Figura 8.4.: Escenario de *Label Flipping* como ataque al modelo. Se etiquetan de manera maliciosa muestras de la clase 2 como muestras de la clase 7. Por tanto, en la inferencia el modelo confunde dichas muestras y las clasifica erróneamente. Imagen extraída de [RYD⁺22].

La arquitectura de los modelos de los clientes usada es una red convolucional compuesta por dos bloques convolucionales seguidos de capas completamente conectadas. El primer bloque consiste en una capa convolucional con 32 filtros de tamaño 5×5 , que opera sobre imágenes en escala de grises con un solo canal. A esta capa le sigue una función de activación tangente hiperbólica (*tanh*) y una operación de submuestreo mediante *max pooling* con una ventana de 2×2 . El segundo bloque aplica una segunda convolución con 64 filtros de tamaño 5×5 , seguida nuevamente por activación *tanh* y *max pooling* del mismo tipo. La salida de la

última capa de pooling se aplana y se conecta a una capa completamente conectada (lineal) de 1024 entradas y 200 unidades ocultas, también activadas con $tanh$. Finalmente, la red termina con una capa lineal de 200 a C salidas, donde $C = 10$ representa el número de clases de clasificación.

Por otro lado, como se adelantaba en la sección anterior, el conjunto de datos de entrenamiento de todos los clientes es el conocido MNIST, compuesto de imágenes de dígitos manuscritos del 0 al 9. Por tanto, el ataque del adversario consiste en cambiar etiquetas de los números a etiquetas erróneas.

Ataque gaussiano

El siguiente ataque es otro ataque al modelo, en esta ocasión un ataque *Gaussiano* [FCJG20]. Como se anticipaba en la presentación y clasificación de los tipos de ataques en **FL**, se trata de un ataque de *envenenamiento del modelo*. En lugar de modificar los datos de entrenamiento como en el caso anterior, ahora se corrompen o envenenan las actualizaciones de los clientes adversarios. Uno o varios clientes adversarios, en lugar de mandar las actualizaciones del entrenamiento como hacen los clientes benignos, simulan estas actualizaciones mediante una distribución *gaussiana*. Al agregar el modelo federado estas modificaciones, sufre un deterioro importante en su rendimiento.

Como ahora el conjunto de datos será el mencionado CIFAR10, la estructura neuronal también cambia. Se implementa para este ataque una red neuronal convolucional profunda diseñada en este caso para la clasificación de imágenes RGB en las 10 clases que ofrece el conjunto. Consta de tres bloques convolucionales. El primer bloque aplica dos convoluciones con 32 y 64 filtros de tamaño 3×3 , activaciones ReLU y una operación de *max pooling* que reduce las dimensiones espaciales a 16×16 . El segundo bloque emplea dos convoluciones con 128 filtros, seguidas de ReLU y *max pooling*, generando mapas de características de 8×8 . El tercer bloque realiza dos convoluciones con 256 filtros y una última capa de *pooling*, dejando una salida de tamaño $256 \times 4 \times 4$. Esta salida es *aplanada* y pasa por una sub-red totalmente conectada con capas lineales de 1024, 512 y finalmente 10 unidades de salida, intercaladas con activaciones ReLU, permitiendo la clasificación final. Al tratarse de un conjunto de datos algo más complejo, la arquitectura debe serlo también.

Ataque Backdoor

Retomando lo ya planteado en la sección de ataques al **FL**, los ataques *backdoor* pretenden *inyectar* una tarea secundaria en el entrenamiento del modelo federado. Se trata entonces de un ataque al modelo dirigido. Por ejemplo, en la experimentación, como se puede ver en la Figura 8.5, el cliente adversario introduce una modificación de las muestras de su conjunto de datos, con el objetivo de que el modelo detecte el patrón y clasifique dichas muestras como pertenecientes a una clase errónea.

Para reducir la agresividad del ataque, en general, la simulación no se hace con un envenenamiento íntegro del conjunto de datos del adversario, sino que se hace con una probabilidad. Además, esto se hace también para no desvelar el patrón de ataque, pues si se entrena el modelo con demasiadas instancias envenenadas, es fácil darse cuenta de que está teniendo lugar el ataque.

Para este experimento se reutiliza la arquitectura neuronal utilizada para el *label flipping*, pues la dificultad de FashionMNIST es muy similar a la de MNIST, compartiendo además las dimensiones de entrada de las imágenes. Además, también cuenta con 10 clases, por lo

8. Entorno experimental

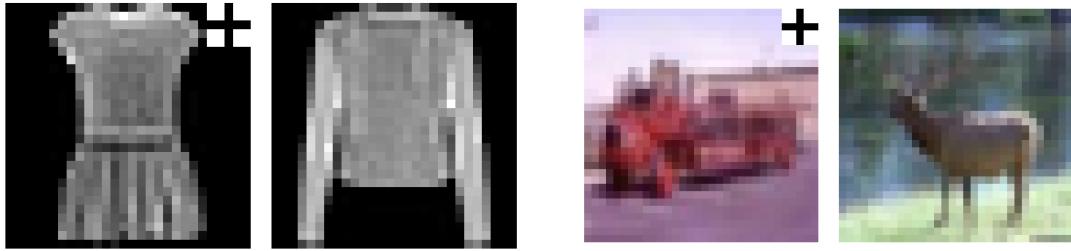


Figura 8.5.: El cliente adversario envenena una muestra de su conjunto de datos. Específicamente, añade un patrón de una cruz en negro sobre blanco para *confundir* al modelo. Además, se etiqueta maliciosamente la muestra envenenada como perteneciente a la clase 4, mientras que a la derecha se observa una muestra de la verdadera clase 4, para FashionMNIST y CIFAR-10.

que se puede volver a usar la misma red neuronal, obteniendo resultados de calidad en el entrenamiento.

Ataque de reconstrucción con GAN

Se trata de un ataque a la privacidad efectivo contra esquemas de **FL**, mediante el uso de una **GAN**. El ataque, descrito en [HAPC17], que tiene como siglas *DMUG*, se basa en las siguientes ideas principales:

- Un adversario que actúa como participante malicioso puede aprovechar el acceso a los parámetros compartidos para entrenar una red generativa **GAN** que produzca ejemplos similares a los datos privados de otros usuarios.
- La **GAN** nunca ve directamente los datos del usuario víctima, pero al interactuar con el modelo que actúa de discriminador compartido², puede aprender la distribución subyacente de los datos y generar ejemplos que “imiten” los datos originales.
- El adversario etiqueta estas muestras falsas como pertenecientes a una clase diferente (por ejemplo, una clase inventada c) y las incluye en su conjunto local de entrenamiento. Esto fuerza a la víctima a ajustar su modelo para distinguir entre la clase real (a) y la clase falsa (c), revelando aún más información sobre la clase privada.
- Este proceso se repite durante múltiples iteraciones del entrenamiento colaborativo. A medida que el modelo aprende, el GAN mejora su capacidad para generar ejemplos de la clase privada.

El ataque, como se menciona en [HAPC17], es efectivo incluso si los parámetros compartidos están ofuscados con mecanismos de **PD**. Esto se debe a que, mientras el modelo aprenda con éxito, el GAN también puede aprender. Sin embargo, como se aprecia en la Figura 8.6, la **PD** puede ofrecer cierta resistencia frente al ataque, empeorando el resultado de generación de la **GAN**.

²En una **GAN**, el **discriminador** es una red neuronal que intenta distinguir entre datos reales y datos falsos (generados por el generador). Su objetivo es maximizar la probabilidad de asignar correctamente una etiqueta de “real” o “falso” a cada entrada. A medida que el generador mejora en producir muestras realistas, el discriminador debe volverse más preciso para seguir diferenciándolas, lo que impulsa el aprendizaje conjunto entre ambas redes.

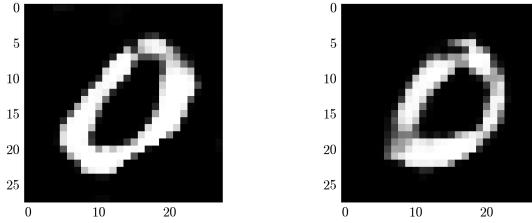


Figura 8.6.: Reconstrucción de la **GAN** tras 100 rondas de entrenamiento de **FL**. La **GAN** pretende reconstruir una instancia de la clase 0. A la izquierda el resultado para una aplicación de **PD** de forma estática durante las 100 rondas. A la derecha la aplicación ha sido dinámica, optimizando los parámetros de privacidad. Ver Sección 9.4.

8.3. Métricas empleadas

El seguimiento de las simulaciones es crucial para entender si están teniendo éxito o no. Por ello, se hace una monitorización de distintas métricas durante las rondas de **FL**, para luego ponerlas en común y poder analizar el funcionamiento de la propuesta hecha, como se hace en el siguiente capítulo.

Definimos previamente los siguientes términos, que jugarán un papel fundamental en la experimentación:

- (I) El desempeño del modelo (*accuracy* en los datos del servidor): En problemas de clasificación multiclas (nuestro caso), la *accuracy* se define como la proporción de predicciones correctas entre el total de muestras. Formalmente, se expresa como:

$$\text{Accuracy} = \frac{TP}{N}$$

donde:

- N es el número total de muestras.
- TP son los *True Positives*, o verdaderos positivos. Aquellas muestras que han sido predichas como la clase a la que pertenecen.

La *accuracy* toma valores en el intervalo $[0, 1]$, donde un valor de 1 indica que todas las predicciones fueron correctas. Como en este caso interesa saber el rendimiento del modelo federado se usa el conjunto de datos del servidor para medir esta métrica.

- (II) ϵ acumulativo por rondas: Se monitoriza este parámetro solo pues δ se optimiza durante el entrenamiento, pero no es tan decisivo a la hora de aplicar **PD**, pues como se comentaba en la definición de (ϵ, δ) -**PD**, se trata de un parámetro que da cierta holgura a la hora de aplicar la **PD**.
- (III) Éxito del ataque: En algunos casos de ataques al modelo, será lo mismo que el primer ítem, pues tienen como objetivo reducir el rendimiento del modelo colaborativo.

En vista de estos parámetros, se considerarán exitosos los experimentos cuando se cumpla que:

8. Entorno experimental

- Se consigue reducir el uso de presupuesto de privacidad, ϵ .
- Se consigue mitigar el éxito del ataque simulado.
- Se consigue mantener el nivel global de *accuracy*, sin que la **PD** afecte a la misma.

Por lo tanto, cualquier combinación de las casuísticas anteriores significará que la optimización de la **PD** está funcionando exitosamente.

Más concretamente, se listan ahora todas las métricas que juegan algún papel en la experimentación y que se miden y monitorizan al final de cada una de las rondas de comunicación entre el servidor y los clientes, aclarando cuáles se usarán para cada experimento:

- **Accuracy:** Se usa para todos los ataques. En el conjunto de prueba, que se encuentra en el servidor, se hace recuento de cuántas imágenes son clasificadas correctamente. Mide qué tan bien el modelo federado clasifica correctamente los datos.
- **Epsilon (ϵ) acumulado por rondas:** También usado para todos los ataques. Cada ronda se suma a la métrica el nivel de presupuesto gastado en la misma, teniendo así un control sobre el presupuesto total gastado hasta esa ronda. Además, esta métrica, como se veía en [7](#), se utiliza para garantizar que no se excede el presupuesto global, en la definición del problema de **PL**.
- **Distancia mínima:** Se usará para el último de los experimentos (DMUG). Se mide, para cada ronda, la distancia entre la reconstrucción de la **GAN** en ese momento y todas las muestras con etiqueta la misma que la clase a generar del conjunto de prueba. De todas estas distancias, se elige la mínima, computando así una forma de saber cuánto se ha acercado la reconstrucción del adversario a una muestra real.
- **Attack Success Rate (ASR):** Lo usaremos para el ataque *backdoor*. Mide el éxito del ataque o de la tarea secundaria. Lo denominaremos **ASR**, y medirá cómo de exitosa es la inyección de la tarea secundaria en el modelo agregado. Para ello, en cada ronda, se modificará el conjunto de prueba (conjunto del servidor) para hacer cómputo y observar cómo clasifica el modelo las imágenes envenenadas. A todas las muestras del conjunto, se les añade el patrón de ataque, y se comprueba cuántas de ellas clasifica el modelo agregado como la clase que el adversario pretende. Se calcula entonces el porcentaje de las imágenes mal clasificadas y esto será la métrica que medirá el desempeño del ataque del adversario.

Resumiendo lo anterior, en la simulación del ataque de *label flipping* y el ataque *gaussiano* se medirá tanto la *accuracy* como el ϵ acumulado. La métrica que mide el éxito del ataque al modelo, en este caso, es la misma que el rendimiento del modelo federado en sí. Por otro lado, para el ataque *backdoor*, se usará el **ASR** también, para poder medir el éxito de la tarea secundaria, que será el reconocimiento de un patrón inyectado en las imágenes. Para el último ataque, el de reconstrucción con **GAN**, se monitorizará *accuracy*, ϵ acumulado y la distancia mínima definida.

8.4. Detalles de implementación

En esta sección se desglosan y detallan las herramientas específicas utilizadas para la experimentación, desde *software* hasta *hardware*. En trabajos experimentales como es justamente

este, la razón de hacer esto es la replicabilidad. Experimentos que no puedan ser fácilmente replicables y reproducibles no deberían tener credibilidad alguna.

- Para la implementación de los escenarios simulados de ataque y las propuestas de defensa se usa como base la librería especializada *Flexible-FL*, [HJLAG⁺[25](#)], que proporciona un espacio de trabajo ideal en *Python* para la simulación de todo tipo de entrenamientos de *FL*. Ofrece una gran modularidad y adaptabilidad, con una capacidad completa de configuración.

Los ataques se implementan en torno a esta librería y, en concreto, el módulo *FLEX-Clash* para incorporar envenenamientos en los datos y las actualizaciones de los clientes. También es con el uso de este módulo con el que se puede configurar la *CDP*, pues ofrece métodos de defensa comunes ya implementados.

- Para la aplicación de la *PD-local* se hace uso de *Opacus* [[YSS⁺\[21\]\(#\)](#)], una librería de *Meta AI*. Se trata de una biblioteca para *PyTorch* que permite entrenar modelos de aprendizaje automático con *PD* de manera eficiente. Además, también ofrece una herramienta para calcular un multiplicador de ruido σ a partir de ϵ y δ , que ha sido crucial para la aplicación de la *CDP*.
- Por último, se hace uso de la librería *SciPy* con el módulo *linprog*, para la correcta definición y resolución de los problemas de *PL*. Para cada ronda, se implementa un método en el que se resuelve el problema como se describe en la Sección [7](#), dando los valores de ϵ y δ para la ronda siguiente.

Se llevan a cabo un total de 8 experimentos, pues para cada uno de ellos se emplea la *PD* de manera estática y de manera dinámica. No obstante, una vez se tiene la versión estática, es fácil replicar el escenario con los mismos parámetros y configuraciones para aplicar una optimización de la privacidad, por lo que se hablará de 4 experimentos a lo largo de su exposición para el correspondiente análisis.

Es decir, la verdadera dificultad de la experimentación ha sido la implementación de los ataques, así como la familiarización con las librerías y herramientas que no son tan conocidas, como *FLEX* y *Opacus*.

En cuanto a los cuatro experimentos llevados a cabo, se muestra ahora en la Tabla [8.2](#) la configuración final para cada uno de ellos.

	<i>Label Flipping</i>	<i>Gaussiano</i>	<i>Backdoor</i>	<i>GAN (DMUG)</i>
Clientes	10	10	10	2
Adversarios	1	1	2	1
Rondas	100	100	100	100
Agregación	<i>CDP</i>	<i>CDP</i>	<i>CDP</i>	<i>PD-local</i>
Prob. Envenenamiento	–	–	0.3	–

Tabla 8.2.: Configuración de los cuatro experimentos llevados a cabo. Se detallan las características específicas de cada uno de ellos, para una mayor replicabilidad.

Para la experimentación adicional sobre el número de atacantes mencionada, se realizan ejecuciones para 2, 3 y 5 atacantes, frente a los propuestos en la Tabla [8.2](#) como base experimental.

Para todo el desarrollo de la experimentación, se ha hecho un uso intensivo de los recursos de cómputo proporcionados por *Google Colab*, específicamente sus unidades de procesamiento

8. Entorno experimental

gráfico (*GPU*). Estas han permitido acelerar significativamente los experimentos, en tareas de entrenamiento distribuido de imágenes y la simulación de ataques adversarios. Gracias al acceso gratuito y relativamente sencillo a este entorno, ha sido posible realizar múltiples pruebas en paralelo, ajustar hiperparámetros de forma iterativa y evaluar el rendimiento de los modelos en tiempos razonables.

A nivel de *hardware*, se ha usado una ***GPU NVIDIA T4***, ofrecida justamente por *Google Colab* y cuyas especificaciones son las siguientes: basada en la arquitectura ***NVIDIA Turing***, integra 2560 *CUDA Cores* (FP32) y 320 *Tensor Cores* para optimizar cargas de trabajo de **ML**. Dispone de 16 GB de memoria GDDR6 con ECC, ofreciendo un ancho de banda de hasta 320 GB/s.

Para el tiempo de cómputo y de uso de *GPU*, se hace un resumen en la Tabla 8.3. Se muestran los tiempos necesarios para la ejecución de los experimentos de manera aproximada, pues todos ellos menos *DMUG* se ejecutan en los tres *datasets*, además de realizar experimentos adicionales variando el número de adversarios.

Ataque	Tiempo por ejecución	N.º ejecuciones	Tiempo total
LabelFlipping	45min	≈ 40	30h
Gaussian	1h10min	≈ 15	17h30min
Backdoor	50min	≈ 25	20h50min
DMUG	1h30min	≈ 30	40h
Total acumulado	—	—	110h

Tabla 8.3.: Tiempo de cómputo empleado en los experimentos del TFG. El tiempo es aproximado, pues los tres primeros fueron lanzados en los tres conjuntos de datos explicados, variando de unos a otros el tiempo de ejecución.

Todo el código implementado, junto a los resultados obtenidos, puede encontrarse en el siguiente enlace a la plataforma *Github*³.

³<https://github.com/gullogullito/DP-LinProg-Optimization>

9. Análisis de los resultados experimentales

Conociendo al detalle la experimentación realizada, se procede a mostrar los resultados obtenidos y al posterior análisis de los mismos. Esta sección está dedicada a analizar con precisión la experimentación seguida, en la que se pone a prueba la optimización hecha de los parámetros de la **PD** frente a un escenario en el que esta se aplica continuamente de manera estática.

El orden en que se muestran los experimentos no ha sido escogido al azar. La línea que siguen los resultados es una creciente; de menos a más visual en el resultado obtenido. Sin embargo, cada uno de ellos tiene alguna particularidad, haciendo su análisis igual de interesante.

En general, en los resultados de los experimentos trataremos de explicar tanto la evolución del entrenamiento y la privacidad a lo largo de las rondas, como el resultado final, que también es de gran relevancia.

9.1. Ataque *Label Flipping*

Como se describe en el escenario propuesto en la Tabla 8.2, 10 clientes entrenarán durante 100 rondas de forma conjunta para agregar un modelo federado en el servidor. En este caso, el ataque, en principio, no es demasiado agresivo, pues solo actuará como adversario uno de los clientes, enviando actualizaciones entrenadas en datos envenenados en cada ronda de comunicación.

La razón detrás de esto último es tratar de simplificar el problema y tener más control sobre el efecto que tiene la presencia de un solo atacante en el entrenamiento conjunto.

Como se muestra en la Figura 9.1, los resultados obtenidos apoyan la hipótesis: el aplicar la **PD** de una forma *más inteligente* da como resultado un modelo federado que tiene un nivel de *accuracy* en el conjunto de prueba prácticamente igual que para la aplicación estática. Sin embargo, para el enfoque dinámico, el nivel de privacidad es mayor, pues el presupuesto de privacidad es aplicado durante las 100 rondas es menor en los tres casos.

Además, se muestra en la Figura para MNIST 9.1a, un fenómeno que también podría considerarse una ventaja del método propuesto. Al principio del entrenamiento, cuando la pérdida del modelo es alta, **se relajan las condiciones de privacidad**, permitiendo un mejor entrenamiento antes. Cuando se tiende a la estabilidad, es cuando se aplican parámetros más estrictos para la **PD**. Este repunte en la *accuracy* en rondas más tempranas deja en evidencia que realmente el algoritmo de optimización está funcionando. Esta idea es clave para conseguir el *trade-off* que se viene comentando a lo largo del trabajo entre privacidad y rendimiento del modelo.

En este primer experimento, el resultado que se buscaba, en términos globales, se consigue para todos los conjuntos de datos. En particular, para FashionMNIST, el ahorro de presupuesto de privacidad es de 22.1 unidades (Tabla 9.1).

No obstante, el orden de presentación de los experimentos se hace en función de la eficacia obtenida. Los siguientes experimentos han dado mejores resultados, a lo cual se le dará una explicación en la correspondiente presentación de los mismos. Sin embargo, en este

9. Análisis de los resultados experimentales

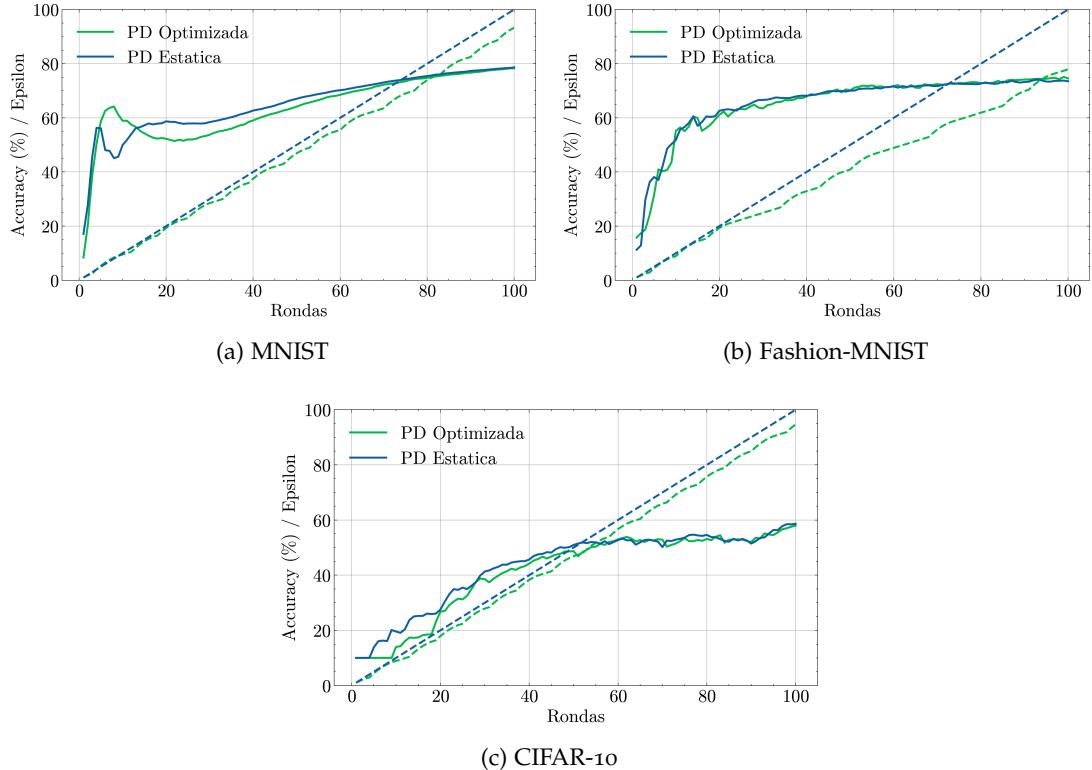


Figura 9.1.: En cada gráfica se compara la aplicación dinámica de PD frente a la versión estática con $\epsilon = 1.0$ fijo. La línea continua muestra la *accuracy* de ambos enfoques, mientras que la discontinua indica el ϵ acumulado. (a) MNIST, (b) Fashion-MNIST y (c) CIFAR-10. En los tres casos se observa un ahorro de presupuesto de privacidad sin pérdida de rendimiento.

primer experimento, sí que vemos de manera visual cómo afecta el uso de esta técnica *más inteligente* de aplicación de privacidad y, aunque en pequeños detalles, vemos que, según los criterios que definimos previamente, el experimento ha sido exitoso (misma *accuracy*, mayor privacidad) para los tres conjuntos de datos.

Cabe mencionar que la proporción entre adversarios y clientes benignos no tiene por qué corresponder con la realidad. Puede que en un escenario real, haya 100 clientes, de los cuales 20 de ellos sean adversarios. Sin embargo, para comprobar la validez y el funcionamiento de la técnica de optimización propuesta, basta con hacer una única comparación: entre la aplicación estática y la dinámica de la PD, como se hace en la Tabla 9.1.

De manera adicional y como se anticipaba en la sección anterior, se muestra ahora el resultado de variar, tanto en el caso estático de aplicación de PD como en el dinámico, el número de atacantes, cubriendo los valores 2,3 y 5 para el conjunto de datos MNIST. El resultado se puede ver en la Figura 9.2.

El objetivo de la visualización de esta comparativa es poner en juicio la validez y resiliencia que tiene el método propuesto de optimización de privacidad frente a escenarios donde los ataques son más agresivos o numerosos.

Se observa en ambos casos, sobre todo en el caso de la aplicación dinámica, un peor

Ronda	PD Estática		PD Optimizada	
	Accuracy (%)	ϵ	Accuracy (%)	ϵ
MNIST				
5	56.14	5.0	58.79	5.4
25	57.92	25.0	52.04	23.04
50	67.03	50.0	64.13	46.90
75	74.28	75.0	73.48	69.0
100	78.64	100.0	78.28	93.3
FashionMNIST				
5	38.15	5.0	31.46	4.4
25	63.95	25.0	63.25	22.4
50	70.68	50.0	70.12	40.9
75	72.58	75.0	72.92	59.40
100	73.50	100.0	74.55	77.9
CIFAR-10				
5	13.81	5.0	10.0	4.4
25	35.49	25.0	31.24	22.4
50	50.95	50.0	48.69	46.9
75	53.59	75.0	53.42	71.2
100	58.63	100.0	57.98	94.7

Tabla 9.1.: Resumen por rondas de la comparación de PD estática con la PD optimizada en un ataque de *label flipping* en tres datasets: MNIST, FashionMNIST y CIFAR-10. Se monitoriza tanto el rendimiento del modelo agregado o federado (*accuracy* %) como el presupuesto acumulado por rondas (ϵ acumulado).

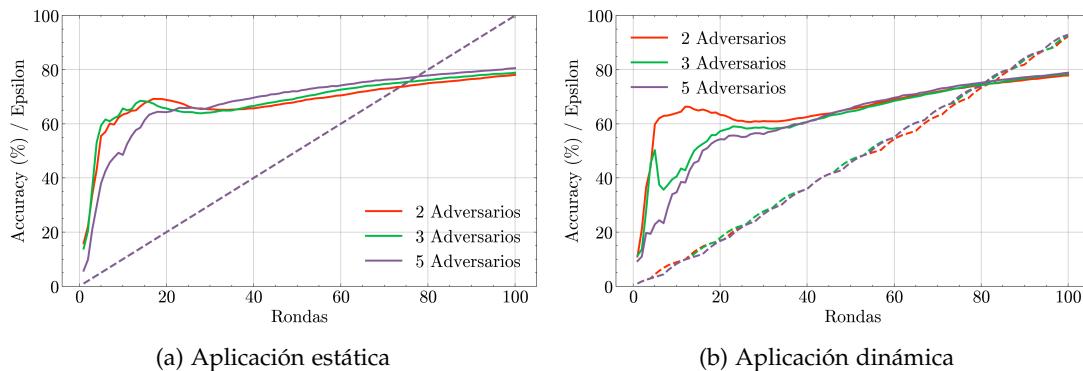


Figura 9.2.: Comparativa simulando un número distinto de adversarios, en el conjunto de datos MNIST. A la izquierda el caso estático y a la derecha el dinámico. Las líneas continuas representan *accuracy* y las discontinuas ϵ acumulado.

entrenamiento en épocas tempranas. Aún cuando no se advierte con absoluta nitidez, en el caso de los 5 atacantes, la privacidad se restringe mucho más. Esto es señal de estar aplicando la filosofía del algoritmo correctamente. Justamente por esto, el entrenamiento es mucho peor. Sin embargo, para este caso, se puede concluir que el resultado es equivalente al caso estático, al menos a largo plazo. Además, se ahorra también presupuesto de privacidad en todos los casos de optimización dinámica. Podemos afirmar que el método es fuerte o resiliente frente

9. Análisis de los resultados experimentales

a un mayor número de atacantes.

9.2. Ataque gaussiano

Este ataque al modelo, en el que se proponen unas configuraciones exactamente iguales a las del ataque anterior, se pretende comprobar de nuevo el funcionamiento del algoritmo propuesto. En este caso, cómo de resiliente es frente a un ataque en el que los clientes mandan actualizaciones que no se corresponden con un entrenamiento real. Se adelanta aquí que esto no afectará tanto al rendimiento del algoritmo, sino al del modelo agregado. Por tanto, una vez más, se trata de un experimento exitoso.

En cuanto al entrenamiento del modelo, si se observan los resultados de la Tabla 9.2, se puede ver que, aunque para MNIST y FashionMNIST el rendimiento es muy bueno, el nivel de *accuracy* para CIFAR-10 no es demasiado alto. En una tarea de clasificación como esta, para este conjunto de datos y usando la red definida previamente, se podría esperar en torno al 70-80 % para la métrica.

Ronda	PD Estática		PD Optimizada	
	Accuracy (%)	ϵ	Accuracy (%)	ϵ
MNIST				
5	95.15	5.0	94.71	3.4
25	98.72	25.0	98.78	18.2
50	99.09	50.0	99.08	45.7
75	99.13	75.0	99.17	74.2
100	99.17	100.0	99.22	99.3
FashionMNIST				
5	80.17	5.0	79.25	3.2
25	87.57	25.0	87.53	24.6
50	89.32	50.0	89.17	50.3
75	90.14	75.0	90.14	77.2
100	90.73	100.0	90.4	98.9
CIFAR-10				
5	10.0	5.0	12.88	3.4
25	34.51	25.0	34.83	23.0
50	49.76	50.0	48.36	46.3
75	59.07	75.0	57.21	68.8
100	62.3	100.0	63.12	92.3

Tabla 9.2.: Resumen por rondas de la comparación de PD estática con la PD optimizada en un ataque *gaussiano* en los tres conjuntos de datos. El ahorro no es demasiado significativo, pero sí es notable en épocas tempranas del entrenamiento. Para CIFAR-10 (abajo) el ahorro es mayor y el rendimiento del modelo es muy similar a largo plazo.

Esto se debe al ataque que se está realizando; se simula que uno de los clientes (adversario) envía actualizaciones totalmente aleatorias, que siguen una distribución *gaussiana*. El ruido que introduce este ataque es el que se pretende mitigar con el uso de CDP. Sin embargo, se reitera que no es relevante en este caso el rendimiento del modelo agregado para la tarea, sino la comparación entre las dos técnicas de aplicación de PD, como se puede ver en la

Figura 9.3.

De hecho, si nos fijamos en los resultados para la ronda 100 para el conjunto CIFAR-10, es mayor incluso la *accuracy* para el segundo caso, el de la **PD** optimizada. Esto no tiene por qué deberse necesariamente a la solución propuesta, pues el objetivo no es mejorar el rendimiento del modelo. En este caso en concreto, se puede explicar con la aleatoriedad del entrenamiento de las redes neuronales. La inicialización de los pesos y la elección de otros hiperparámetros relevantes puede hacer que ocurran estas variaciones. Lo realmente relevante es el ahorro en privacidad a lo largo de las rondas, como es el caso del experimento en CIFAR-10, ahorrando hasta 7.7 unidades de presupuesto de privacidad.

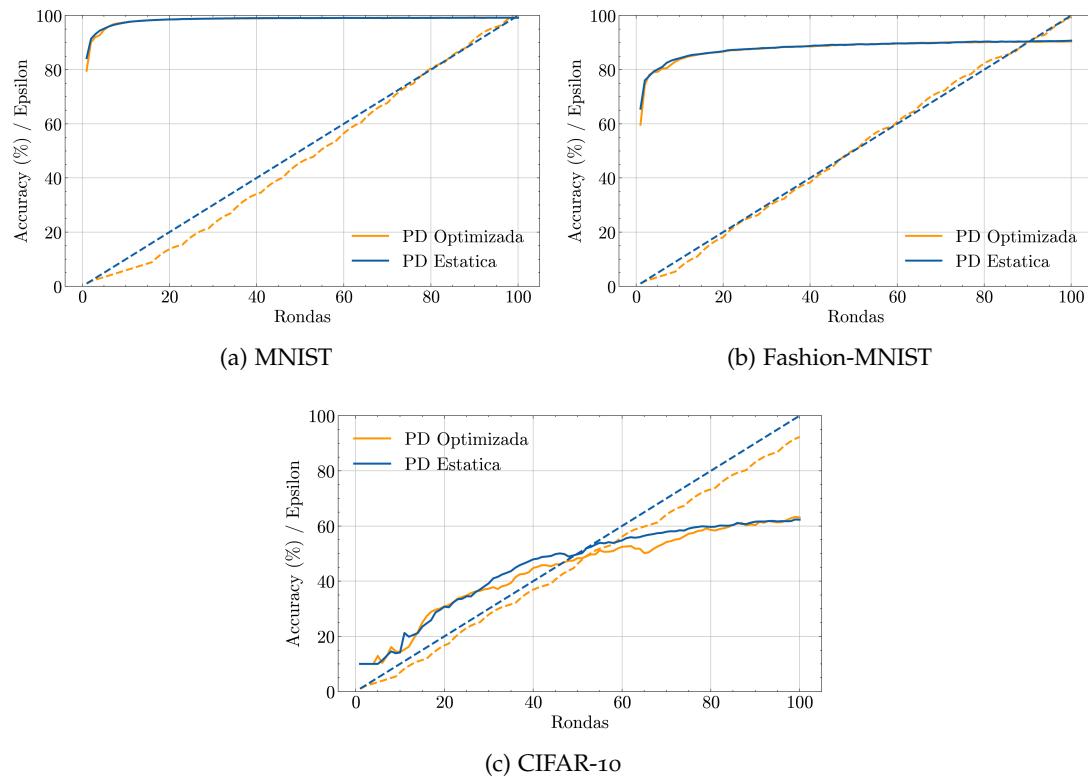


Figura 9.3.: De nuevo, arriba a la izquierda los resultados para el ataque *gaussiano* en MNIST, arriba a la derecha para FashionMNIST y abajo para CIFAR-10. Se observa también un ahorro significativo de presupuesto de privacidad para el experimento en MNIST y CIFAR-10, sobre todo. Además, siempre se mantiene un rendimiento similar del modelo federado en el conjunto de prueba.

También ocurre que para MNIST, en las primeras rondas de entrenamiento, el ahorro es bastante significativo. Esto se debe al funcionamiento de la optimización realizada; al principio el entrenamiento es muy bueno, por lo que se restringe la privacidad, a cambio de rendimiento en el modelo. Cuando la *accuracy* tiende a estabilizarse, el algoritmo busca variaciones más altas de relajamiento de privacidad pues lo hace con respecto a la variación de la pérdida del modelo. Como esta se mantiene constante, la privacidad aplicada es menor para tratar de seguir obteniendo un mejor rendimiento (aunque en este caso ya no es posible, pues está cerca del 100 % de *accuracy*). Algo muy similar ocurre para el experimento en

9. Análisis de los resultados experimentales

FashionMNIST.

Se puede apreciar también el efecto de usar una optimización de este tipo; el entrenamiento puede ser algo más irregular (ver el caso de CIFAR-10). Al ir variando el ruido añadido con **CDP**, hay rondas en las que el rendimiento decae. Cuando esto ocurre, se *relaja* la aplicación de privacidad, dando lugar a más irregularidad y altibajos en el rendimiento del modelo. No obstante, vemos que el resultado final es prácticamente el mismo, en la Tabla 9.2.

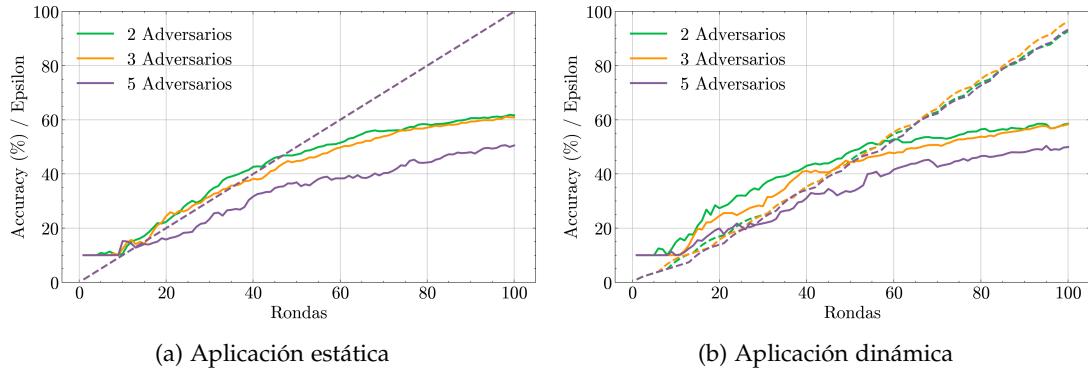


Figura 9.4.: Comparativa simulando un número distinto de adversarios para el ataque *gausiano* en el conjunto de datos CIFAR-10. De nuevo, a la izquierda el caso estático y a la derecha el dinámico. Las líneas continuas representan *accuracy* y las discontinuas *e* acumulado. Aparentemente, el método de optimización sigue funcionando y muestra resiliencia al aumento de atacantes en el escenario simulado.

9.3. Ataque Backdoor

El siguiente en la lista de ataques al modelo es el ataque *backdoor*, en el que, como ya se explicaba, se trata de inyectar una tarea secundaria en el entrenamiento del modelo. En este caso, se pretende hacer que el modelo prediga como cierta etiqueta aquellas muestras con un patrón determinado. Este es, por tanto, uno de los dos experimentos en los que se mide una nueva métrica. Se trata de uno de los dos ataques en los que el éxito no se mide implícitamente en el deterioro del modelo agregado, sino que existe una forma para monitorizar cómo de efectivo está siendo el ataque. Se usará el **ASR**.

Al ser un ataque que no pretende corromper de manera directa el modelo federado, veremos, en general, métricas de rendimiento del modelo del servidor mayores que las que veíamos en los experimentos anteriores.

Se puede ver en la Tabla 9.3 cómo evoluciona en este caso el entrenamiento, para los distintos conjuntos de datos utilizados. Podemos sacar algunas conclusiones directamente de aquí:

- En rondas más tempranas y a lo largo de las 100 que tienen lugar, el éxito del ataque se mantiene la mayoría del tiempo por debajo en el caso de la aplicación dinámica. En el caso del experimento en CIFAR-10, como la privacidad aplicada es menor en pos de tener un mejor rendimiento en las primeras rondas, observamos altos valores de **ASR** para el caso de **PD** optimizada. No obstante, conforme trascurren el resto de rondas

Ronda	PD Estática			PD Optimizada		
	Accuracy (%)	ϵ	ASR (%)	Accuracy (%)	ϵ	ASR (%)
MNIST						
5	95.08	5.0	10.2	95.48	3.40	10.56
25	98.78	25.0	12.99	98.79	18.8	12.57
50	99.16	50.0	29.22	99.09	47.3	28.16
75	99.33	75.0	53.39	99.12	74.2	52.79
100	99.34	100.0	69.97	99.16	98.9	72.25
FashionMNIST						
5	79.22	5.0	13.14	80.36	3.40	12.4
25	87.77	25.0	30.21	87.82	19.40	28.48
50	89.61	50.0	61.46	89.69	43.70	53.84
75	90.36	75.0	83.05	90.36	70.20	81.37
100	90.70	100.0	90.74	90.74	96.30	89.47
CIFAR-10						
5	10.0	5.0	0.0	10.0	3.70	0.0
25	38.32	25.0	19.79	29.4	10.6	3.28
50	52.26	50.0	22.8	49.1	46.1	28.74
75	59.47	75.0	43.98	58.36	67.6	44.36
100	65.3	100.0	76.79	62.8	89.9	73.74

Tabla 9.3.: Resumen por rondas de la comparación de PD estática con la PD optimizada en un ataque *backdoor*, mostrando también el ASR. El ahorro de presupuesto no es tan notable como en otros experimentos, pero el enfoque dinámico consigue mitigar el ataque sobre todo en rondas tempranas, manteniendo siempre el modelo federado un rendimiento similar.

esto se va estabilizando, terminando la métrica por encima para el caso de aplicación estática de privacidad.

- La *accuracy* se mantienen prácticamente igual para los dos casos en los tres conjuntos de datos, como se buscaba en la experimentación. En el experimento hecho en CIFAR-10, sí se observa cierta pérdida de rendimiento para el caso de la PD optimizada, pero no es demasiado significativa.
- En este experimento el ahorro hecho de presupuesto de privacidad es más notable en el último caso, justificando el ítem anterior. Para CIFAR-10, se ahoran hasta 10.1 unidades de presupuesto.

Todo esto se resume en lo siguiente: para el ataque *backdoor*, se consigue que el modelo tenga una mayor resiliencia a lo largo del tiempo a la tarea secundaria inyectada, dando peores resultados durante la mayoría de las rondas. Además, otro objetivo cumplido es el de mantener el rendimiento del modelo. Bien es cierto que el presupuesto ahorrado y, por tanto, la privacidad aplicada no es tan remarcable. Esto no quita que de los tres ítems que se daban para poder clasificar un experimento como exitoso, aquí se cumplen dos de ellos.

Lo descrito previamente también se refleja en la Figura 9.5. Se observa un entrenamiento muy similar para los dos modelos agregados, en los tres experimentos para cada *dataset*. Igual con la particularidad de que, al principio, el entrenamiento para el caso de aplicación dinámica de privacidad es algo más irregular que para la estática. Esto se debe, como se

9. Análisis de los resultados experimentales

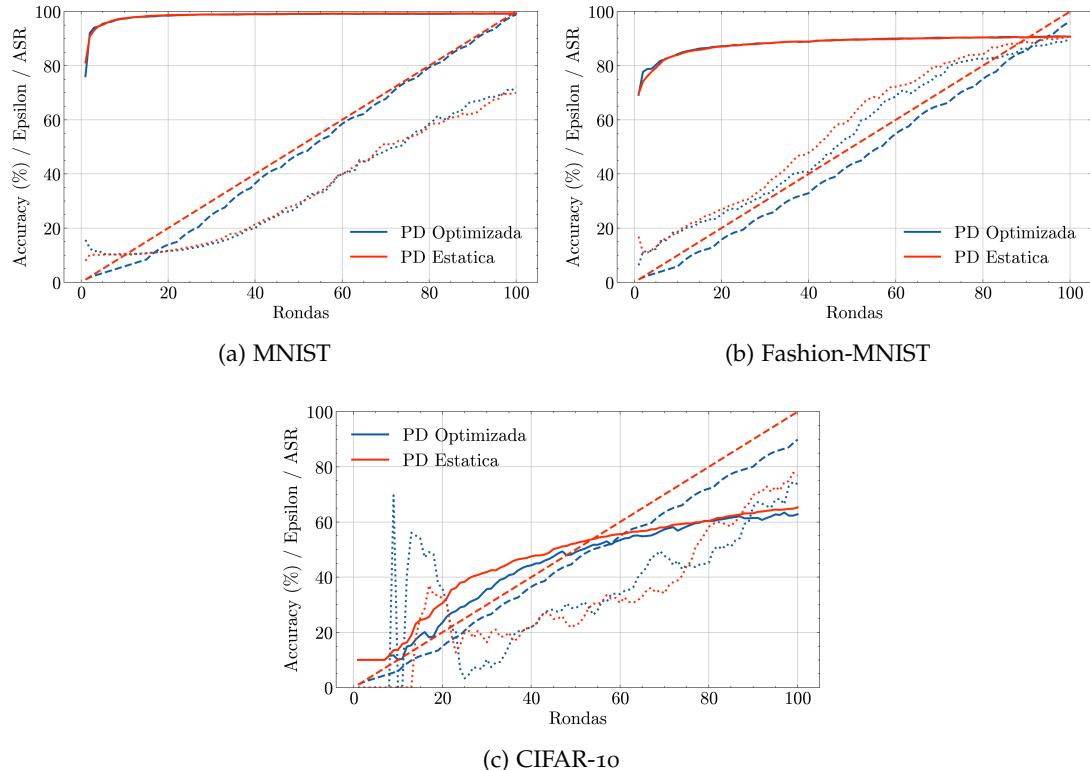


Figura 9.5.: Comparativa entre aplicación estática y dinámica para el escenario del ataque *backdoor* para MNIST (arriba izquierda), FashionMNIST (arriba derecha) y CIFAR-10 (abajo). Las líneas continuas representan el rendimiento del modelo agregado (*accuracy*), las discontinuas el ϵ acumulado por rondas y las líneas punteadas representan el **ASR**, es decir, el éxito del modelo. Cuanto más alto, significa que el modelo ha aprendido y asimilado mejor la tarea secundaria inyectada.

comentaba en el ataque *gaussiano*, a que el algoritmo está tratando de ajustar y aumentar la privacidad aplicada, debido al alto rendimiento del modelo.

La aplicación de **PD** es algo más estable aquí que en el resto de la experimentación, seguramente también por el buen rendimiento del modelo. Por último, en cuanto al éxito del ataque, se confirma lo que se venía comentando para la Tabla 9.3; se observa un peor resultado en general para el ataque en el caso de **PD** optimizada. Por ejemplo, si nos fijamos en la ronda 40 de entrenamiento para FashionMNIST, el caso estático ya presenta un **ASR** del 45 %, aproximadamente, mientras que para el caso dinámico todavía está en 40 %.

Por último, se muestra en 9.6, como para los dos ataques anteriores, un análisis para el experimento adicional en el que se evalúan y comparan los dos enfoques (estático y dinámico) para distintos valores de número de atacantes (3,4 y 5). En este caso queda en evidencia que el nivel de privacidad aplicado es menor cuando el número de atacantes es 5.

Se observa que para mantener un rendimiento igual en todos los casos, se sacrifica privacidad conforme aumentan los adversarios. Para este caso, el algoritmo no parece ser tan resiliente, pues prioriza el rendimiento del modelo, restando así privacidad en escenarios más agresivos para los clientes benignos.

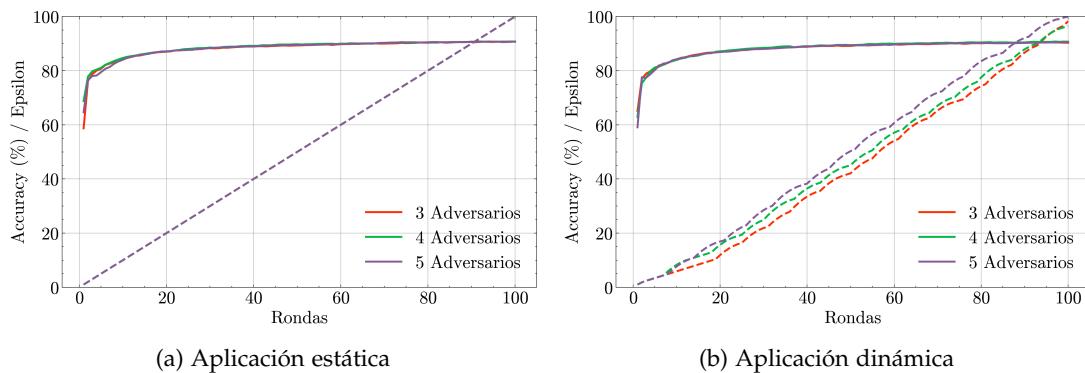


Figura 9.6.: Comparativa simulando un número distinto de adversarios para el ataque *backdoor* en el conjunto de datos CIFAR-10. De nuevo, a la izquierda el caso estático y a la derecha el dinámico. Las líneas continuas representan *accuracy* y las discontinuas *e* acumulado. En esta simulación, el método de optimización va perdiendo potencia conforme aumenta el número de atacantes, consumiendo en el caso de 5 atacantes las 100 unidades de presupuesto iniciales.

9.4. Ataque de reconstrucción con GAN

Para este último ataque, un cliente adversario, con datos de las clases $(5, 6, 7, 8, 9)$, trata de reconstruir una muestra de la clase 0 del cliente benigno con las clases $(0, 1, 2, 3, 4)$. La GAN va generando imágenes, las etiqueta con la clase 0, y las va incluyendo en su conjunto de datos de entrenamiento. Así, poco a poco, tras 100 rondas se tienen reconstrucciones como las que se veían en la Figura 8.6.

En este caso, se opta por un solo cliente adversario y otro benigno pues, el entrenamiento de la **GAN** es bastante costoso, tanto en recursos como en tiempo. Además, cuantos más clientes haya en la agregación de los modelos, más información revelan de sus datos de entrenamiento. Es por esto que con 2 clientes únicamente, es suficiente para poder llevar a cabo la comparativa que nos concierne.

También, como se comentaba en la sección de descripción del entorno experimental, la implementación de este ataque es bastante sensible a cambios, tanto en los datos de entrenamiento como en el número de nodos o clientes en el escenario federado. Es por esto que no se realiza la experimentación ni para CIFAR-10 ni Fashion-MNIST, así como tampoco se realiza la experimentación adicional variando el número de adversarios.

Por otra parte, en concordancia con el ataque anterior, como en este caso el objetivo del ataque no es específicamente alterar el rendimiento del modelo (se trata de un ataque a la privacidad), se ha de medir una tercera métrica que dé una idea de cómo está funcionando el ataque. En concreto, es la distancia mínima que se comentaba en la sección de métricas empleadas.

La aplicación inteligente de PD en este experimento parece haber sido la más exitosa pues, como se muestra en la Figura 9.7, la comparativa es bastante clara. Se ahorra hasta un 37.1 de presupuesto de privacidad de los 100 disponibles, mientras que el rendimiento se mantiene igual y la reconstrucción de la GAN es peor. Esto cumple justamente con los tres ítems que se buscaban encontrar en la experimentación.

Además, vemos que a lo largo de las rondas, la distancia mínima a los datos reales es mayor en el caso de la aplicación de la PD dinámica, sobre todo en las rondas más tempranas.

9. Análisis de los resultados experimentales

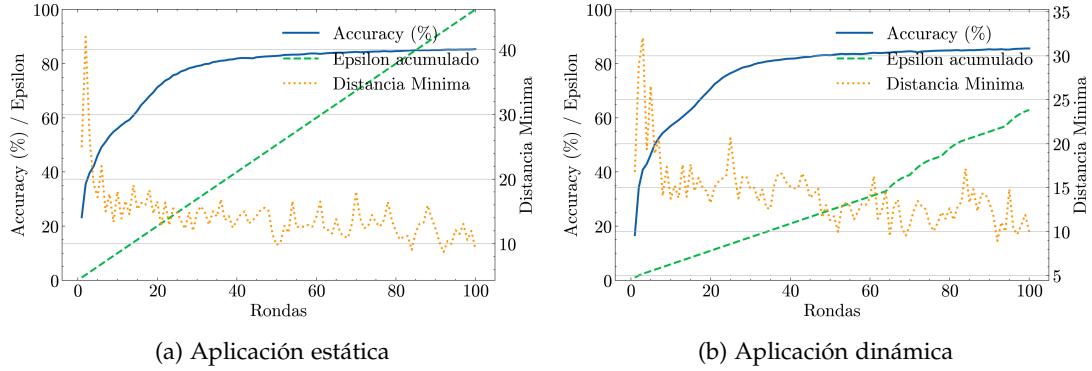


Figura 9.7.: Comparativa de privacidad estática/dinámica en DMUG. Es el caso más pronunciado de ahorro de presupuesto, pasando de 100 a solo 62.9. Además, la defensa parece mostrar una mayor resiliencia al ataque a lo largo de todas las rondas de comunicación. El rendimiento del modelo federado es el mismo en los dos casos.

Parece que en las primeras rondas el nivel de resiliencia es mayor, pues si nos fijamos en el ϵ aplicado, este es menor que para el caso estático. Véase la Tabla 9.4.

Ronda	PD Estática			PD Optimizada		
	Accuracy (%)	ϵ	Distancia Min	Accuracy (%)	ϵ	Distancia Min
5	46.1	5.0	17.20	46.48	3.40	26.57
25	76.19	25.0	14.80	76.49	13.4	20.64
50	82.87	50.0	9.81	83.08	25.9	11.30
75	84.49	75.0	13.29	84.65	44.4	10.87
100	85.36	100.0	9.23	85.63	62.9	10.01

Tabla 9.4.: Resumen por rondas de la comparación de PD estática con la PD optimizada en un ataque de reconstrucción con una GAN, para el conjunto de datos MNIST.

Las razones por las que el efecto de ahorro de presupuesto pueda ser mayor en esta casuística pueden ser varias. Sin embargo, la que más parece explicar la realidad es que, al tratarse de un ataque a la privacidad y usar PD-local, el control sobre el ahorro se realiza en los clientes, no en el servidor. Esto proporciona una mayor trazabilidad y dominio sobre los modelos locales antes de ser agregados. Para los ataques al modelo, al usarse CDP, se está haciendo una defensa a nivel global, por lo que los efectos de la optimización pueden verse mitigados por esto.

El estudio demuestra que la optimización lineal adaptativa de PD-local es especialmente efectiva frente a ataques de reconstrucción; conserva la precisión, reduce significativamente el gasto de privacidad y retraza la convergencia de la GAN del adversario. El ahorro del 37.1 de presupuesto confirma empíricamente el potencial del enfoque para escenarios donde la privacidad se gestiona de forma descentralizada.

Parte III.

Conclusiones y Trabajos futuros

10. Conclusiones

Este Trabajo de Fin de Grado ha abordado el problema de la privacidad en entornos sensibles de **FL**, proponiendo un enfoque basado en la optimización dinámica de la misma mediante la **PL**. La idea central ha sido regular dinámicamente el nivel de perturbación (a nivel de cliente o de servidor) para cada ronda, resolviendo iterativamente un problema de optimización que balancea precisión, privacidad y presupuesto total disponible. Los objetivos principales propuestos y cumplidos han sido simular ataques en un escenario de **FL** donde se aplique una optimización de la privacidad, tratando de mitigar el efecto de los ataques mientras se mantiene el rendimiento del modelo federado.

Para realizar este análisis de optimización, ha sido necesario sentar unas bases sólidas dentro de este área, empezando por fundamentos básicos del Álgebra Lineal, Análisis Funcional y conjuntos afines y convexos. Todo esto nos llevó al enunciado y demostración del Teorema de Minkowski-Carathéodory, que resalta la importancia del papel que juegan los puntos extremos en la optimización. A continuación, y de forma natural, se realizó un estudio exhaustivo de los **PPL**, así como un análisis completo y minucioso del método Simplex (haciendo siempre hincapié en su eficiencia computacional).

Para cerrar el tema de la optimización, dando completitud al trabajo, se introdujeron los **PPC**, desarrollando un método interesante para la resolución de los mismos. Se concluyó con la prueba del Teorema de Krein-Milman y el Principio del máximo de Bauer. Estos resultados complementan a la perfección la parte desarrollada sobre optimización, pues abren una vía de trabajo muy interesante que se comentará en la sección posterior.

Pasando a la segunda parte del trabajo, se comenzó con un preámbulo necesario que sería el eje central del trabajo: el aprendizaje. Se sientan las bases comunes y la teoría clásica de **ML** y **DL**. Es en esta línea en la que también se explicaron los fundamentos de la teoría del **FL**, herramienta crucial en el mundo de la privacidad en el aprendizaje. Justamente dentro de este contexto, se explicó y formalizó el concepto de **PD**, cuyo desarrollo teórico brinda una interpretabilidad íntegra en la aplicación práctica del trabajo.

Además, en el contexto del **FL**, se introdujo y se detalló una taxonomía completa de los posibles ataques en un escenario de entrenamiento colaborativo (siguiendo la clasificación de [RBJLL⁺23]), para poder entender los implementados en este trabajo.

Todo esto llevó a la parte final del trabajo y el culmen del desarrollo tanto de la optimización como del **FL**: la experimentación. En ella, se ha demostrado que una aplicación inteligente de la privacidad, justamente haciendo uso de la teoría de la optimización, lleva a una defensa más robusta frente a los ataques, manteniendo siempre el rendimiento del modelo entrenado.

Una vez se hace síntesis de todo el contexto dado, se puede arrojar luz sobre los resultados obtenidos, remarcando y listando las principales conclusiones que se pueden extraer del trabajo:

- El enfoque se integra fácilmente en esquemas de **FL** sin modificar la arquitectura del modelo, y puede extenderse a distintos conjuntos de datos y tipos de ataque, como se demuestra en la experimentación.

10. Conclusiones

- La capacidad de relajar las condiciones de privacidad, así como la de restringirlas, proporciona una resiliencia notable ante ataques tanto al modelo como a la privacidad, ajustándose en función de las necesidades para esa ronda de entrenamiento en específico.
- El ruido añadido durante la aplicación de la **PD** puede aplicarse sin que la repercusión en el rendimiento sea significativa.
- Una gestión dinámica del presupuesto de privacidad permite **reducir el consumo acumulado de ϵ (presupuesto de privacidad)** en más de un 30 % en diversos escenarios, sin deterioro en la *accuracy* del modelo global.
- Frente a ataques de reconstrucción como *DMUG*, la estrategia adaptativa ha aumentado la robustez frente a la extracción de datos privados, elevando la distancia entre las imágenes generadas y las reales. De hecho, se consigue reducir en un 37.1 % el uso de presupuesto de privacidad, brindando un entorno mucho más seguro de aprendizaje.
- En ataques al modelo, la protección adaptativa permite preservar el rendimiento del modelo incluso bajo el ataque de varios adversarios a la vez, ajustando el ruido según la sensibilidad de cada ronda.
- Desde un punto de vista práctico, la formulación mediante **PL** ha permitido incorporar importantes restricciones sobre el rendimiento del modelo, el presupuesto de privacidad total o el ruido añadido en cada ronda, entre otras. Todo esto bajo un marco de eficiencia computacional, de gran relevancia en el ámbito de **DL**.

En conjunto, los resultados avalan el uso de técnicas de optimización dinámica para el control y aplicación de la privacidad en entornos distribuidos, aportando mayor flexibilidad que enfoques puramente estáticos. La explicabilidad es también un tema de gran importancia en el marco de la privacidad y la **IA** segura; aplicar técnicas transparentes y explicables aporta una mayor confianza que modelos de *caja negra*. En este caso, a pesar de haber sido desarrollado hace varias décadas, el método Símplex continúa siendo una herramienta fundamental en el campo de la optimización lineal. Su relevancia no solo es patente en entornos académicos y de investigación teórica, sino que también se extiende a aplicaciones modernas y de actualidad en **ML**, como es este caso. Esto pone de manifiesto su vigencia y versatilidad en tareas actuales de gran complejidad.

Todas estas afirmaciones nos llevan a concluir que se han cumplido con éxito los objetivos propuestos en este Trabajo de Fin de Grado. Se han explorado por completo las posibilidades abiertas al principio de su desarrollo y se han reunido con éxito los requisitos necesarios para poder confirmar el cumplimiento de los objetivos.

11. Trabajos futuros

Pese a los buenos resultados obtenidos, el cumplimiento de los objetivos propuestos y el desarrollo de unas bases sólidas en esta línea de investigación, el enfoque desarrollado para la aplicación inteligente o dinámica de la PD admite mejoras que podrían ser implementadas o estudiadas en un futuro. Tanto desde el ámbito teórico como desde el práctico, el análisis de técnicas que proporcionen privacidad en ámbitos de manejo de datos sigue estando a la orden del día.

Durante el desarrollo del trabajo se han ido haciendo apuntes acerca del alcance del trabajo, abriendo continuamente líneas de investigación interesantes que se salían del mismo. Se listan a continuación cuáles serían y qué aportaría su estudio:

- **Escalabilidad:** Extender los experimentos a escenarios con mayor número de clientes y ataques simultáneos, explorando mecanismos de reparto justo del presupuesto total de privacidad.
- **Compatibilidad con otras defensas y evaluar otros ataques:** Evaluar cómo se comporta la optimización adaptativa combinada con defensas adicionales (clipping dinámico, codificación de gradientes, etc.). Sería también de gran interés explorar nuevos ataques y evaluar la resiliencia del método de optimización para los mismos.
- **Datasets complejos y configuraciones de clientes:** Aplicar el enfoque a conjuntos de datos como CIFAR-100 o CelebA, donde el aprendizaje es más complicado, sobre todo en un escenario colaborativo. Además se podrían probar configuraciones donde los clientes tengan los mismos datos, donde exista un desbalanceo del número de muestras por cliente,... etc.
- **Implementación en marcos reales:** Haciendo referencia a la portabilidad que se comentaba en el primer ítem de las conclusiones, sería interesante importar el sistema directamente a la librería **Flexible-FL** [HJLAG⁺25], pudiendo así evaluar su impacto en muchos otros escenarios.
- **Optimización convexa:** Estudiar si el uso de métodos como el gradiente proyectado puede permitirnos definir problemas algo más complejos, al no tratarse únicamente de restricciones lineales. La convexidad es una herramienta poderosa y valdría la pena contemplar si para casos más complejos es preferible antes que la PL.

Estos desarrollos permitirían consolidar una defensa adaptable y eficiente para el despliegue seguro de modelos colaborativos en entornos federados reales, con garantías teóricas y eficacia empírica demostrada.

Acrónimos

- AI Act** Ley o Acto de Inteligencia Artificial de la UE. [xvii](#)
- ASR** *Attack Success Rate.* [104, 112–114](#)
- CDP** *Central Differential Privacy.* [81, 99, 100, 105, 110, 112, 116](#)
- DL** *Deep Learning.* [xvii, 61, 62, 64, 79, 119, 120](#)
- FL** *Federated Learning.* [xvii, xix–xxii, 71–74, 78, 80, 81, 83, 84, 86, 89, 95, 99–103, 105, 119](#)
- GAN** *Generative Adversarial Networks.* [89, 90, 97, 98, 102–104, 115, 116](#)
- HFL** *Horizontal Federated Learnig.* [77, 84, 85, 89, 90](#)
- IA** Inteligencia Artificial. [xvii, xviii, 61, 80, 120](#)
- ML** *Machine Learning.* [xvii, xxi, xxii, 45, 61, 62, 71, 74, 83, 89, 106, 119, 120](#)
- PC** Programación Convexa. [3, 45, 46, 52, 57, 94](#)
- PD** Privacidad Diferencial. [xvii, xix–xxii, 77–82, 93–95, 99, 100, 102–105, 107–116, 119–121](#)
- PL** Programación Lineal. [xx–xxii, 3, 4, 14, 18, 20–24, 28–32, 44, 45, 52, 57, 94, 95, 99, 104, 105, 119–121](#)
- PPC** Problema de Programación Convexa. [21, 45, 119](#)
- PPL** Problema de Programación Lineal. [21, 24, 28, 42, 43, 93, 119](#)
- PPLC** Problema de Programación Lineal en forma Canónica. [24, 25, 27, 29, 30, 32, 41](#)
- PPLE** Problema de Programación Lineal Estándar. [22](#)
- PPLN** Problema de Programación Lineal Normal. [22–24](#)
- TFL** *Transfer Federated Learnig.* [77](#)
- VFL** *Vertical Federated Learnig.* [77, 84, 85, 90](#)

Bibliografía

- [ABHKS17] Karim Abouelmehdi, Abderrahim Beni-Hssane, Hayat Khaloufi, y Mostafa Saadi. Big data security and privacy in healthcare: A review. *Procedia Computer Science*, 113:73–80, 2017.
- [ASOM12] Edinson Montoro Alegre, Martha Hilda Timoteo Sánchez, Carole Huamán Oriundo, y Gladys Melgarejo. Programación lineal aplicada a un tipo de programación convexa. *Pesquimat*, 15(2):4, 2012.
- [BCM⁺13] Battista Biggio, Igino Corona, Davide Maiorca, Blaine Nelson, Nedim Šrndić, Pavel Laskov, Giorgio Giacinto, y Fabio Roli. Evasion attacks against machine learning at test time. En *Machine learning and knowledge discovery in databases: European conference, ECML pkDD 2013, prague, czech Republic, September 23-27, 2013, proceedings, part III 13*, páginas 387–402. Springer, 2013.
- [BCMC18] Arjun Nitin Bhagoji, Supriyo Chakraborty, Prateek Mittal, y Seraphin Calo. Model poisoning attacks in federated learning. En *Proc. workshop secur. mach. learn.(secML) 32nd conf. neural inf. process. syst.(neurIPS)*, páginas 1–23, 2018.
- [BCMC19] Arjun Nitin Bhagoji, Supriyo Chakraborty, Prateek Mittal, y Seraphin Calo. Analyzing federated learning through an adversarial lens. En *International conference on machine learning*, páginas 634–643. PMLR, 2019.
- [Bero3] Leonard D Berkovitz. *Convexity and optimization in Rn*. John Wiley & Sons, 2003.
- [Biso6] Christopher Bishop. *Pattern Recognition and Machine Learning*, volumen 16, páginas 140–155. 01 2006.
- [Bla77] Robert G Bland. New finite pivoting rules for the simplex method. *Mathematics of operations Research*, 2(2):103–107, 1977.
- [Bla25] Víctor Blanco. Capítulo 4: Caracterización de mínimos en optimización. Material docente (Capítulo de apunte) Capítulo 4, Universidad de Granada, 2025. Disponible en línea: <https://www.ugr.es/~vblanco/Docs/capitulo4.pdf>.
- [BMM17] Mateusz Buda, Atsuto Maki, y Maciej A Mazurowski. A systematic study of the class imbalance problem in convolutional neural networks. *arXiv preprint arXiv:1710.05381*, 2017.
- [BPS19] Eugene Bagdasaryan, Omid Poursaeed, y Vitaly Shmatikov. Differential privacy has disparate impact on model accuracy. En H. Wallach, H. Larochelle, A. Beygelzimer, F. d’Alché-Buc, E. Fox, y R. Garnett, editores, *Advances in Neural Information Processing Systems*, volumen 32. Curran Associates, Inc., 2019.
- [BRG⁺21] Daniel Bernau, Jonas Robl, Philip W Grassal, Steffen Schneider, y Florian Kerschbaum. Comparing local and central differential privacy using membership inference attacks. En *IFIP Annual Conference on Data and Applications Security and Privacy*, páginas 22–42. Springer, 2021.
- [BSJ⁺20] Nuria Rodríguez Barroso, Goran Stipcich, Daniel Jiménez-López, José Antonio Ruiz-Millán, Eugenio Martínez-Cámarra, Gerardo González-Seco, María Victoria Luzón, Miguel Ángel Veganzones, y Francisco Herrera. Federated learning and differential privacy: Software tools analysis, the sherpa.ai FL framework and methodological guidelines for preserving data privacy. *CoRR*, abs/2007.00914, 2020.

Bibliografía

- [BV04] Stephen P Boyd y Lieven Vandenberghe. *Convex optimization*. Cambridge university press, 2004.
- [BVH⁺20] Eugene Bagdasaryan, Andreas Veit, Yiqing Hua, Deborah Estrin, y Vitaly Shmatikov. How to backdoor federated learning. En *International conference on artificial intelligence and statistics*, páginas 2938–2948. PMLR, 2020.
- [BVT⁺24] Dr Preeti Bala, Prof Vaishnav, Prof Shikha Tiwari, Prof Ashish Bhatnagar, et al. Multifaceted sentiment analysis in social media with ml. *Prof Shikha and Bhatnagar, Prof Ashish, Multifaceted Sentiment Analysis in Social Media With ML (May 3, 2024)*, 2024.
- [CCI⁺22] Marcos F Criado, Fernando E Casado, Roberto Iglesias, Carlos V Regueiro, y Senén Barro. Non-iid data and continual learning processes in federated learning: A long road ahead. *Information Fusion*, 88:263–280, 2022.
- [CLL⁺23] Ahmad Chaddad, Qizong Lu, Jiali Li, Yousef Katib, Reem Kateb, Camel Tanougast, Ahmed Bouridane, y Ahmed Abdulkadir. Explainable, domain-adaptive, and federated artificial intelligence in medicine. *IEEE/CAA Journal of Automatica Sinica*, 10(4):859–876, 2023.
- [CNV⁺23] Kristopher K Coelho, Michele Nogueira, Alex B Vieira, Edelberto F Silva, y José Augusto M Nacif. A survey on federated learning for security and privacy in healthcare applications. *Computer Communications*, 207:113–127, 2023.
- [Coo17] John D. Cook. Adding laplace or gaussian noise to a database. <https://www.johndcook.com/blog/2017/09/20/adding-laplace-or-gaussian-noise-to-database/>, September 2017. Accessed: 2025-06-01.
- [DAHGHS18] Kristopher De Asis, J Hernandez-Garcia, G Holland, y Richard Sutton. Multi-step reinforcement learning: A unifying algorithm. En *Proceedings of the AAAI conference on artificial intelligence*, volumen 32, 2018.
- [Dan51] George B Dantzig. Maximization of a linear function of variables subject to linear inequalities. *Activity analysis of production and allocation*, 13:339–347, 1951.
- [Dan90] George B Dantzig. Origins of the simplex method. En *A history of scientific computing*, páginas 141–151. 1990.
- [Den12] Li Deng. The mnist database of handwritten digit images for machine learning research. *IEEE Signal Processing Magazine*, 29(6):141–142, 2012.
- [DFVH⁺20] Vladimir Dvorkin, Ferdinando Fioretto, Pascal Van Hentenryck, Jalal Kazempour, y Pierre Pinson. Differentially private convex optimization with feasibility guarantees. *arXiv preprint arXiv:2006.12338*, 2020.
- [DMNS06] Cynthia Dwork, Frank McSherry, Kobbi Nissim, y Adam Smith. Calibrating noise to sensitivity in private data analysis. En Shai Halevi y Tal Rabin, editores, *Theory of Cryptography*, páginas 265–284, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.
- [DR⁺14] Cynthia Dwork, Aaron Roth, et al. The algorithmic foundations of differential privacy. *Foundations and Trends® in Theoretical Computer Science*, 9(3–4):211–407, 2014.
- [Eur24] European Commission. Regulatory framework proposal on artificial intelligence, 2024. Accessed: 2025-05-03.
- [Fac25] Facultad de Relaciones Laborales y Recursos Humanos, Universidad de Granada. Información general: acceso, secretaría, plataforma prado y otros servicios. <https://laborales.ugr.es/estudiantes/informacion>, 2025. [Accedido: 29-jun-2025].
- [FCJG20] Minghong Fang, Xiaoyu Cao, Jinyuan Jia, y Neil Gong. Local model poisoning attacks to {Byzantine-Robust} federated learning. En *29th USENIX security symposium (USENIX Security 20)*, páginas 1605–1622, 2020.

- [FRL21] Stanislav Fort, Jie Ren, y Balaji Lakshminarayanan. Exploring the limits of out-of-distribution detection. *Advances in neural information processing systems*, 34:7068–7081, 2021.
- [FTFC21] Carlos Ernesto Flores-Tapia y Karla Lissette Flores-Cevallos. Método simplex de programación lineal aplicado a una empresa distribuidora de mobiliario. *Entorno*, (71):22–33, 2021.
- [GACW21] Magali Goirand, Elizabeth Austin, y Robyn Clay-Williams. Implementing ethics in health-care ai-based applications: a scoping review. *Science and Engineering Ethics*, 27(5):61, 2021.
- [Gla25] Glassdoor. Sueldo de investigador. https://www.glassdoor.es/Sueldos/investigador-r-sueldo-SRCH_K00,12.htm, 2025. [Accedido: 30-jun-2025].
- [God17] Michelle Goddard. The eu general data protection regulation (gdpr): European regulation that has a global impact. *International Journal of Market Research*, 59(6):703–705, 2017.
- [Goo19] Google Cloud. Nvidia tesla t4 gpus now available in beta. <https://cloud.google.com/blog/products/ai-machine-learning/nvidia-tesla-t4-gpus-now-available-in-beta>, February 2019. Accedido el 6 de julio de 2025.
- [GR20] Alejandro Garcés Ruiz. Optimización convexa: aplicaciones en operación y dinámica de sistemas de potencia. 2020.
- [GT12] Geoff Gordon y Ryan Tibshirani. Karush-kuhn-tucker conditions. *Optimization*, 10(725/36):725, 2012.
- [HAPC17] Briland Hitaj, Giuseppe Ateniese, y Fernando Perez-Cruz. Deep models under the gan: information leakage from collaborative deep learning. En *Proceedings of the 2017 ACM SIGSAC conference on computer and communications security*, páginas 603–618, 2017.
- [HJLAG⁺25] Francisco Herrera, Daniel Jiménez-López, Alberto Argente-Garrido, Nuria Rodríguez-Barroso, Cristina Zuheros, Ignacio Aguilera-Martos, Beatriz Bello, Mario García-Márquez, y María Victoria Luzón. Flex: Flexible federated learning framework. *Information Fusion*, 117:102792, 2025.
- [HLWZ] S Hu, J Lu, W Wan, y LY Zhang. Challenges and approaches for mitigating byzantine attacks in federated learning.(2021). *CoRR abs/2112.14468*.
- [HTFFo4] Trevor Hastie, Robert Tibshirani, Jerome Friedman, y James Franklin. The elements of statistical learning: Data mining, inference, and prediction. *Math. Intell.*, 27:83–85, 11 2004.
- [HYG19] Xu Han, Haoran Yu, y Haisong Gu. Visual inspection with federated learning. En *Image Analysis and Recognition: 16th International Conference, ICIAR 2019, Waterloo, ON, Canada, August 27–29, 2019, Proceedings, Part II* 16, páginas 52–64. Springer, 2019.
- [KMY⁺16] Jakub Konečný, H. Brendan McMahan, Felix X. Yu, Peter Richtárik, Ananda Theertha Suresh, y Dave Bacon. Federated learning: Strategies for improving communication efficiency. *CoRR abs/1610.05492*, 2016.
- [Kri09] Alex Krizhevsky. Learning multiple layers of features from tiny images. <https://www.cs.toronto.edu/~kriz/cifar.html>, 2009. Accessed: 2025-06-22.
- [Lab25] Systems Optimization Laboratory. Professor george dantzig, stanford operations research department. <https://convexoptimization.com/sol/dantzig.html>, 2025.
- [LBBH98] Yann LeCun, Léon Bottou, Yoshua Bengio, y Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.
- [LBH15] Yann LeCun, Yoshua Bengio, y Geoffrey Hinton. Deep learning. *nature*, 521(7553):436–444, 2015.
- [LMY⁺12] Linyuan Lü, Matúš Medo, Chi Ho Yeung, Yi-Cheng Zhang, Zi-Ke Zhang, y Tao Zhou. Recommender systems. *Physics reports*, 519(1):1–49, 2012.

Bibliografía

- [LRBAG⁺24] M. Victoria Luzón, Nuria Rodríguez-Barroso, Alberto Argente-Garrido, Daniel Jiménez-López, Jose M. Moyano, Javier Del Ser, Weiping Ding, y Francisco Herrera. A tutorial on federated learning from theory to practice: Foundations, software frameworks, exemplary use cases, and selected trends. *IEEE/CAA Journal of Automatica Sinica*, 11(4):824–850, 2024.
- [LSP19] Leslie Lamport, Robert Shostak, y Marshall Pease. The byzantine generals problem. En *Concurrency: the works of leslie lamport*, páginas 203–226. 2019.
- [LSTS20] Tian Li, Anit Kumar Sahu, Ameet Talwalkar, y Virginia Smith. Federated learning: Challenges, methods, and future directions. *IEEE signal processing magazine*, 37(3):50–60, 2020.
- [LWXO21] Xinjian Luo, Yuncheng Wu, Xiaokui Xiao, y Beng Chin Ooi. Feature inference attack on model predictions in vertical federated learning. En *2021 IEEE 37th international conference on data engineering (ICDE)*, páginas 181–192. IEEE, 2021.
- [M⁺10] M. R. Moosavi et al. An adaptive nearest neighbor classifier for noisy environments. En *Electrical Engineering (ICEE), 2010 18th Iranian Conference on*, páginas 1–6. IEEE, 2010.
- [M⁺19] M. R. Moosavi et al. Dealing with noise problem in machine learning data-sets. *Procedia Computer Science*, 161:536–543, 2019.
- [McS09] Frank D McSherry. Privacy integrated queries: an extensible platform for privacy-preserving data analysis. En *Proceedings of the 2009 ACM SIGMOD International Conference on Management of data*, páginas 19–30, 2009.
- [Mit97] Tom M Mitchell. *Machine learning*, volumen 1. McGraw-hill New York, 1997.
- [MMR⁺17a] Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, y Blaise Aguera y Arcas. Communication-Efficient Learning of Deep Networks from Decentralized Data. En Aarti Singh y Jerry Zhu, editores, *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics*, volumen 54 de *Proceedings of Machine Learning Research*, páginas 1273–1282. PMLR, 20–22 Apr 2017.
- [MMR⁺17b] Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, y Blaise Aguera y Arcas. Communication-efficient learning of deep networks from decentralized data. En *Artificial intelligence and statistics*, páginas 1273–1282. PMLR, 2017.
- [MT07] Frank McSherry y Kunal Talwar. Mechanism design via differential privacy. En *48th Annual IEEE Symposium on Foundations of Computer Science (FOCS'07)*, páginas 94–103. IEEE, 2007.
- [MTDC19] Nishat I Mowla, Nguyen H Tran, Inshil Doh, y Kijoon Chae. Federated learning-based cognitive detection of jamming attack in flying ad-hoc network. *IEEe Access*, 8:4338–4350, 2019.
- [Nad18] Ashok Kumar Reddy Nadikattu. Iot and the issue of data privacy. *International Journal of Innovations in Engineering Research and Technology*, 5(10):23–26, 2018.
- [NDZY21] Hao Ni, Xin Dong, Jinsong Zheng, y Guangxi Yu. *An introduction to machine learning in quantitative finance*. World Scientific, 2021.
- [NPC22] Sadaf Naz, Khoa T Phan, y Yi-Ping Phoebe Chen. A comprehensive review of federated learning for covid-19 detection. *International Journal of Intelligent Systems*, 37(3):2371–2392, 2022.
- [NS08] Arvind Narayanan y Vitaly Shmatikov. Robust de-anonymization of large sparse datasets. En *2008 IEEE Symposium on Security and Privacy (sp 2008)*, páginas 111–125, 2008.
- [NSH19] Milad Nasr, Reza Shokri, y Amir Houmansadr. Comprehensive privacy analysis of deep learning: Passive and active white-box inference attacks against centralized and federated learning. En *2019 IEEE symposium on security and privacy (SP)*, páginas 739–753. IEEE, 2019.
- [NW99] Jorge Nocedal y Stephen J Wright. *Numerical optimization*. Springer, 1999.

- [POG⁺24] Ke Pan, Yew-Soon Ong, Maoguo Gong, Hui Li, A Kai Qin, y Yuan Gao. Differential privacy in deep learning: A literature survey. *Neurocomputing*, página 127663, 2024.
- [RBJLL⁺23] Nuria Rodríguez-Barroso, Daniel Jiménez-López, M Victoria Luzón, Francisco Herrera, y Eugenio Martínez-Cámara. Survey on federated learning threats: Concepts, taxonomy on attacks and defences, experimental study and challenges. *Information Fusion*, 90:148–173, 2023.
- [Roc93] R Tyrrell Rockafellar. Lagrange multipliers and optimality. *SIAM review*, 35(2):183–238, 1993.
- [RYD⁺22] Miguel A Ramirez, Sangyoung Yoon, Ernesto Damiani, Hussam Al Hamadi, Claudio Agostino Ardagna, Nicola Bena, Young-Ji Byon, Tae-Yeon Kim, Chung-Suk Cho, y Chan Yeob Yeun. New data poison attacks on machine learning classifiers for mobile exfiltration. *arXiv preprint arXiv:2210.11592*, 2022.
- [SLW25] Aras Selvi, Huikang Liu, y Wolfram Wiesemann. Differential privacy via distributionally robust optimization. *Operations Research*, 2025.
- [SS18] Pramila P Shinde y Seema Shah. A review of machine learning and deep learning applications. En *2018 Fourth international conference on computing communication control and automation (ICCUBEA)*, páginas 1–6. IEEE, 2018.
- [Su1] Miguel Martín Suárez. *Análisis Funcional*. Universidad de Granada, Granada, España, versión 2.0 edición, December 2011.
- [Tik96] Vladimir M Tikhomirov. The evolution of methods of convex optimization. *The American Mathematical Monthly*, 103(1):65–71, 1996.
- [TLG⁺19] Stacey Truex, Ling Liu, Mehmet Emre Gursoy, Lei Yu, y Wenqi Wei. Demystifying membership inference attacks in machine learning as a service. *IEEE transactions on services computing*, 14(6):2073–2089, 2019.
- [TTGL20] Vale Tolpegin, Stacey Truex, Mehmet Emre Gursoy, y Ling Liu. Data poisoning attacks against federated learning systems. En *Computer security—ESORICS 2020: 25th European symposium on research in computer security, ESORICS 2020, guildford, UK, September 14–18, 2020, proceedings, part i* 25, páginas 480–501. Springer, 2020.
- [TTS15] Duygu Sinanc Terzi, Ramazan Terzi, y Seref Sagiroglu. A survey on security and privacy issues in big data. En *2015 10th International Conference for Internet Technology and Secured Transactions (ICITST)*, páginas 202–207. IEEE, 2015.
- [Unk10] Author Unknown. Overfitting. En *Encyclopedia of Machine Learning*, páginas 744–745. Springer, 2010.
- [WYJ⁺22] Donghua Wang, Wen Yao, Tingsong Jiang, Guijian Tang, y Xiaoqian Chen. A survey on physical adversarial attack in computer vision. *arXiv preprint arXiv:2209.14262*, 2022.
- [XJK01] Eric P. Xing, Michael I. Jordan, y Richard M. Karp. Feature selection for high-dimensional genomic microarray data. En *Proceedings of the Eighteenth International Conference on Machine Learning*, páginas 601–608. Morgan Kaufmann Publishers Inc., 2001.
- [XRV17] Han Xiao, Kashif Rasul, y Roland Vollgraf. Fashion-mnist: A novel image dataset for benchmarking machine learning algorithms. <https://github.com/zalandoresearch/fashion-mnist>, 2017. Accessed: 2025-06-22.
- [YLCT19] Qiang Yang, Yang Liu, Tianjian Chen, y Yongxin Tong. Federated machine learning: Concept and applications. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 10(2):1–19, 2019.
- [YSS⁺21] Ashkan Yousefpour, Igor Shilov, Alexandre Sablayrolles, Davide Testuggine, Karthik Prasad, Mani Malek, John Nguyen, Sayan Ghosh, Akash Bharadwaj, Jessica Zhao, Graham Cormode, y Ilya Mironov. Opacus: User-friendly differential privacy library in PyTorch. *arXiv preprint arXiv:2109.12298*, 2021.

Bibliografía

- [YYZH16] Ganzhao Yuan, Yin Yang, Zhenjie Zhang, y Zhifeng Hao. Convex optimization for linear query processing under approximate differential privacy. En *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2016.
- [ZLH19] Ligeng Zhu, Zhijian Liu, y Song Han. Deep leakage from gradients. *Advances in neural information processing systems*, 32, 2019.