UNIVERSIDAD
DE GRANADA

MITIGANDO LAS VULNERABILIDADES DEL APRENDIZAJE FEDERADO MEDIANTE BLOCKCHAIN

MARIO GARCÍA MÁRQUEZ

Trabajo Fin de Grado

Doble Grado en Ingeniería Informática y Matemáticas

Tutores
Nuria Rodríguez Barroso
Francisco Herrera Triguero

FACULTAD DE CIENCIAS
E.T.S. INGENIERÍAS INFORMÁTICA Y DE TELECOMUNICACIÓN

Granada, 14 de junio de 2024

ÍNDICE GENERAL

Yo, **Mario García Márquez**, alumno del **Doble Grado de Ingeniería Informática y Matemáticas** de la **Escuela Técnica Superior de Ingeniería Informática y de Telecomunicaciones** y la **Facultad de Ciencias** de la **Universidad de Granada**, con DNI 77147974-V, autorizo la ubicación de la siguiente copia de mi Trabajo Fin de Grado en la Biblioteca del Centro para que pueda ser consultada por las personas que lo deseen.

Fdo: Mario García Márquez

Granada a 11 de junio del 2024

D. **Francisco Herrera Triguero**, Catedrático del Área de Informática del departamento de **Ciencias de la Computación e Inteligencia Artificial** de la Universidad de Granada, Dª **Nuria Rodríguez Barroso**, Docente Invitada y Contrado Investigador con Cargo a Proyecto del departamento de **Ciencias de la Computación e Inteligencia Artificial** de la Universidad de Granada.

Informan:

Que el presente trabajo, titulado **Mitigando las Vulnerabilidades del Aprendizaje Federado Mediante Blockchain**, ha sido realizado bajo su supervisión por **Mario García Márquez**, y autorizan la defensa de dicho trabajo ante la Comisión de Evaluación que corresponda.

Y para que conste, expide y firma el presente documento en Granada a 11 de Junio de 2024.

Los tutores:

Francisco Herrera Triguero **Nuria Rodríguez Barroso**

Yo, **Mario García Márquez**, alumno del **Doble Grado en Ingeniería Informática y Matemáticas** de la **Escuela Técnica Superior de Ingeniería Informática y Matemática** y de la **Facultad de Ciencias** de la **Universidad de Granada**, con DNI 77147974-V, declaro explícitamente que el trabajo presentado es original, entendido en el sentido de que no he utilizado ninguna fuente sin citarla debidamente.

Fdo: Mario García Márquez

Granada a 11 de junio del 2024

AGRADECIMIENTOS

Este trabajo se puede ver como la conclusión de muchos años de educación y aprendizaje. La realización de este no podría haberse llevado a cabo sin la ayuda de mucha gente. Si buscamos al culpable más directo de estar leyendo esto no puedo decir otra persona que no sea mi tutora, Nuria. Durante estos 5 años de carrera me ha acompañado durante casi la mitad de ellos y además de una más que sobresaliente labor docente (porque hay profesores que marcan y eso es algo que se aprende con la experiencia) también es una de las mejores personas que he conocido jamás y una excelente amiga. Muchas gracias por ser el gigante que me ha puesto sobre sus hombros y me ha enseñado el camino a seguir.

Pero quedarnos en el culpable directo sería poco, y es que hay gente que siempre ha estado ahí, como mi familia que siempre me ha servido un apoyo incondicional a cualquier decisión que he tomado y siempre me ha animado con todo lo académico que me ha acabado llenando tanto. También tengo que agradecer a mis compañeros de carrera: Nerea, Javi, Pablo, Maxim, Isa, Inma, Julio, Jaime y muchos más que no cabrían en tan poco espacio. Gracias por hacerme pensar, mejorar y reírme tanto. Aunque las horas de clases se me hayan hecho eternas, habéis logrado que los años se me hayan hecho cortos.

También, pero no menos importante, tengo que agradecer a todos mis compañeros de Alemania que me han dado su calor incluso por los kilómetros que nos separan. Gracias Alba, Fueyo, Dani, Víctor y Ali. Sin vosotros difícilmente sería quien soy ahora y tampoco lo sería así este trabajo.

Muchas gracias a todos.

RESUMEN

En este proyecto se estudia el estado actual de la aplicabilidad de las tecnologías de tipo *blockchain* al Aprendizaje Federado (FL) para la mitigación de ataques adversarios. Para ello se implementan arquitecturas estándar de FL y de *blockchain* aplicado a FL, comprobándose su capacidad como defensa ante ataques y se propone una nueva arquitectura que mejora las anteriores en este sentido.

Para introducir el proyecto y conocer mejor sus fundamentos, se realiza una introducción al álgebra lineal, teoría de la probabilidad, inferencia y teoría de la información. De forma seguida, se explica la base de la optimización de funciones no lineales, el operador de agregación Krum y nociones sobre Aprendizaje Automático, más concretamente Aprendizaje Profundo.

Los siguientes capítulos se enfocan en explicar FL, así como los distintos ataques a los que es vulnerable este esquema, haciendo un especial énfasis en los tratados en el trabajo. También se explica la tecnología *blockchain* y sus beneficios en sistemas distribuidos.

Finalmente, se explica el estado actual del paradigma de *blockchain* aplicado a FL, se expone la hipótesis de cómo ciertas arquitecturas podrían servir como una capa de defensa para FL y se propone una nueva arquitectura enfocada en la seguridad del modelo. Esta nueva arquitectura, Krum Federated Chain (KFC), se basa en las propuestas actuales de aplicabilidad de *blockchain* a FL con un efoque especial en defensa logrado por la aplicación del operador de agregación Krum. Esto es seguido por la realización y análisis de una serie de experimentos para contrastar la hipótesis planteada y la propuesta, observando como KFC mejora los métodos de defensa ofrecidos por *blockchain* mantiendo el modelo seguro bajo escenarios más restrictivos, junto a conclusiones y vías de trabajo futuro.

Palabras clave: aprendizaje federado, ataques adversarios, redes neuronales, blockchain, clasificación de imágenes, aprendizaje automático, optimización no lineal.

ABSTRACT

This study aims to examine the integration of blockchain technologies with Federated Learning (FL), emphasizing the potential of this combination to protect federated frameworks from adversarial attacks. Initially, the context of the problem is introduced.

TRUSTWORTHY AI

Artificial Intelligence (AI) systems have become an integral part of daily life, often operating unobtrusively. From recommendation systems to Instagram filters, AI significantly influences various aspects of human existence. Recently, the rising prominence of systems such as ChatGPT and autonomous vehicles has sparked widespread concern regarding their security and potential negative impacts on sectors such as healthcare, education, culture, and democracy. In response to these concerns, the concept of Trustworthy AI systems has emerged, founded on several technical requirements.

FEDERATED LEARNING

A critical technical requirement for Trustworthy AI is security and data governance, which has led to the development of Federated Learning (FL). FL is an innovative architecture that enables collaborative model training across decentralized devices while maintaining data privacy. Despite its advantages, the decentralized nature of FL can be a vulnerability. Since data remains in the nodes, traditional data exploration techniques cannot be employed, rendering the framework susceptible to data poisoning attacks.

BLOCKCHAIN APPLIED TO FEDERATED LEARNING

Blockchain technologies have gained substantial popularity in recent years, partly due to the success of innovations like Bitcoin. The integration of blockchain into FL has emerged as a promising solution to enhance its security and integrity. By leveraging the immutable and transparent nature of blockchain, FL systems can establish a tamper-resistant record of model updates and participant contributions, thereby mitigating the risks of data manipulation and unauthorized access.

MATHEMATICAL FOUNDATION

To fully comprehend blockchain technology and machine learning, it is essential to delve into the key mathematical concepts underlying these topics.

This foundational chapter begins with an introduction to basic linear algebra, which is a critical component for understanding machine learning models. Subsequently, we explore probability, inference, and information theory. Here, we introduce the fundamental concepts of probability theory, including random variables, probability distributions, and conditional probability. The discussion then progresses to parameter estimation and maximum likelihood estimation, as machine learning models are frequently optimized using these estimators.

Further, the chapter addresses nonlinear optimization, outlining the essential conditions of this theory and explaining the gradient descent method for unconstrained optimization, which is fundamental for optimizing machine learning models. Lastly, we examine the Krum aggregation operator, which provides a robust algorithm for aggregating gradient estimates, ensuring resilience against outliers.

In addition, we address general deep learning algorithms by introducing neural networks from a theoretical perspective. We define the general structure of a neural network, followed by a discussion on the loss function based on cross-entropy. We then define output units and present specific examples such as sigmoidal units. This is followed by an introduction to hidden units, in a manner similar to our treatment of output units.

After establishing a background in general neural networks, we focus on convolutional neural networks (CNNs) and the classification problem. We define the convolution operation, which plays a crucial role in CNN architecture. Finally, we introduce the notion of the EfficientNet architecture, which will be employed during the experiments in this study.

BLOCKCHAIN AND FEDERATED LEARNING

The following chapters provide a formal exposition of FL. We begin by defining an FL system and presenting a taxonomy of the various types of attacks to which FL is vulnerable. We delve deeper into attacks that specifically target the federated model, as well as some current defenses against these attacks that have been explored in the literature.

Next, we explain blockchain systems, detailing their main components and how they can be utilized to preserve trust and privacy throughout the entire system. We then elucidate the consensus mechanisms that enable blockchain systems to make decisions while maintaining a decentralized structure.

Finally, we explore the combination of FL and blockchain, illustrating the current state of this paradigm. We discuss successful applications of this technological integration and examine the different consensus mechanisms and architectures that have been proposed to achieve the proper integration of both technologies.

KRUM FEDERATED CHAIN

Having established the core concepts, the subsequent chapters are dedicated to our proposed framework. We begin by hypothesizing that Proof Of Federated Learning (PoFL), a consensus mechanism specifically designed for FL, can, under certain circumstances, serve as a valid defense against adversarial attacks, even though this was not its original purpose.

Inspired by its disadvantages, we then propose Krum Federated Chain (KFC), a novel blockchain architecture applied to FL. This architecture leverages the PoFL consensus mechanism and the Krum aggregation operator to address more complex scenarios where our initial hypothesis does not hold.

Following this, we detail the experimental setup in which multiple image classification models are trained using KFC, along with some standard architectures from the literature, to validate our proposal. We analyze the results, demonstrating how KFC provides a robust defense against adversarial attacks in federated frameworks, even in cases where PoFL alone does not suffice, thereby establishing KFC as a viable architecture for integrating blockchain and FL.

Finally, we present the conclusions of this study and propose potential directions for future research.

Keywords: federated learning, neuronal networks, adversarial attacks, blockchain, image classification, machine learning, non linear optimization.

LISTA DE ABREVIACIONES

AA	Aprendizaje Automático
BFT	Tolerante a Fallos Bizantinos (*Byzantine Fault Tolerance*)
CNN	Red Neuronal Convolucional (Convolutional Neural Network)
DP	Privacidad Diferencial (*Differential Privacy*)
f.d.p.	Función de Densidad de Probabilidad
FedAvg	*Federated Averaging*
FL	Aprendizaje Federado *(Federated Learning)*
f.m.p.	Función Masa de Probabilidad
FTL	*Federated Transfer Learning*
HFL	Aprendizaje Federado Horizontal (*Horizontal Federated Learning*)
IA	Inteligencia Artificial
i.i.d.	Independientes e Idénticamente Distribuidas
IoT	Internet de las Cosas
KFC	*Krum Federated Chain*
lr	*Learning Rate*
MLP	Perceptrón Multicapa (*Multilayer Perceptron*)
P2P	*Peer-to-Peer*
PoFL	Prueba de Aprendizaje Federado (*Proof of Federated Learning*)
PoS	Prueba de Participación (*Proof of Stake*)
PoW	Prueba de Trabajo (*Proof of Work*)
v.a.	Variable Aleatoria
VFL	Aprendizaje Federado Vertical (*Vertical Federated Learning*)

Parte I

INTRODUCCIÓN

1

INTRODUCCIÓN

1.1 CONTEXTO

La Inteligencia Artificial (IA) ha tenido un impacto significativo en nuestro día a día, aunque a menudo de manera imperceptible. Desde la personalización de recomendaciones de productos en línea hasta encontrar el trayecto óptimo para moverte de un punto a otro en aplicaciones de mapas, la IA lleva más de 20 años estando presente en varios aspectos de nuestra vida cotidiana.

Sin embargo, ahora está más presente que nunca. Con la aparición de productos como chatGPT, asistentes de voz como Alexa o Siri, sistemas de reconocimiento facial como FaceID o la conducción autónoma como Tesla Autopilot, la IA se ha convertido que una parte fundamental de la vida de muchas personas. También ha aumentado enormemente el interés general, tanto de personas particulares como de empresas, por esta tecnología con el fin de aplicarla a todas las áreas posibles maximizando el beneficio producido por su uso.

Aunque aún se está explorando el potencial de la IA en varios campos, hay una creciente preocupación sobre cómo sistemas de IA no supervisados por humanos pueden causar un impacto negativo en áreas más delicadas como la educación, salud, democracia, justicia, cultura o ciencia. Esto ha llevado a la definición de *High-Risk AI Systems* por parte de *EU AI Act* [24], los cuales son sistemas que pueden suponer un riesgo a los derechos fundamentales de las personas o a su integridad, como sistemas que procesen datos biométricos o aquellos usados en vehículos.

En este contexto, emergen los conceptos de sistemas de *Trustworthy AI* o IA confiable y *Responsible AI* [34]. Los sistemas de *Trustworthy AI* están basados en siete requisitos técnicos que se deben de mantener durante durante todo el ciclo de vida del sistema de IA:

1. Supervisión e inspección humana.
2. Robustez y seguridad.
3. Privacidad y gobernanza de los datos.
4. Transparencia.

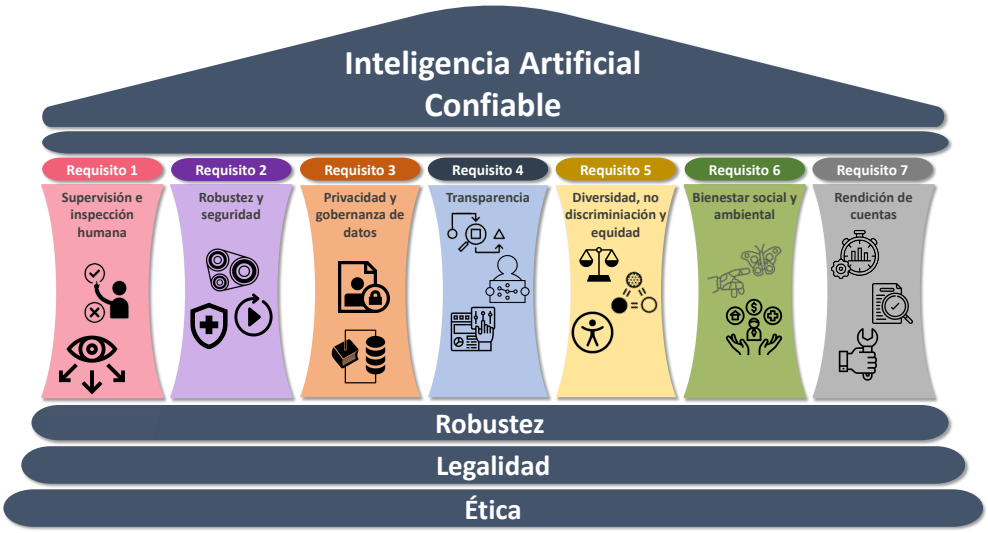


Figura 1: Diagrama de los requisitos técnicos de un sistema de *Trusworthy AI*. Fuente: [6].

5. Diversidad, no discriminación y equidad.
6. Bienestar social y ambiental.
7. Rendición de cuentas.

Estos requisitos están basados en tres pilares fundamentales: (1) legalidad, (2) ética, y (3) robustez, tanto desde perspectivas técnicas como sociales. Estos requisitos y sus pilares quedan ilustrados en la Figura 1.

Uno de los requisitos técnicos de estos sistemas es garantizar la privacidad. Para ello, en 2016 e impulsado por Google, aparece el concepto de Aprendizaje Federado *(Federated Learning)* (FL) [20], un paradigma de aprendizaje que permite el entrenamiento distribuido asegurando la privacidad de los datos. También durante este tiempo se aumenta el interés de las masas en las tecnologías *blockchain*, tecnologías que permiten el registro digital descentralizado de transacciones compartidas entre una red que es inmutable, en parte impulsadas por el éxito de Bitcoin [13].

1.2 MOTIVACIÓN

Por su parte, el FL permite el entrenamiento de un modelo distribuido sin que los clientes tengan que compartir sus datos privados con ninguna entidad. Para ello cada cliente entrena de manera local un modelo con sus datos personales, siendo este

modelo el que luego es compartido con un servidor central, que finalmente agrega todos los resultados de los clientes para obtener un nuevo modelo global. FL también incluye más ventajas como costes reducidos en comunicación o robustez respecto a otras alternativas distribuidas.

Por otro lado, las tecnologías *blockchain* ofrecen un esquema distribuido para computación [5] donde los datos son verificables e inmutables por todas las entidades mediante el uso de funciones criptográficas. A esto hay que añadir que es un sistema descentralizado donde no hay un servidor central, por lo que no hay un único servidor que sea vulnerable a ataques. Otros beneficios de estas tecnologías es una alta escalabilidad y facilidad para rastrear cambios a los datos compartidos [11].

Debido a estas características, el FL se ha empleado en muchas situaciones con éxito [18]. Sin embargo, al igual que cualquier modelo de Aprendizaje Automático (AA), es sensible a ataques. Se conocen varios tipos de ataques adversarios, enfocándose tanto en la funcionalidad del modelo como en la privacidad de los datos. Particularmente esto es una vulnerabilidad del FL, debido a que al no tener acceso a los datos del entrenamiento, no se pueden aplicar técnicas de inspección de datos por lo que estos ataques se vuelven mucho más difíciles de mitigar. Son muchas las propuestas para mitigar estos ataques [29] pero todavía no se ha logrado solucionar al completo esta vulnerabilidad.

Es por ello que muchos trabajos han desarrollado maneras de combinar la tecnología FL y *blockchain* [14, 28, 45] con aplicabilidad a Internet de las Cosas (IoT) industrial, detección de anomalías o resistencia a fallos de dispositivos. Uno de los principales argumentos para esto ha sido el de ofrecer una resistencia ante distintos tipos de ataques.

1.3 PROPUESTA

En este trabajo exponemos la hipótesis de que Prueba de Aprendizaje Federado (*Proof of Federated Learning*) (PoFL), un mecanismo de consenso diseñado para la combinación entre FL y *blockchain* con el fin de mejorar la eficiencia energética del sistema, podría ser un mecanismo viable de defensa contra ciertos ataques a un esquema federado. Sin embargo, esta hipótesis consta de algunos requisitos restrictivos sobre el escenario, no siendo aplicable de forma generalizada como mecanismo de defensa.

Impulsados por estas limitaciones, proponemos la principal aportación de este trabajo, *Krum Federated Chain* (KFC), una nueva arquitectura de FL y *blockchain* que tiene como objetivo el ofrecer una capa de seguridad robusta en los escenarios en los que PoFL no se considera una defensa consistente.

Para verificar nuestras hipótesis, hemos entrenado tres modelos de clasificación de imágenes en distintos conjuntos de datos y en ambos escenarios: (1) cumpliendo las hipótesis necesitadas para PoFL y (2) cuando estas restricciones no se cumplen, ambos

bajo ataques hacia la funcionalidad del modelo, siendo estos un ataque de *backdoor* y un ataque bizantino. También hacemos una comparación con arquitecturas clásicas en la literatura para verificar los resultados.

Los resultados de los experimentos nos indican que, cumpliendo las hipótesis, PoFL resulta ser eficaz y logra mitigar los ataques. Sin embargo, en caso de que no se cumplan, PoFL se muestra completamente vulnerable ante estos ataques. Por el contrario, KFC demuestra ser robusto, mitigando todos los ataques en todos los escenarios sin ninguna pérdida de rendimiento o compromiso con la seguridad del modelo, postulándose así como un método excelente para la defensa en esquemas federados.

1.4 ESTRUCTURA

Esta memoria se organiza de la siguiente manera:

- Primero hablaremos de todos los conceptos previos para hablar nuestra propuesta en la Parte II. Esta parte incluye:
  - Una introducción al álgebra lineal en la Sección 4.
  - Una introducción a la probabilidad y teoría de la información en la Sección 5.
  - Una introducción a la optimización no lineal sin restricciones en la Sección 6.
  - Un estudio del operador de agregación Krum en la Sección 7.
  - Una introducción a la teoría del aprendizaje en la Sección 8.
  - Una introducción al *Deep Learning* en la Sección 9.
- Una vez dados los conceptos previos, podemos introducir conceptos más avanzados en la Parte III.
  - Hablaremos de FL en la Sección 10.
  - Introduciremos el concepto de *blockchain* en la Sección 11.
- Una vez explicados todos los conceptos necesarios, hablaremos de nuestra propuesta en la Parte IV.
  - Empezaremos hablando sobre cómo combinar *Blockchain* y FL en la Sección 12.
  - Expondremos nuestra propuesta en la Sección 13.
  - Plantearemos los experimentos realizados en la Sección 14.
  - Analizaremos los resultados en la Sección 15
- Finalmente, hablaremos de las conclusiones del trabajo en la Sección 16.

2

OBJETIVOS

El objetivo de este proyecto será el de comprobar cómo la tecnología *blockchain* puede servir para mitigar ataques a un esquema federado, concretamente a los ataques al modelo.

Para ello se analizará el estado actual de paradigma *blockchain* aplicado al FL, se estudiará la viabilidad de las distintas propuestas en el campo como mecanismo de defensa y se propondrá una nueva arquitectura que mejore de algún modo las soluciones ya existentes.

Los requerimientos del proyecto que desarrollemos se resumen en:

1. Estudio y escritura sobre el estado actual de FL.
2. Estudio y escritura sobre la tecnología *blockchain*.
3. Estudio y escritura sobre la combinación y aplicabilidad de FL y *blockchain*.
4. Implementar experimentos para comprobar la viabilidad de las propuestas.
5. Estudio de una base matemática del trabajo.
6. Escritura de la base matemática del trabajo.
7. Estudio de los fundamentos de *deep learning*.
8. Redacción de los fundamentos de *deep learning*.
9. Análisis y escritura de los resultados y conclusiones del experimento.

3

PLANIFICACIÓN Y PRESUPUESTO

Para mostrar como ha sido la planificación de tareas a lo largo del tiempo se ha hecho uso de un Diagrama de Gantt, una herramienta visual que permite ver la distribución de las tareas a lo largo de un periodo.

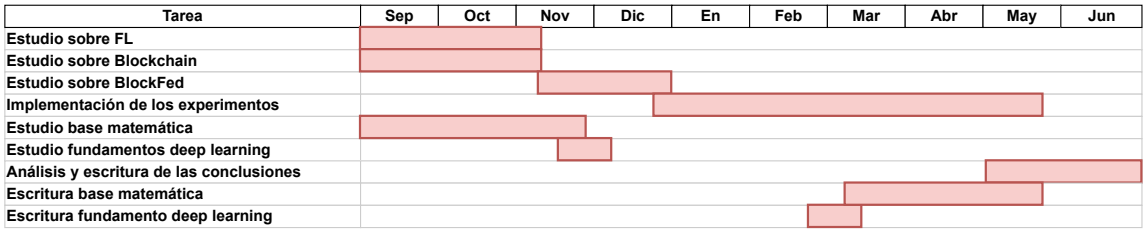


Figura 2: Diagrama de Gantt mostrando la planificación temporal.

Como podemos ver se ha trabajado de manera constante durante todo el curso en este trabajo. Las primeras tareas han sido la debida documentación sobre los fundamentos matemáticos, FL, *blockchain* y la aplicabilidad de estas dos tecnologías juntas. Durante el estudio de esta última, comenzó la implementación de los experimentos y su ejecución, la cuál ha ocupado gran parte en el tiempo debido a las múltiples iteraciones realizadas para verificar nuestras hipótesis. Una vez teniendo los experimentos encaminados, se ha dedicado el resto del tiempo a la escritura de la memoria y al análisis de estos.

En total se han empleado 10 meses para realizar el proyecto.

A la hora de calcular el presupuesto se tiene en cuenta los siguientes factores:

- **Coste personal:** sabiendo que el sueldo medio de un Ingeniero Informático Jr. en España asciende a 1800 euros mensuales y su cotización social media es del 32,6 %, el gasto de personal de manera mensual sería de 2670,60 euros.
  Sabiendo la duración del proyecto y los datos anteriores deducimos que el gasto de personal para este proyecto es de $10 \times 2670,60 =$ **26.706 euros**.
- **Coste del equipo:** para la realización de los experimentos se ha usado el clúster *Talos* del Instituto Andaluz de Ciencia de Datos e Inteligencia Computacional (DaSCI) que destaca por contar con 8 tarjetas gráficas NVIDIA A100 40GB. Tras

una aproximación del tiempo total de ejecución, el precio del uso de este clúster en un servicio como Google Cloud asciende a **4.601,35 euros**.

Por lo tanto, el presupuesto total de este trabajo sería de **31.307,35 euros**.

Parte II

FUNDAMENTOS TEÓRICOS

4

ÁLGEBRA LINEAL

En este capítulo introduciremos conceptos básicos de álgebra lineal [10]. Se necesitará un buen entendimiento en álgebra lineal para ser capaces de trabajar con muchos algoritmos de AA, especialmente con aquellos de *deep learning*.

4.1 CONCEPTOS BÁSICOS

En nuestro estudio del álgebra lineal nos encontraremos con distintos tipos de objetivos matemáticos:

- **Escalares:** un escalar es un número o elemento de un cuerpo, a diferencia de los demás elementos que estudiaremos más adelante que se suelen componer de varios elementos.
- **Vectores:** un vector es un conjunto ordenado de números. Podemos referirnos a cada número del vector mediante su índice en el conjunto.
- **Matrices:** una matriz es un conjunto ordenado de números de dos dimensiones. Así, para referirnos a un número lo haremos mediante dos índices.
- **Tensores:** un algunas ocasiones necesitaremos más de dos ejes. En este caso recurriremos a los tensores, conjuntos de números organizados en una cuadrícula general con un número variable de ejes.

Una operación importante en matrices es la **transposición**. La operación transposición equivale a obtener la imagen de un espejo a través de la diagonal de la matriz. Es decir:

Definición 1. Dada una matriz A, definimos su **transpuesta**, denotada por A^T verificando

$$(A^T)_{i,j} = A_{j,i}. \tag{1}$$

Pese a que hayamos definido la transpuesta como una operación sobre matrices, es fácilmente aplicable a vectores viéndolos como matrices fila y a escalares viéndolos como una matriz de una sola entrada.

Es notorio mencionar que el transpuesto de un escalar es siempre él mismo, esto es $a^T = a$.

En estas estructuras podemos definir las siguientes **operaciones**:

1. Dadas dos matrices A y B de misma dimensión, podemos definir su suma como la matriz como la suma de las matrices componente a componente, esto es:
$$C = A + B \implies C_{i,j} = A_{i,j} + B_{i,j}. \tag{2}$$
2. Definimos la suma y producto de una matriz y un escalar mediante la operación componente a componente, esto es:
$$C = aA + c \implies C_{i,j} = aA_{i,j} + c. \tag{3}$$
3. El producto matricial es una de las operaciones más usadas del álgebra lineal en el AA.

   Definición 2. Sea $A \in \mathbb{R}^{n\times m}$ y $B \in \mathbb{R}^{m\times k}$, entonces el **producto matricial** de A y B es una matriz $C \in \mathbb{R}^{n\times k}$ verificando
$$C_{i,j} = \sum_{k=1}^{m} A_{i,k}B_{k,j}. \tag{4}$$

Propiedades del producto matricial:

1. El producto matricial es distributivo:
$$A(B + C) = AB + AC. \tag{5}$$
2. El producto matricial es asociativo:
$$(AB)C = A(BC). \tag{6}$$
3. El producto matricial no es conmutativo:
$$AB \neq BA. \tag{7}$$
4. La transpuesta de un producto es el producto inverso de sus transpuestas, esto es:
$$(AB)^T = B^T A^T. \tag{8}$$

De cara a la resolución de ecuaciones lineales haremos uso de la inversión de matrices. Para ello, primero debemos de introducir el concepto de matriz identidad.

Definición 3. Definimos la **matriz identidad** como aquella que no modifica ningún vector cuando es multiplicado por ella. La matriz identidad de dimensión n quedará denotada por $I_n \in R^{n\times n}$ y verifica que
$$\forall x \in R^n, I_n x = x. \tag{9}$$

Esta matriz se compone con una diagonal de 1 y todos sus demás elementos 0.

Definición 4. Dada una matriz $A \in \mathbb{R}^{n\times n}$ definimos su matriz inversa $A^{-1} \in \mathbb{R}^{n\times n}$ como aquella matriz que cumple que

$$AA^{-1} = A^{-1}A = I_n. \tag{10}$$

Cabe destacar que dada una matriz $A \in \mathbb{R}^{n\times n}$ su matriz inversa no tiene por qué existir. También, si A^{-1} es la matriz inversa de A, es fácil ver que

$$(A^{-1})^{-1} = A. \tag{11}$$

Definición 5. Se define el **determinante** de una matriz cuadrada $A \in \mathbb{R}^{n\times n}$ y se denotará por $det(A)$ a la suma de los $n!$ productos formados por los n-factores que se obtienen de multiplicar n-elementos de la matriz de tal forma que cada producto contenga un solo elemento de cada fila y de cada columna de A.

También nos interesará definir unos tipos especiales de matrices bastante común en AA:

Definición 6. Diremos que una matriz A es simétrica si $A = A^T$.

Definición 7. Diremos que una matriz A es ortogonal si $A^{-1} = A^T$.

4.2 SISTEMAS LINEALES DE ECUACIONES

Ahora conocemos suficiente álgebra lineal como para poder definir un sistema lineal de ecuaciones.

Definición 8. Dada una matriz conocida $A \in \mathbb{R}^{m\times n}$, un vector conocido $b \in \mathbb{R}^m$ y un vector desconocido $x \in \mathbb{R}^n$ definimos un **sistema lineal de ecuaciones** como la ecuación

$$Ax = b. \tag{12}$$

Cada elemento x_i de nuestro vector desconocido x es una variable que deseamos conocer.

Otra forma de escribir la ecuación 12 de manera más explícita es:

$$\begin{aligned} A_{1,1}x_1 + A_{1,2}x_2 + \ldots + A_{1,n}x_n &= b_1 \\ A_{2,1}x_1 + A_{2,2}x_2 + \ldots + A_{2,n}x_n &= b_2 \\ &\ldots \\ A_{m,1}x_1 + A_{m,2}x_2 + \ldots + A_{m,n}x_n &= b_m. \end{aligned} \tag{13}$$

Gran parte de las operaciones en el *deep learning* se basan en ecuaciones de este tipo con la agregación de operaciones no lineales.

Definición 9. Dado un conjunto de vectores $\{v^{(1)}, \ldots, v^{(n)}\}$ diremos que son **linealmente independientes** si dado un vector del conjunto, no se puede expresar como combinación lineal de los demás vectores. Esto es:

$$\nexists \alpha_j \ / \ v^{(i)} = \sum_{k=1}^{n} \alpha_k v^{(k)}, \ k \neq i. \tag{14}$$

4.3 NORMAS

Otro concepto muy usado en el *deep learning* son las normas de vectores y matrices. Las normas son funciones que llevan vectores a escalares no negativos. De manera intuitiva se podría ver la norma de un vector x como la distancia desde el origen hasta x. De manera más rigurosa:

Definición 10. Una norma es una función $f : \mathbb{R}^n \to \mathbb{R}_0^+$ tal que:

1. $f(x) = 0 \implies x = 0$
2. $\forall y \in \mathbb{R}^n, f(x + y) \leq f(x) + f(y)$ (desigualdad triangular)
3. $\forall \alpha \in \mathbb{R}, f(\alpha x) = |\alpha| f(x)$

Definición 11. Sea $x \in \mathbb{R}^n$, se define la norma L^p con $p \in [1, \infty)$ de x como:

$$||x||_p = (\sum_{i=1}^{n} |x_i|^p)^{\frac{1}{p}}. \tag{15}$$

Algunos casos particulares de la norma L^p son:

1. **Norma euclídea:** es el caso cuando $p = 2$. Es la medida más usada en el AA y se suele denotar como $||x||$ por simplicidad. Con bastante frecuencia mediremos su cuadrado debido a su sencillez de cálculo, pues es fácil ver que $||x||^2 = xx^T$.
2. **Norma L^1:** también bastante usada se trata del caso $p = 1$:

$$||x||_1 = \sum_{i=1}^{n} |x_i|. \tag{16}$$

3. **Norma L^∞:** también conocida como **norma del máximo** está definida mediante:

$$||x||_\infty = \max_i |x_i|. \tag{17}$$

4.4 DESCOMPOSICIÓN EN AUTOVALORES

Muchos objetos matemáticos pueden ser mejor comprendidos separándolos en partes. Por ejemplo, los números enteros pueden ser vistos como una descomposición en números primos. En el caso de las matrices, una de las descomposiciones más usadas es en **autovalores** o **valores propios** y los autovectores o vectores propios.

Definición 12. Un vector $v \in \mathbb{R}^n$ y un escalar $\lambda \in \mathbb{R}$ se dirán que son un **vector propio** y un **valor propio** respectivamente de una matriz cuadrada $A \in \mathbb{R}^{n \times n}$ si se verifica que

$$Av = \lambda v. \tag{18}$$

Proposición 1. *Si v es un vector propio de una matriz cuadrada A, entonces también lo será sv con $s \in \mathbb{R}, s \neq 0$. Es más, v y sv comparten el mismo valor propio:*

$$Av = \lambda v \implies A(sv) = \lambda(sv). \tag{19}$$

Proposición 2. *Dada una matriz cuadrada A, entonces su determinante $det(A)$ se corresponde con el producto de sus valores propios:*

$$det(A) = \prod_i \lambda_i. \tag{20}$$

Veamos la descomposición matricial que se produce mediante este tipo de vectores. Para ello supongamos que nuestra matriz A tiene n vectores propios linealmente independientes $\{v^{(1)}, \ldots, v^{(n)}\}$ asociados a los valores propios $\{\lambda_1, \ldots, \lambda_n\}$. Podemos unir estos vectores en una matriz V con un vector propio por columna:

$$V = [v^{(1)}, \ldots, v^{(n)}]. \tag{21}$$

Del mismo modo, podemos unir los valores propios en un vector denotado por λ. Entonces:

Proposición 3. *La **descomposición en valores propios** de A viene dada por*

$$A = Vdiag(\lambda)V^{-1}. \tag{22}$$

Un problema surge cuando dada una matriz queremos buscar sus vectores y valores propios. Sin embargo, en el caso de que una matriz sea simétrica, el cuál es un caso bastante común en AA, esta puede ser descompuesta usando solo valores propios y vectores propios de números reales.

Proposición 4. *Toda matriz A simétrica puede descomponerse en valores propios reales de la forma:*

$$A = Q\Lambda Q^T \tag{23}$$

donde Q es una matriz ortogonal compuesta de los vectores propios de A y Λ es una matriz diagonal.

Mediante el uso de los valores propios de una matriz A podemos clasificar las matrices en:

- **Definida positiva:** si $\lambda_i > 0 \ \forall i \in 1, \ldots, n$.

- **Semidefinida positiva:** si $\lambda_i \geq 0 \; \forall i \in 1, \ldots, n$.
- **Definida negativa:** si $\lambda_i < 0 \; \forall i \in 1, \ldots, n$.
- **Semidefinida negativa:** si $\lambda_i \leq 0 \; \forall i \in 1, \ldots, n$.

Cabe destacar que dada una matriz A no tiene por qué ser clasificada en uno de los casos anteriores. Sin embargo es interesante pues si una matriz A es semidefinida positiva entonces podemos garantizar que $x^T Ax \geq 0, \forall x$ lo cual es un requisito en ciertos algoritmos de AA.

5

PROBABILIDAD E INFERENCIA ESTADÍSTICA

En este capítulo desarrollaremos la Teoría de la Probabilidad y la Teoría de la Información de una forma muy básica con el objetivo de facilitar la comprensión del resto del trabajo [10].

La teoría de la probabilidad es un marco de trabajo matemático para representar la incertidumbre de los hechos. Nos proporciona tanto métodos para cuantificar la incertidumbre cómo axiomas para derivar nuevos enunciados. En la IA, usamos la teoría de la probabilidad principalmente de dos maneras. Primero, las leyes de la probabilidad nos indican cómo un sistema de IA debería de razonar, de forma que diseñamos nuestros algoritmos para calcular o aproximar varias expresiones derivadas usando la teoría de la probabilidad. Segundo, podemos usar probabilidad y estadística para analizar de manera teórica el comportamiento de sistemas de IA.

Mientras que la teoría de la probabilidad nos permite enunciar con incertidumbre y razonar en la presencia de esta, la teoría de la información nos permite cuantificar la cantidad de incertidumbre en una distribución de probabilidad.

5.1 VARIABLES ALEATORIAS

Para la definición de Variable Aleatoria (v.a.) necesitamos introducir conceptos básicos de teoría de la medida [30].

La primera definición básica para la teoría de la probabilidad es el concepto de σ-álgebra que nos servirá de base para construir los demás conceptos.

Definición 13. Una familia de subconjuntos de X, denotada por Σ, se dirá que es una **σ-álgebra** sobre X si se cumplen las siguientes propiedades:

1. El conjunto vacío $\emptyset$ es un elemento de Σ.
2. Si A es un elemento de Σ, entonces el complementario de A, $\bar{A} = \Sigma \setminus A$ también pertenece a Σ.

3. Sea $A_1, A_2, \ldots$ una sucesión de elementos de Σ, entonces la **unión numerable** de todos los conjuntos de la sucesión también es un elemento de Σ, es decir: $\bigcup A_i \in \Sigma$.

Una vez definido lo que es una σ-álgebra y con el concepto de función de medida de Teoría de la Medida podemos definir el espacio muestral, que contempla todos los posibles resultados y escenarios de un experimento aleatorio.

Definición 14. Un **espacio muestral** se define como una tripleta $(\Omega, \mathcal{A}, \mu)$ tal que:

1. Ω es el conjunto de todos los sucesos elementales.
2. $\mathcal{A}$ es una familia de subconjuntos de Ω tal forma una σ-álgebra.
3. μ es una función de medida de conjuntos que permite asignar una probabilidad a los sucesos del espacio muestral.

Este concepto es clave de cara a los espacios probabilísticos, pues será la base para su definición y se le incorporará un conjunto de sucesos de interés sobre la que se define la función de probabilidad. El concepto de espacio de probabilidad o espacio probabilístico será crucial en el desarrollo de esta sección pues a partir de este se modelizan todos los experimentos aleatorios.

Definición 15. Un **espacio de probabilidad** se define como una tripleta $(\Omega, \mathcal{B}, P)$ tales que:

1. Ω es el espacio muestral (sucesos elementales).
2. $\mathcal{B}$ es una familia de sucesos aleatorios que forma una σ-álgebra.
3. P es una función de probabilidad que asigna una probabilidad a cada uno de los sucesos.

Para entender la definición anterior necesitaremos el concepto de función de probabilidad, la cual no es otra cosa que una extensión de función de medida verificando **Axiomas de Kolmogorov**.

Definición 16. Decimos que una función de medida P definida sobre una σ-álgebra $\mathcal{B}$ es una **medida o función de probabilidad** si verifica que:

1. **Axioma 1**: la función toma valores en el intervalo cerrado $[0, 1]$, es decir,

$$0 \leq P(A) \leq 1 \quad \forall A \in \mathcal{B}. \tag{24}$$

2. **Axioma 2**: la probabilidad del total es 1 y la del elemento vacío es 0, es decir,

$$P(\mathcal{B}) = 1 \quad P(\emptyset) = 0. \tag{25}$$

3. Si $A_1, A_2, \ldots$ son sucesos de $\mathcal{B}$ disjuntos dos a dos entonces la probabilidad de la unión es la suma de las probabilidades, es decir,

$$P(A_1 \cup A_2 \cup A_3 \cup \ldots) = \sum P(A_i). \tag{26}$$

En este punto contamos con las herramientas necesarias para introducir el concepto de v.a..

Definición 17. Una **Variable Aleatoria** (v.a.) X es una función real definida en el espacio de probabilidad $(\Omega, \mathcal{B}, P)$ asociada a un determinado experimento aleatorio, esto es

$$X : \Omega \to \mathbb{R}. \tag{27}$$

Con respecto al número de diferentes valores que puede tomar una variable, esta será:

1. **Discreta**: toma un conjunto finito o numerable de valores.
2. **Continua**: toma un conjunto no numerable de valores.

5.2 DISTRIBUCIONES DE PROBABILIDAD

Una **distribución de probabilidad** de una variable aleatoria es una función que asigna a cada valor posible de la v.a. una probabilidad (es decir, un valor real entre 0 y 1) de que este suceso ocurra.

En función del tipo de v.a. sobre el cual definamos la distribución de probabilidad podemos diferenciar dos tipos de distribuciones de probabilidad.

5.2.1 *Distribuciones de Probabilidad definidas sobre v.a. discretas*

Las distribuciones de probabilidad definidas sobre v.a. discretas se definen mediante la **Función Masa de Probabilidad (f.m.p.)**. Esta función asocia a cada punto del espacio muestral una probabilidad de que el suceso ocurra.

Definición 18. La **función masa de probabilidad (f.m.p.)** P definida sobre el espacio muestral $\mathcal{A}$ de una variable aleatoria X asigna a cada punto $x_i \in \mathcal{A}$ (suceso) una probabilidad de que este ocurra, es decir,

$$P(x_i) = p_i \tag{28}$$

donde p_i es la probabilidad del suceso $X = x_i$.

Una f.m.p. verifica las siguientes propiedades:

Proposición 5. *Sea P una función masa de probabilidad definida sobre una variable aleatoria discreta X, entonces:*

1. *El dominio de P serán todos los posibles valores de X.*

2. $0 \leq P(x) \leq 1 \forall x \in X$. *Así, una probabilidad de 0 indicará un suceso imposible, es decir, X no puede tomar ese valor. Una probabilidad de 1 indicará un suceso seguro, es decir, X siempre tomará ese valor.*
3. $\sum_{x \in X} P(x) = 1$. *Esta propiedad se conoce como* ***normalización*** *y es fundamental para cumplir los axiomas de Kolmogorov pues de no ser así podríamos tener sucesos con probabilidad mayor que 1 o que la probabilidad del total no fuese 1.*

5.2.2 *Distribuciones de Probabilidad definidas sobre v.a. continuas*

En el caso de las distribuciones de probabilidad definidas sobre una v.a. continua utilizamos la noción de Función de Densidad de Probabilidad (f.d.p.) para describirlas.

Definición 19. Una función p se dice que es una **función de densidad de probabilidad (f.d.p.)** sobre una v.a. continua X si cumple que:

- El dominio de p son todos los posibles valores de X.
- $p(x) \leq 1 \ \forall x \in X$.
- $\int_X p(x)dx = 1$.

En contraposición con la f.m.p., esta función no devuelve la probabilidad para un determinado punto del espacio muestra si no que devuelve la posibilidad de pertenecer a una región infinitesimal de volumen dx.

Proposición 6. *Para conocer la probabilidad de que nuestra v.a. X tome valores en un intervalo concreto, bastaría con integrar la f.d.p. en dicho intervalo. En el caso univariante y suponiendo que nuestra variable toma valores reales se podría denotar de la siguiente manera:*

$$P(x \in [a,b]) = P(a \leq x \leq b) = \int_a^b p(x)dx = \int_{[a,b]} p(x)dx. \tag{29}$$

5.3 PROBABILIDAD MARGINAL

A veces, conocemos la distribución de probabilidad de una distribución sobre un conjunto de varias variables pero nos interesa conocer la distribución sobre un subconjunto de ellas. La distribución resultante sería conocida como **distribución marginal de probabilidad** del subconjunto de variables elegidas.

A continuación mostramos como obtener las probabilidad marginales:

1. **Caso discreto**: consideraremos las v.a. discretas X e Y y su función de probabilidad conjunta $P(X,Y)$. Para obtener la distribución marginal de cualquiera de las v.a. bastaría con considerar la regla de la suma para la otra variable. Esto es,

$$\forall x \in X, \ P(X = x) = \sum_y P(X = x, Y = y). \tag{30}$$

2. **Caso continuo**: para v.a. continuas X e Y, conocida la función de densidad de probabilidad $p(X, Y)$. Para conocer la marginal respecto a una de sus variables, usaremos la integración en la otra variable. Es decir,

$$p(X) = \int_Y p(x, y)dy. \tag{31}$$

5.4 PROBABILIDAD CONDICIONADA

En múltiples ecuaciones nos interesará conocer la probabilidad de que un suceso ocurra en el caso en el que otro haya ocurrido. A esta probabilidad se le conoce como **probabilidad condicionada**. Denotamos a la probabilidad de que $X = x$ cuando sabemos que $Y = y$ como $P(X = x|Y = y)$. Esta probabilidad se calcula mediante la siguiente expresión:

$$P(X = x|Y = y) = \frac{P(X = x, Y = y)}{P(Y = y)} \tag{32}$$

donde $P(Y = y)$ es la probabilidad marginal de que la variable Y tome el valor y.

5.5 CONCEPTOS Y RESULTADOS BÁSICOS DE PROBABILIDAD

Un resultado elemental de la Teoría de la Probabilidad que nos resultará muy útil es la conocida **Regla de la Cadena**:

Proposición 7. ***Regla de la Cadena****. Toda probabilidad conjunta sobre un conjunto de v.a. puede ser descompuesta en distribuciones condicionales sobre una sola variable.*

Demostración. La demostración de esta proposición se puede decir que es una consecuencia directa de la definición de probabilidad condicionada. Aplicando de forma sucesiva obtenemos la siguiente descomposición:

$$P(X^{(1)}, X^{(2)}, \ldots, X^{(n)}) = P(X^{(1)}) \prod_{i=2}^{n} P(X^{(i)}|X^{(1)}, \ldots, X^{(i-1)}). \tag{33}$$

□

Otro concepto básico es el concepto de independencia entre dos v.a..

Definición 20. Dadas dos v.a. X e Y, diremos que son **independientes** si la distribución de probabilidad conjunta se puede descomponer como el producto de de las probabilidades de cada variable. Esto es,

$$P(X = x, Y = y) = P(X = x)P(Y = y) \quad \forall x \in X, y \in Y. \tag{34}$$

Diremos que un conjunto de v.a. son independientes cuando las variables lo son dos a dos.

De cara a los próximos estudios estadísticos y probabilísticos de la muestra de una v.a. nos van a resultar necesarias las siguientes medidas:

- **Esperanza matemática**: se define la esperanza matemática de una función f sobre una distribución de probabilidad $P(X)$ como el valor medio que toma f cuando la v.a. X sigue la distribución P. Dicho de manera formal:
  Definición 21. Se define la **esperanza matemática** en el caso discreto como

$$E[f(X)] = \sum_X P(X)f(X). \tag{35}$$

  Definición 22. Se define la **esperanza matemática** en el caso continuo como

$$E[f(X)] = \int_X P(x)f(x)dx. \tag{36}$$

- **Varianza**: nos da una medida de la dispersión de los datos. Formalmente se puede definir de la siguiente manera:
  Definición 23. Se define la **varianza** de una v.a. como

$$Var(f(X)) = E[(f(X) - E[f(X)])^2]. \tag{37}$$

  Dado que la varianza se define como la esperanza matemática del cuadrado de una función es directo ver que por definición la varianza siempre será no negativa.
- **Covarianza**: nos da una medida sobre cómo de relacionadas están dos v.a. de forma lineal. Formalmente se puede definir de la siguiente manera.
  Definición 24. Se define la **covarianza** entre una función $f(X)$ y una función $g(Y)$ con respecto a una distribución de probabilidad P como

$$Cov(f(X), g(Y)) = E[(f(X) - E[f(X)])(g(Y) - E[g(Y)])]. \tag{38}$$

  A partir de esta definición, cuando tenemos un vector aleatorio $X \in \mathbb{R}$ definimos la **matriz de covarianzas** cómo la matriz $n \times n$ que satisface que

$$Cov(X)_{i,j} = Cov(X^{(i)}, X^{(j)}). \tag{39}$$

  Claramente esta matriz es simétrica y su diagonal está compuesta por las varianzas de las componentes del vector aleatorio.

5.6 PROBABILIDAD BAYESIANA

En ocasiones, dadas dos variables aleatorias X e Y, nos vemos en el caso de conocer $P(Y|X)$ y queremos conocer $P(X|Y)$.

A esta última probabilidad se le conoce como probabilidad posterior y a su estudio se le conoce cómo Probabilidad Bayesiana Objetiva, la cuál es de gran importancia pues una gran cantidad de procedimientos de AA se basan en ella [8].

Para comenzar a desarrollar esta teoría comenzaremos enunciando el famoso **Teorema de Bayes**, que es el principio de la teoría y que nos permite calculas la probabilidad posterior deseada conociendo $P(X)$ de la siguiente forma.

Teorema 1. ***Teorema de Bayes****: dadas X e Y dos variables aleatorias, se cumple la siguiente igualdad:*

$$P(X|Y) = \frac{P(Y|X)P(X)}{P(Y)}. \tag{40}$$

Este teorema nos resultará de gran utilidad en la conocida inferencia Bayesiana, de la cuál se hace uso en *deep learning*.

5.7 TEORÍA DE LA INFORMACIÓN

La teoría de la información es una rama de las matemáticas aplicadas que gira entorno cuantificar cuanta información contiene una señal. Originalmente fue inventada para estudiar cómo mandar mensajes sobre un canal ruidoso, tales como una transmisión por radio. Esta rama cuenta con numerosas aplicaciones en la ingeniería informática, sin embargo nosotros nos centraremos en su aplicación para caracterizar distribuciones de probabilidad o para cuantificar la similitud entre estas.

Definición 25. Se define el concepto de **autoinformación** de un suceso o evento $X = x$ siendo X una v.a. bajo una cierta distribución de probabilidad P como

$$I(x) = -\log P(x). \tag{41}$$

Si nos fijamos, esta definición se centra en trabajar con un único resultado. Con el fin de extender este concepto introducimos la conocida **entropía de Shannon**.

Definición 26. Definimos la **entropía de Shannon** para medir la incertidumbre en toda la distribución de probabilidad como

$$H(X) = E[I(X)] = -E[\log P(X)] = H(P). \tag{42}$$

En el caso de que la distribución de nuestra v.a. X sea continua, la entropía de Shannon recibirá el nombre de **entropía diferencial**.

En el caso de que tengamos dos distribuciones de probabilidad diferentes, $P(X)$ y $Q(X)$, sobre la misma v.a. X, podemos medir cómo de distintas son estas dos distribuciones usando la divergencia de Kullback-Leibler.

Definición 27. En el contexto anterior, se define la **divergencia de Kullback-Leibler (divergencia KL)** como

$$D_{KL}(P||Q) = E[\log \frac{P(x)}{Q(x)}] = E[\log P(x) - \log Q(x)]. \tag{43}$$

Esta medida se puede simplificar mediante la siguiente medida.

Definición 28. En el contexto anterior, se define la **entropía cruzada** como

$$H(P, Q) = -E_{X\sim P}[\log Q(X)]. \tag{44}$$

Proposición 8. *Otra definición alternativa de la entropía cruzada es*

$$H(P, Q) = H(P) + D_{KL}(P||Q). \tag{45}$$

Esta nueva noción es interesante pues minimizar la entropía cruzada respecto a la distribución Q es equivalente a minimizar la divergencia KL, dado que Q no participa en el término omitido.

5.8 ESTIMACIÓN DE MÁXIMA VEROSIMILITUD

Durante el estudio de los experimentos aleatorios [7], muchas veces querremos obtener conclusiones acerca del comportamiento de una o varias varías características de nuestra población basándonos en la observación de las mismas en un subconjunto de la población original. La inferencia estadística se centra en resolver este tipo de problemas.

En todo problema de inferencia estadística existen usa serie de elementos base del problema:

- **La población**: es el conjunto de elementos en el que se pretende estudiar una determinada característica.
- **La característica que se desea estudiar**: se suele representar por la v.a. que la cuantifica.
- **La muestra de la que se dispone para el estudio**: es un subconjunto de la población sobre el que se va a estudiar la característica para inferir las conclusiones.

Normalmente basándonos en la muestra observada intentaremos descubrir la distribución desconocida F de la v.a. involucrada en el problema. A F se le conocerá por el nombre de **distribución teórica**. Según el conocimiento previo que se tenga sobre la distribución teórica se pueden plantear dos situaciones:

1. Se conoce la forma de la función de distribución teórica salvo uno o varios parámetros, es decir, se sabe que F pertenece a la familia de funciones de distribuciones
$$F \in \{F_\theta, \theta \in \Theta\}, \tag{46}$$
donde F_θ tiene una forma funcional fija y conocida dependiente de uno o varios parámetros θ, que se mueve dentro de $\Theta \subset \mathbb{R}^k$ conocido como espacio paramétrico. A este caso se le conoce como **inferencia paramétrica**.
2. No se conoce nada acerca de F salvo aspectos muy generales como que la v.a. sea discreta o continua, la existencia o no existencia de momentos o aspectos similares. A este caso se le conoce como**inferencia no paramétrica**.

Nosotros, de cara a las futuras aplicaciones en AA nos centraremos en el primer caso, en la **inferencia paramétrica**. Además nos centraremos en la estimación puntual, es decir, buscar valores concretos θ dentro de nuestro espacio paramétrico Θ.

Primero hablaremos de la muestra que queremos estudiar y le aplicaremos algunas restricciones que nos harán más cómodo trabajar con ella.

Definición 29. Definimos una **muestra aleatoria simple**, de tamaño n, de una v.a. X con distribución teórica F, como un vector $(X_1, \ldots, X_n)$ formado por n v.a. Independientes e Idénticamente Distribuidas (i.i.d.), es decir, con función de distribución común F.

La ventaja de trabajar con v.a. i.i.d. es que la función de distribución conjunta del vector aleatorio formado por dichas variables será igual al producto de la distribuciones marginales de cada una de ellas, que al tener la misma distribución se tiene que

$$F_{(X_1,\ldots,X_n)}(x_1, \ldots, x_n) = F_{X_1}(x_1) \ldots F_{X_n}(x_n) = \prod_{i=1}^{n} F_X(x_i), \quad (x_1, \ldots, x_n) \in \mathbb{R}^n. \tag{47}$$

Definición 30. Definimos el concepto de **estimador**, $T(X_1, \ldots, X_n)$ cómo una función que toma la muestra y devuelve valores del espacio paramétrico, es decir,

$$T : X^n \to \Theta. \tag{48}$$

Definición 31. Un estimador $T(X_1, \ldots, X_n)$ de θ se dice insesgado o centrado en el parámetro θ si
$$E_\theta[T(X_1, \ldots, X_n)] = \theta, \quad \forall \theta \in \Theta. \tag{49}$$

Nuestro objetivo será encontrar un estimador con propiedades suficientemente buenas. Como vemos la definición anterior no plantea apenas ninguna condición para que una función sea un estimador, es por tanto que debemos de imponer restricciones a este. Para ello daremos la noción de función de verosimilitud.

Definición 32. Sea X una v.a. con distribución en una familia paramétrica de distribuciones $\{F_\theta, \theta \in \Theta\}$. Sea $f_\theta(x)$ la f.m.p. o la f.d.p. de X. Sea $X_1, \ldots, X_n$ una muestra

aleatoria simple de X y $f_\theta^n(x_1, \ldots, x_n)$ su f.m.p. o f.d.p. conjunta con $\theta \in \Theta$. Para cada realización muestral se define la **función de verosimilitud** asociada a dichos valores de la muestra como una función de θ de la siguiente forma:

$$L_{x_1,\ldots,x_n} : \Theta \to \mathbb{R}^+ \cup \{0\} \tag{50}$$

$$L_{x_1,\ldots,x_n}(\theta) = f_\theta^n(x_1, \ldots, x_n). \tag{51}$$

Definición 33. En las condiciones de la definición previa, se define el **estimador de máxima verosimilitud** de θ como aquel estimador $\hat{\theta}(X_1, \ldots, X_n)$ que verifica que

$$\forall (x_1, \ldots, x_n) \in X^n, \quad L_{x_1,\ldots,x_n}(\hat{\theta}(x_1, \ldots, x_n)) = \max_{\theta \in \Theta} L_{x_1,\ldots,x_n}(\theta). \tag{52}$$

Podemos ver este este estimador de manera intuitiva como aquel estimador que, dada una muestra, nos da el parámetro θ que hace que sea más probable observar los datos de la muestra. Además este estimador es muy interesante porque cuenta con muy buenas propiedades, veamos algunas de ellas.

Proposición 9. *Bajo condiciones bastantes generales, si las ecuaciones de verosimilitud tienen solución única, $\hat{\theta}(X_1, \ldots, X_n)$, esta solución es fuertemente consistente, es decir, cuando la muestra tiende a infinito, el estimador converge al parámetro de forma casi segura.*

$$\hat{\theta}(X_1, \ldots, X_n) \xrightarrow[n\to\infty]{c.s.} \theta, \quad \forall \theta \in \Theta \tag{53}$$

Teorema 2. ***Teorema de invarianza de Zehna**: sea X una v.a. con distribución en una familia paramétrica de distribuciones $\{F_\theta, \theta \in \Theta\}$. Sea $X_1, \ldots, X_n$ una muestra aleatoria simple de X. Sea g una función medible. Si $\hat{\theta}(X_1, \ldots, X_n)$ es el estimador de máximo verosimilitud de θ, entonces $g(\hat{\theta}(X_1, \ldots, X_n))$ lo será de $g(\theta)$.*

Estas propiedades harán del estimador de máximo verosimilitud uno muy deseable a calcular. Es bastante común en el *deep learning* desear minimizar la función negativa de la función de verosimilitud o la entropía cruzada con el fin de encontrar los pesos (o en este caso los parámetros) de nuestro modelo. Esto implica un problema de minimización, por lo que estudiaremos métodos para ello a continuación.

6

OPTIMIZACIÓN NO LINEAL

Gran parte de los problemas de AA consisten en minimizar una función de pérdida respecto a una serie de parámetros. En este capítulo estudiaremos algunas técnicas para resolver problemas de optimización no restringida en dimensión finita [17, 32, 26]. Empezaremos formulando el problema.

Dada una función $f : \mathcal{D} \subset \mathbb{R}^n \to \mathbb{R}$; $f \in C^1(\mathcal{D})$ y $\mathcal{D}$ abierto, buscamos un mínimo (local) x^* de f. Escribiremos este problema cómo:

$$\min_{x \in \mathcal{D}} f(x). \tag{54}$$

Definición 34. Un elemento x^* se dirá que es un mínimo local de f si existe $\delta > 0$ tal que $f(x^*) \leq f(x)$ para todo x que cumpla que $||x^* - x|| < \delta$.

La existencia de la solución solo se puede garantizar bajo ciertas condiciones (como contraejemplo bastaría con considerar $n = 1$, $f(x) = e^x$). Para ello primero debemos hablar de diferenciación de funciones.

Definición 35. Diremos que una función $f : \mathbb{R}^n \to \mathbb{R}$ es diferenciable en un punto x si podemos encontrar un vector $a \in \mathbb{R}^n$ tal que para todo $y \in \mathbb{R}^n$,

$$f(x + y) = f(x) + < a, y > + o(y) \tag{55}$$

donde $< a, y >$ denota el producto escalar entre a y y . El vector a es conocido como el gradiente de $f(x)$ en x y se denota por $\nabla f(x)$.

En otras palabras, una función es diferenciable en un punto x si admite una aproximación lineal de primer orden en x. Hemos usado la notación o pequeña que denota que la función o es mucho más pequeña que y cuando $y \to 0$. Esta definición se usa en lugar de usar un límite con el fin de dar una mayor intuición geométrica.

Está claro que el gradiente está únicamente determinado siendo sus componentes

$$\nabla f_i(x) = \frac{\partial f(x)}{\partial x_i}. \tag{56}$$

Supongamos que $f(x)$ es diferenciable en el segmento $[x, x+y]$. Consideramos entonces la función de una variable $\phi(t) = f(x+ty)$ y calculamos su derivada para $0 \leq t \leq 1$.

$$\frac{\phi(t+\varepsilon) - \phi(t)}{\varepsilon} = \frac{f(x+(t+\varepsilon)y) - f(x+ty)}{\epsilon}, \tag{57}$$

$$= \frac{< \nabla f(x+ty), \varepsilon y > + o(\varepsilon y)}{\varepsilon} \tag{58}$$

$$\phi'(t) = \lim_{\varepsilon \to 0} \frac{\phi(t+\varepsilon) - \phi(t)}{\varepsilon} =< \nabla f(x+ty), y > \tag{59}$$

Definición 36. A la cantidad

$$f'(x;y) = \lim_{\varepsilon \to 0} \frac{f(x+\varepsilon y) - f(x)}{\varepsilon}$$

se le llama derivada direccional de $f(x)$ en la dirección de y. Si $f(x)$ tiene derivada direccional en todas las direcciones entonces se dice que $f(x)$ es diferenciable Gateaux en el punto x.

Se deduce de la ecuación 59 que si $f(x)$ es diferenciable en x, también es diferenciable Gateaux, con

$$f'(x;y) = \phi'(0) =< \nabla f(x), y > . \tag{60}$$

Además, si una función $f(x)$ es diferenciable en $[x, x+y]$, entonces, usando la ecuación 59 y la igualdad:

$$\phi(1) = \phi(0) + \int_0^1 \phi'(t)dt \tag{61}$$

obtenemos que

$$\begin{aligned} f(x+y) &= f(x) + \int_0^1 < \nabla f(x+ty), y > dt \\ = f(x) + < \nabla f(x), y > &+ \int_0^1 < \nabla f(x+ty) - \nabla f(x), y > dt. \end{aligned} \tag{62}$$

Definición 37. Diremos que una función $f(x)$ en $\mathcal{R}^n$ es doblemente diferenciable en un punto x si es diferenciable en x y si podemos encontrar una matríz simétrica H de tamaño $n \times n$ tal que para todo $y \in \mathcal{R}^n$ se cumple que

$$f(x+y) = f(x) + \langle \nabla f(x), y \rangle + \frac{\langle Hy, y \rangle}{2} + o(||y||^2). \tag{63}$$

Esta matriz H se conoce como Hessiana y se denotará por $\nabla^2 f(x)$ o $f''(t)$. En otras palabras, una función es doblemente diferenciable en un punto x si admite una aproximación cuadrática en un entorno de x.

Consideramos de nuevo la función escalar $\phi(t) = f(x + ty)$. Asumimos que f es doblemente derivable. Procediendo de igual manera que en el caso anterior, se puede ver que

$$\phi''(t) = \langle \nabla^2 f(x + ty)y, y\rangle. \tag{64}$$

Por lo tanto, si usamos la formula de Taylor expresando el resto en la forma de Lagrange tenemos que

$$\phi(1) = \phi(0) + \phi'(0) + \frac{\phi''(\xi)}{2}, \quad 0 \leq \xi \leq 1 \tag{65}$$

y por tanto podemos encontrar un ξ tal que

$$f(x + y) = f(x) + \langle \nabla f(x), y\rangle + \frac{\langle \nabla^2 f(x + \xi y)y, y\rangle}{2}. \tag{66}$$

Una vez conocidas las nociones anteriores, podemos afirmar que una **condición suficiente para la existencia** de al menos un mínimo local es la siguiente:

Proposición 10. *Si existe un $x^0 \in \mathcal{D}$ tal que $\mathcal{L}_f(x^0) = \{x \in \mathcal{D} : f(x) \leq f(x^0)\}$ es compacto, (es decir, cerrado y acotado pues nos encontramos en dimensión finita) entonces f tiene un mínimo local x^* .*

Demostración. Es consecuencia directa de la compacidad del conjunto $\mathcal{L}_f(x^0)$. □

Daremos también una **condición necesaria** para que un punto sea un mínimo local.

Teorema 3. *Si f es continuamente diferenciable en un entorno de x^* y x^* es un mínimo local de f, entonces $\nabla f(x^*) = 0$.*

Demostración. Supongamos que $\nabla f(x^*) \neq 0$. Entonces podemos encontrar $\varepsilon > 0$ lo suficientemente pequeño tal que

$$\begin{aligned} f(x^* - \varepsilon \nabla f(x^*)) &= f(x^*) - \varepsilon ||\nabla f(x^*)||^2 + o(\varepsilon \nabla f(x^*)) \\ &= f(x^*) - \varepsilon(||\nabla f(x^*)||^2 + \varepsilon^{-1} o(\varepsilon)) < f(x^*). \end{aligned} \tag{67}$$

Pero esto es una contradicción que x^* sea un mínimo local. □

Teorema 4. *Sea $f : \mathcal{D} \subset \mathbb{R}^n \to \mathbb{R}$; $f \in C^1(\mathcal{D})$, $\mathcal{D}$ abierto y $x^* \in \mathcal{D}$. Supongamos que x^* es un mínimo local de f, entonces se cumple que*

$$\nabla f(x^*) = 0. \tag{68}$$

Además, si $f \in C^2(\mathcal{D})$, entonces lo siguiente también se cumple $\nabla^2 f(x^*)$ es semidefinida positiva.

Demostración. Por el teorema 3, $\nabla f(x^*) = 0$ y por lo tanto para un y arbitrario y un ε lo suficientemente pequeño tenemos que:

$$f(x^*) \leq f(x^* + \varepsilon y) = f(x^*) + \varepsilon^2 \frac{\langle \nabla^2 f(x^*) y, y \rangle}{2} + o(\varepsilon^2), \tag{69}$$

$$\langle \nabla^2 f(x^*) y, y \rangle \geq o(\varepsilon^2)/\varepsilon^2 \tag{70}$$

Pasando al límite con $\varepsilon \to 0$, obtenemos que $\langle \nabla^2 f(x^*) y, y \rangle \geq 0$. Como y es arbitrario, concluimos la prueba. □

Nos interesará también conocer cuando nuestro mínimo local es también global, para ello daremos una condición suficiente basada en la convexidad de nuestro dominio y nuestra función.

Definición 38. $\mathcal{D} \subset \mathbb{R}^n$ se dice **convexo**, si dados dos puntos $x, y \in \mathcal{D}$ entonces $\lambda x + (1-\lambda) y \in \mathcal{D}, \lambda \in [0,1]$.

Definición 39. $f : \mathcal{D} \subset \mathbb{R}^n \to \mathcal{R}$, se dice **convexa** en $\mathcal{D}$ si dados $x, y \in \mathcal{D}$

$$\lambda f(x) + (1-\lambda) f(y) \geq f(\lambda x + (1-\lambda) y) \tag{71}$$

con $\lambda \in [0,1]$.

Teorema 5. *Si $\mathcal{D} \neq \emptyset \subset \mathbb{R}^n$ es convexo y f es convexa en $\mathcal{D}$, entonces todo mínimo local es también un mínimo global.*

Demostración. Sea x^* un mínimo local, entonces existe $\delta > 0$ tal que

$$||y - x^*|| \leq \delta \implies f(x^*) \leq f(y). \tag{72}$$

Supongamos $x \in \mathcal{D}$ arbitrario. Consideremos el punto $x^* + t(x - x^*) \in \mathcal{D}$ por ser convexo. Por ser f convexa tenemos entonces que

$$f(x^* + t(x - x^*)) \leq (1-t) f(x^*) + t f(x) \tag{73}$$

Cogiendo un t tal que $0 < t < \delta / (||x|| + ||x^*||)$ tenemos que

$$||(x^* + t(x - x^*)) - x^*|| \leq \delta. \tag{74}$$

Por lo tanto

$$f(x^*) \leq f(x^* + t(x - x^*)) \leq (1-t) f(x^*) + t f(x). \tag{75}$$

Operando tenemos que

$$f(x^*) + (t-1) f(x^*) \leq t f(x) \tag{76}$$

$$t f(x^*) \leq t f(x) \implies f(x^*) \leq f(x), \tag{77}$$

donde x era arbitrario llegando a nuestra conclusión. □

6.1 DESCENSO POR EL GRADIENTE

Ahora vamos a analizar un método de optimización no restringida: el descenso por el gradiente. Este método, aunque raramente se implementa en su forma más pura, es un modelo para construir algoritmos más realistas, tales como el famoso descenso estocástico por el gradiente, el cual es el método por excelencia para optimizar funciones en el AA. Daremos una prueba para su convergencia, y discutiremos los aspectos teóricos y los de implementación de estos métodos.

Supongamos que para cualquier punto x, podemos calcular el gradiente de una función $\nabla f(x)$. En este caso, el método más simple para minimizar $f(x)$ es el método del descenso por el gradiente, en el que, comenzando en una aproximación inicial x^0, construimos una sucesión

$$x^{k+1} = x^k - \gamma_k \nabla f(x^k) \tag{78}$$

donde el parámetro $\gamma_k \geq 0$ es el tamaño del paso. Existen varias maneras de deducir el método 78.

Primero, si recordamos la condición necesaria de primer orden (Teorema 3) y su demostración, tenemos que si la condición no se cumple en x, es decir, $\nabla f(x) \neq 0$, entonces el valor de la función puede reducirse mediante el punto $x - \varepsilon \nabla f(x)$ con un valor de ε lo suficientemente pequeño. Aplicando este procedimiento de manera secuencial, es como llegamos a este método.

Segundo, en un punto x^k la función diferenciable $f(x)$ puede ser aproximada por la función lineal $f_k(x) = f(x^k) + < \nabla f(x^k), x - x^k >$ en términos del orden de $o(x - x^k)$. Por lo tanto, podemos buscar el mínimo de la aproximación $f_k(x)$ en un entorno de x^k. Por lo tanto sería natural adoptar su solución como la nueva aproximación x^{k+1}.

Consideremos ahora la variante más sencilla de este método en la que $\gamma_k \equiv \gamma$:

$$x^{k+1} = x^k - \gamma \nabla f(x^k). \tag{79}$$

Queremos observar el comportamiento de este método bajo ciertas suposiciones sobre $f(x)$ y γ.

Teorema 6. *Sea $f(x)$ diferenciable en $\mathbb{R}^n$, y que el gradiente de $f(x)$ cumpla la condición de Lipschitz:*

$$||\nabla f(x) - \nabla f(y)|| \leq L||x - y||$$

$f(x)$ esté acotada inferiormente:

$$f(x) \geq f^* > -\infty$$

y que γ cumpla que

$$0 < \gamma < 2/L$$

.

Entonces, el método del gradiente tiende a cero:

$$\lim_{k\to\infty} \nabla f(x^k) = 0$$

Demostración. Si sustituimos $x = x^k$ y $y = -\gamma\nabla f(x^k)$ en la ecuación (62) y usamos la ecuación (60) obtenemos que:

$$\begin{aligned} f(x^{k+1}) = f(x^k) - \gamma||\nabla f(x^k)||^2 - \gamma\int_0^1 < \nabla f(x^k - t\gamma\nabla f(x^k)), \nabla f(x^k) > dt \\ \leq f(x^k) - \gamma||\nabla f(x^k)||^2 + L\gamma^2||\nabla f(x^k)||^2\int_0^1 tdt \\ = f(x^k) - \gamma(1 - \frac{1}{2}L\gamma)||\nabla f(x^k)||^2. \end{aligned} \tag{80}$$

Por lo tanto tenemos que:

$$f(x^{k+1}) \leq f(x^k) - \gamma(1 - \frac{1}{2}L\gamma)||\nabla f(x^k)||^2. \tag{81}$$

Podemos sumar estas desigualdades desde o hasta s para obtener lo siguiente:

$$f(x^{s+1}) \leq f(x^0) - \gamma(1 - \frac{1}{2}L\gamma)\sum_{k=0}^{s}||\nabla f(x^k)||^2. \tag{82}$$

Sabiendo por (62) que $\alpha = \gamma(1 - \frac{1}{2}L\gamma) > 0$, podemos despejar la suma para obtener que

$$\sum_{k=0}^{s}||\nabla f(x^k)||^2 \leq \alpha^{-1}(f(x^0) - f(x^{s+1})) \leq \alpha^{-1}(f(x^0) - f^*) \tag{83}$$

para toda s. Por lo tanto la suma es finita y por ello el término principal debe de tender a o, es decir, $||\nabla f(x^k)|| \to 0$. □

Este método nos da una manera de obtener una sucesión que converja a un punto crítico de la función. Sin embargo debemos de recordar que el hecho de que $\nabla f(x) = 0$ si bien es una condición necesaria no es suficiente para que x sea un mínimo. El contraejemplo por excelencia para este tipo de situaciones son los famosos puntos de silla. Es decir, el descenso por el gradiente puede "quedarse atascado" en cualquier punto estacionario, ya sea este un mínimo o un punto de silla. Además, este método no "distingue" entre un mínimo global o un mínimo local y no hay ningún tipo de garantía a la convergencia de un mínimo global.

Aún así, el método del descenso por el gradiente es la base para el principal método de optimización usado en el AA, el descenso estocástico por el gradiente. En un problema de AA, normalmente intentamos minimizar una función de pérdida o coste, que se suele calcular como la suma de esta en cada uno de los valores de la muestra que usamos.

El descenso estocástico por el gradiente [10] es un método de optimización basado en el descenso por el gradiente ya expuesto en el que, en lugar de calcular el gradiente, se hace uso de un estimador insesgado del gradiente de la función de coste mediante el uso una muestra llamada *minibatch* $\{x^{(1)}, \ldots, x^{(m)}\}$. Formalmente sería:

$$\hat{g} = \frac{1}{m} \nabla_\theta \sum_i L(f(x^{(i)}; \theta), y^{(i)}) \tag{84}$$

donde actualizaríamos θ en la dirección de $\hat{g}$. Aquí, L es la función de coste o pérdida, $y^{(i)}$ será lo que próximamente llamemos etiqueta de $x^{(i)}$ y $f(x; \theta)$ es un funcional.

7

OPERADOR DE AGREGACIÓN KRUM

Un problema común que nos encontraremos más adelante es el de agregar una serie de estimaciones de un gradiente para obtener un nuevo vector que nos permita resumir la información contenida por los vectores anteriores. Esto es, obtener una estimación del gradiente de una función mediante estimaciones previas en un algoritmo iterativo.

En nuestro caso podemos considerar que tenemos n vectores, de los cuales f pueden ser bizantinos, esto es, con valores arbitrarios. En cada paso t de nuestro algoritmo, se considera un vector real denotado por $x_t \in \mathbb{R}^d$ de parámetros. Cada vector consiste en un valor estimado $V_p^t = G(x_t, \xi_p^t)$ del gradiente $\nabla Q(x_t)$ de la función de coste Q, donde ξ_p^t es una variable aleatoria representando una muestra de un conjunto de datos. Sin embargo, un vector bizantino V_b^t se puede desviar arbitrariamente del vector esperado.

Dadas las estimaciones podemos calcular un nuevo vector $F(V_1^t, \ldots, V_n^t)$ mediante una función determinista $F : (\mathbb{R}^d)^n \to \mathbb{R}^d$ llamada regla de agregación y los vectores estimados. Finalmente, podemos actualizar nuestros parámetros mediante la siguiente ecuación basada en el descenso estocástico por el gradiente

$$x_{t+1} = x_t - \eta_t \cdot F(V_1^t, \ldots, V_n^t). \tag{85}$$

Esperamos que los vectores no bizantinos (benignos) sean estimaciones insesgadas del gradiente $\nabla Q(x_t)$. De manera más precisa, en cada ronda t, los vectores benignos V_i^t son vectores aleatorios i.i.d. con $V_i^t \sim G(x_t, \xi_i^t)$ tales que $E_{\xi_i^t}[G(x_t, \xi_i^t)] = \nabla Q(x_t)$.

Como veremos más adelante en esta memoria, la regla de agregación más común consiste en calcular la media de los vectores. Sin embargo, ninguna combinación lineal de los vectores puede tolerar a un vector bizantino, y por tanto esta regla no es resistente a ataques bizantinos.

Lema 1. *Sea F_{lin} una regla de agregación de la forma $F_{lin}(V_1, \ldots, V_n) = \sum_{i=1}^n \lambda_i \cdot V_i$ donde λ_i son escalares distintos de cero. Sea $U \in \mathbb{R}^d$. Entonces, un solo vector bizantino puede hacer que F siempre compute U como salida. Particularmente, un único vector bizantino puede evitar la convergencia del modelo.*

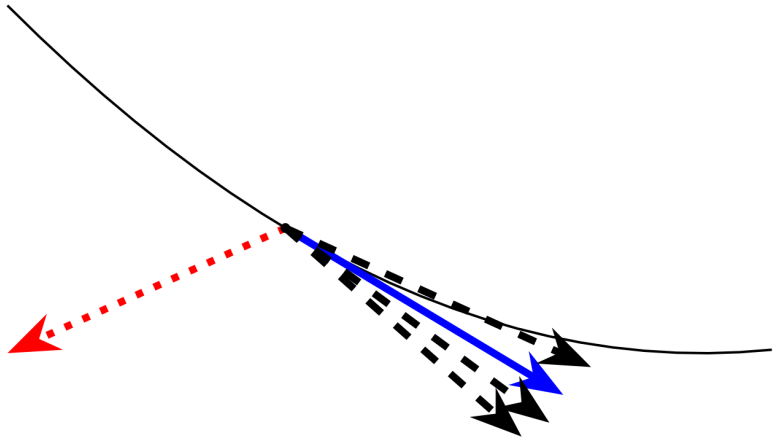


Figura 3: Las estimaciones del gradiente producidas por los estimadores correctos (flechas discontinuas negras) están distribuidas alrededor del gradiente (flecha azul sólida). Un vector bizantino puede tener un valor arbitrario (flecha roja discontinua). Fuente: [2].

Demostración. La demostración resulta inmediata. Dado que se puede asumir que se tiene conocimiento total sobre los vectores V_i y la regla de agregación F_{lin} (y por tanto de los escalares λ_i), puede proponer el el vector $V_n = \frac{1}{\lambda_n} \cdot U - \sum_{i=1}^{n-1} \frac{\lambda_i}{\lambda_n} V_i$, haciendo así que $F_{lin}(V_1, \ldots, V_n) = U$. □

Viendo esto, nuestro objetivo es buscar una regla de agregación resistente a ataques bizantinos. De manera intuitiva, queremos que el resultado (esperado) F de nuestra regla no se aleje demasiado del gradiente "real" g. Esto se puede expresar mediante el uso de una cota inferior del producto escalar del vector F y g, así acotando el ángulo que ambos forman. Particularmente, si $E[F]$ se encuentra en una bola centrada en g con radio r, entonces el producto escalar está acotado inferiormente por un término que implica a $\sin \alpha = r/||g||$.

Otra condición que buscamos, aunque más técnica, es que los momentos de F deben de depender de los momentos del estimador (correcto) del gradiente G. Esto se debe a que las cotas de los momentos de G se suelen usar para controlar los efectos de la naturaleza discreta del método del descenso estocástico por el gradiente.

Definición 40. Sea $0 \leq \alpha < \pi/2$ cualquier ángulo, y cualquier entero $0 \leq f \leq n$. Sean $V_1, \ldots, V_n$ vectores aleatorios i.i.d. en $\mathbb{R}^d$, $V_i \sim G$, con $E[G] = g$. Sean $B_1, \ldots, B_f$ vectores aleatorios en $\mathbb{R}^d$, posiblemente dependientes de los vectores V_i. Diremos

que una regla de agregación F es (α, f)-Resistente bizantina si, para cualesquiera $1 \leq j_1 < \ldots < j_f \leq n$, el vector

$$F = F(V_1, \ldots, \underbrace{B_1}_{j_1}, \ldots, \underbrace{B_f}_{j_f}, \ldots, V_n) \tag{86}$$

cumple que $\langle E[F], g \rangle \geq (1 - \sin\alpha) \cdot ||g||^2 > 0$ y que para $r \in \{2, 3, 4\}$, $E[||F||^r]$ está superiormente acotada por una combinación linear de los términos $E[||G||^1], \ldots, E[||G||^{r_{n-1}}]$ con $r_1 + \ldots + r_{n-1} = r$.

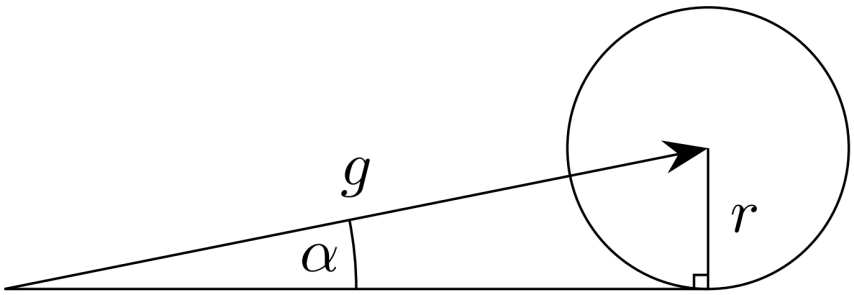


Figura 4: Si $||E[F - g]|| \leq r$, entonces $\langle E[F], g \rangle$ está inferiormente acotado por $(1 - \sin\alpha)||g||^2$ donde $\sin\alpha = r/||g||$. Fuente: [2].

Tras haber dado una definición formal de lo que es una regla de agregación resistente, procedemos a introducir la regla de agregación Krum, que cumplirá la condición ya explicada. La idea detrás de Krum consiste en que calcular el promedio puede verse también como calcular el vector que minimiza la suma de las distancias al cuadrado de los vectores V_i. La idea de Krum vuelve a ser la de encontrar un vector que minimice sumas de distancias a los demás vectores, pero excluyendo de estas a los vectores que estén ´´muy lejos´´ de manera que un vector bizantino no pueda forzar a la regla a escoger un vector arbitrario.

Formalmente, se define la regla de agregación Krum $KR(V_1, \ldots, V_n)$ de la siguiente manera. Para todos $i \neq j$, denotamos por $i \to j$ el hecho de que el vector V_j está en el conjunto de los $n - f - 2$ vectores más cercanos a V_i. Para cada vector i se define la siguiente puntuación

$$s(i) = \sum_{i \to j} ||V_i - V_j||^2. \tag{87}$$

Finalmente, $KR(V_1, \ldots, V_n) = V_{i_*}$ donde $s(i_*) \leq s(i)$ para todo i.

A continuación daremos una condición sobre n, f y el estimador del gradiente usado para que la regla Krum sea (α, f)-resistente bizantina.

Teorema 7. *Sean $V_1, \dots, V_n$ vectores aleatorios d-dimensionales i.i.d. tales que $V_i \sim G$ con $E[G] = g$ y $E[||G - g||^2] = d\sigma^2$. Sean $B_1, \dots, B_f$ vectores aleatorios, posiblemente dependientes de los vectores V_i. Si $2f + 2 < n$ y $\eta(n, f)\sqrt{d} \cdot \sigma < ||g||$, donde*

$$\eta(n, f) = \sqrt{2\left(n - f + \frac{f \cdot (n - f - 2) + f^2 \cdot (n - f - 1)}{n - 2f - 2}\right)}, \tag{88}$$

entonces la regla de agregación Krum KR es (α, f)-resistente bizantina, donde el α se define como

$$\sin \alpha = \frac{\eta(n, f) \cdot \sqrt{d}\sigma}{||g||}. \tag{89}$$

Demostración. Sin perder generalidad, podemos asumir que los vectores $B_1, \dots, B_f$ ocupan las últimas f posiciones de la lista de argumentos de KR. Diremos que un índice es correcto si se refiere a un vector entre $V_1, \dots, V_{n-f}$ y bizantino si se refiere a un vector entre $B_1, \dots, B_f$. Para cada índice i, denotamos por $\delta_c(i)$ (o $\delta_b(i)$) a la cantidad de índices correctos (o bizantinos) j tales que $i \to j$. Claramente tenemos que

$$\begin{aligned} \delta_c(i) + \delta_b(i) &= n - f - 2 \\ n - 2f - 2 \leq \sigma_c(i) &\leq n - f - 2 \\ \sigma_b(i) &\leq f. \end{aligned} \tag{90}$$

Nos centraremos primero en acotar el producto escalar. Empezaremos dando una cota superior de $||E[KR - g]||^2$. Recordemos que, si j es correcto, entonces $E[V_j] = g$. Sea i_* el índice del vector elegido por la regla Krum.

$$\begin{aligned} ||E[KR - g]||^2 &\leq ||E\left(KR - \frac{1}{\delta_c(i_*)} \sum_{i_* \to j \text{ correctos}} V_j\right)||^2 \\ &\leq E[||KR - \frac{1}{\delta_c(i_*)} \sum_{i_* \to j \text{ correctos}} V_j||^2] \quad \text{(Desigualdad de Jensen)} \\ &\leq \sum_{i \text{ correcto}} E[||KR - \frac{1}{\delta_c(i_*)} \sum_{i_* \to j \text{ correctos}} V_j||^2]\mathbb{1}(i_* = i) \\ &\quad + \sum_{k \text{ biz}} E[||B_k - \frac{1}{\delta_c(k)} \sum_{k \to j \text{ correctos}} V_j||^2]\mathbb{1}(i_* = k) \end{aligned} \tag{91}$$

Ahora analizamos el caso $i_* = i$ con i un índice correcto.

$$\begin{aligned} ||V_i - \frac{1}{\delta_c(i)} \sum_{i_* \to j \text{ correctos}} V_j||^2 &= ||\frac{1}{\delta_c(i)} \sum_{i_* \to j \text{ correctos}} V_i - V_j||^2 \\ &\leq \frac{1}{\delta_c(i)} \sum_{i_* \to j \text{ correctos}} ||V_i - V_j||^2 \end{aligned} \tag{92}$$

y por tanto

$$\begin{aligned} E[||V_i - \frac{1}{\delta_c(i)} \sum_{i_* \to j \text{ correctos}} V_j||^2] &\leq \frac{1}{\delta_c(i)} \sum_{i_* \to j \text{ correctos}} E[||V_i - V_j||^2] \\ &\leq 2d\sigma^2. \end{aligned} \tag{93}$$

Ahora analizamos el caso $i_* = k$ siendo k un índice bizantino. El hecho de que k minimiza la puntuación significa que para todos los índices correctos i

$$\begin{aligned} &\sum_{k \to j \text{ correctos}} ||B_k - V_j||^2 + \sum_{k \to l \text{ bizantinos}} ||B_k - B_l||^2 \\ &\leq \sum_{i \to j \text{ correctos}} ||V_i - V_j||^2 + \sum_{i \to l \text{ bizantinos}} ||V_i - B_l||^2. \end{aligned} \tag{94}$$

Entonces, para todo índice correcto i

$$\begin{aligned} &||B_k - \frac{1}{\delta_c(k)} \sum_{k \to j \text{ correctos}} V_j||^2 \leq \frac{1}{\delta_c(k)} \sum_{k \to j \text{ correctos}} ||B_k - V_j||^2 \\ &\leq \frac{1}{\delta_c(k)} \sum_{i \to j \text{ correctos}} ||V_i - V_j||^2 + \frac{1}{\delta_c(k)} \underbrace{\sum_{i \to l \text{ bizantinos}} ||V_i - B_l||^2}_{D^2(i)}. \end{aligned} \tag{95}$$

Nos fijamos a continuación en el término $D^2(i)$. Cada vector correcto i tiene $n - f - 2$ vecinos, y $f + 1$ no vecinos. Por lo tanto, debe de existir un vector correcto $\zeta(i)$ que está más lejos de i que cualquiera de sus vecinos. En particular, para cada índice bizantino l tal que $i \to l$, $||V_i - B_l||^2 \leq ||V_i - V_{\zeta(i)}||^2$. Por lo tanto

$$\begin{aligned} ||B_k - \frac{1}{\delta_c(k)} \sum_{k \to j \text{ correctos}} V_j||^2 &\leq \frac{1}{\delta_c(k)} \sum_{i \to j \text{ correctos}} ||V_i - V_j||^2 + \frac{\delta_b(i)}{\delta_c(k)} ||V_i - V_{\zeta(i)}||^2 \\ E[||B_k - \frac{1}{\delta_c(k)} \sum_{k \to j \text{ correctos}} V_j||^2] &\leq \frac{\delta_c(i)}{\delta_c(k)} 2d\sigma^2 + \frac{\delta_b(i)}{\delta_c(k)} \sum_{i \neq j \text{ correctos}} E[||V_i - V_j||^2] \mathbb{1}(\zeta(i) = j) \\ &\leq \left(\frac{\delta_c(i)}{\delta_c(k)} + \frac{\delta_b(i)}{\delta_c(k)} (n - f - 1) \right) 2d\sigma^2 \\ &\leq \left(\frac{n - f - 2}{n - 2f - 2} + \frac{f}{n - 2f - 2} \cdot (n - f - 1) \right) 2d\sigma^2. \end{aligned} \tag{96}$$

Juntando todo obtenemos

$$\begin{aligned} ||E[KR - g]||^2 &\leq (n - f) 2d\sigma^2 + f \cdot \left(\frac{\delta_c(i)}{\delta_c(k)} + \frac{\delta_b(i)}{\delta_c(k)} (n - f - 1) \right) 2d\sigma^2 \\ &\leq \left(\frac{n - f - 2}{n - 2f - 2} + \frac{f}{n - 2f - 2} \cdot (n - f - 1) \right) 2d\sigma^2 \\ &\leq \underbrace{2 \left(n - f + \frac{f \cdot (n - f - 2) + f^2 \cdot (n - f - 1)}{n - 2f - 2} \right)}_{\eta^2(n,f)} d\sigma^2. \end{aligned} \tag{97}$$

Dado que por hipótesis, $\eta(n,f)\sqrt{d}\sigma < ||g||$, entonces $E[KR]$ pertenece a la bola centrada en g y con radio $\eta(n,f)\sqrt{d}\sigma$. Esto implica que

$$\langle E[KR], g\rangle \geq (||g|| - \eta(n,f)\sqrt{d}\sigma)||g|| = (1-\sin\alpha)||g||^2 \tag{98}$$

y por tanto se cumple la primera propiedad para que KR sea una regla (α, f)-resistente bizantina. Ahora nos centraremos en la segunda condición.

$$\begin{aligned} E[||KR||^r] &= \sum_{i \text{ correctos}} E[||V_i||^r]\mathbb{1}(i_* = i) + \sum_{k \text{ bizantinos}} E[||B_k||^r]\mathbb{1}(i_* = k) \\ &\leq (n-f)E[||G||^r] + \sum_{k \text{ bizantinos}} E[||B_k||^r]\mathbb{1}(i_* = k) \end{aligned} \tag{99}$$

Denotando por C a una constante genérica, cuando $i_* = k$, entonces tenemos para todos los índices correctos i

$$\begin{aligned} ||B_k - \frac{1}{\delta_c(k)} \sum_{k\to j \text{ correctos}} V_j|| &\leq \sqrt{\frac{1}{\delta_c(k)} \sum_{i\to j \text{ correctos}} ||V_i - V_j||^2 + \frac{\delta_b(i)}{\delta_c(k)}||V_i - V_{\zeta(i)}||^2} \\ &\leq C \cdot \left(\sqrt{\frac{1}{\delta_c(k)}} \sum_{i\to j \text{ correctos}} ||V_i - V_j|| + \sqrt{\frac{\delta_b(i)}{\delta_c(k)}}||V_i - V_{\zeta(i)}|| \right) \\ &\leq C \cdot \sum_{j \text{ correctos}} ||V_j|| \quad \text{(desigualdad triangular).} \end{aligned} \tag{100}$$

Donde la segunda desigualdad proviene del hecho de que las normas en dimensión finita son equivalentes. Ahora

$$\begin{aligned} ||B_k|| &\leq ||B_k - \frac{1}{\delta_c(k)} \sum_{k\to i \text{ correctos}} V_j|| + ||\frac{1}{\delta_c(k)} \sum_{k\to i \text{ correctos}} V_j|| \\ &\leq C \cdot \sum_{j \text{ correctos}} ||V_j|| \\ ||B_k||^r &\leq C \cdot \sum_{r_1+\ldots+r_{n-f}=r} ||V_1||^{r_1} \ldots ||V_{n-f}||^{r_{n-f}} \end{aligned} \tag{101}$$

donde por la independencia de los V_i, finalmente obtenemos que $E[||KR||^r]$ está acotado superiormente por una combinación lineal de la forma $E[||V_1||^{r_1}] \ldots E[||V_{n-f}||^{r_{n-f}}] = E[||G||^{r_1}] \ldots E[||G||^{r_{n-f}}]$ con $r_1 + \ldots + r_{n-f} = r$ terminando así la prueba. □

8

TEORÍA DEL APRENDIZAJE

En este capítulo introduciremos los conceptos básicos del AA para introducir posteriormente el concepto de *Deep Learning*. Hablaremos del concepto de aprendizaje y de los diferentes tipos de problemas a los que se enfrenta el AA

8.1 CONCEPTO DE APRENDIZAJE AUTOMÁTICO

Un algoritmo de AA es un algoritmo capaz de aprender información a partir de un cierto conjunto de datos. Para saber de qué hablamos cuando hablamos de aprender podemos referirnos a [22], "Un programa de ordenador se dice que aprende de una experiencia E con respecto a un conjunto de tareas T y medida de rendimiento P, si su rendimiento como tarea en T, medido con P, mejora tras conocer la experiencia E". Desde el punto de vista de las tareas T, el AA es interesante pues nos permite resolver una gran cantidad de tareas que son muy difíciles o incluso imposibles de resolver con programas escritos y diseñados por humanos, ya sea por el complejidad o por el tamaño de la tarea.

Otra definición interesante del concepto de aprendizaje es el de [23]. Para ello primero mencionamos los componentes del problema. Hay una entrada x, una función objetivo $f : \mathcal{X} \to \mathcal{Y}$, donde $\mathcal{X}$ es el espacio de todas las posibles entradas y $\mathcal{Y}$ es el espacio de los posibles resultados. Además, contamos con un conjunto de muestras $\mathcal{D} \subset \mathcal{X} \times \mathcal{Y}$ donde $y = f(x)$ si $(x, y) \in \mathcal{D}$. Finalmente, existe un algoritmo que usa el conjunto $\mathcal{D}$ para escoger una fórmula $g : \mathcal{X} \to \mathcal{Y}$ que aproxime f. Este algoritmo escogerá g dentro de un conjunto de funciones $\mathcal{H}$.

Si consideramos los múltiples componentes del problema, la función objetivo f y las muestras de entrenamiento $\mathcal{D}$ están dictadas por el problema. Sin embargo, el algoritmo de aprendizaje y el conjunto de funciones en el que buscamos $\mathcal{H}$ no lo están, y son por lo tanto somos nosotros quienes las fijamos. Al conjunto $\mathcal{H}$ y al algoritmo se le refiere informalmente como el modelo de aprendizaje.

El AA ha sido aplicado a una gran variedad de situaciones. En la literatura se pueden encontrar ejemplos:

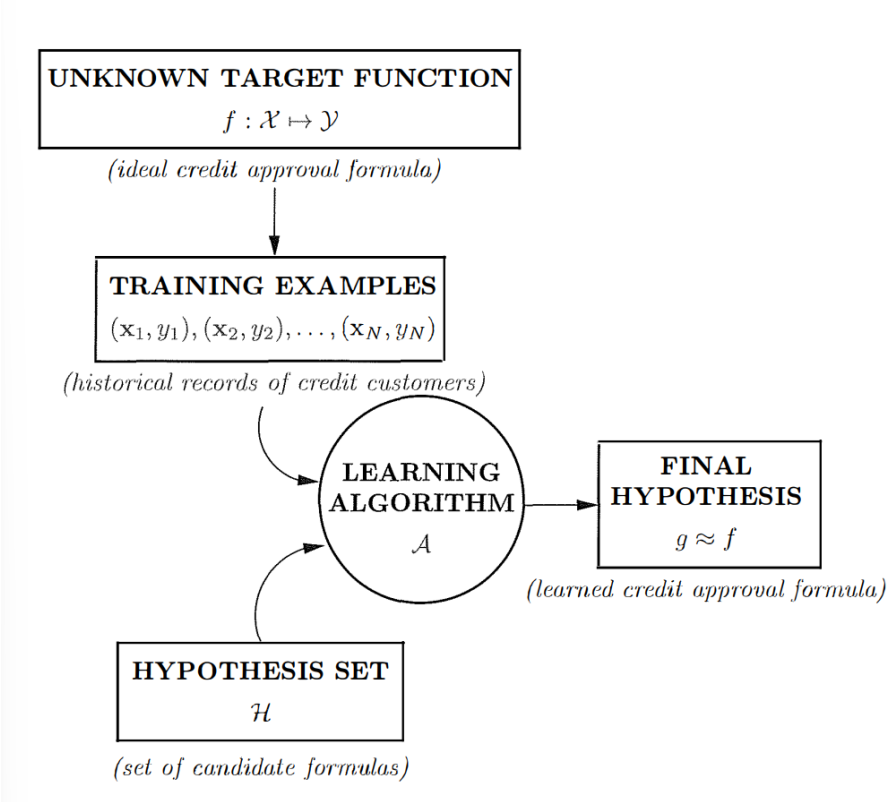


Figura 5: Escenario básico de un problema de aprendizaje. Fuente: [23].

- Clasificación.
- Clasificación con valores perdidos.
- Regresión.
- Transcripción.
- Detección de anomalías.
- Predicción de valores perdidos.
- Estimación de densidad.

Con respecto a la medida de rendimiento P, esta debe de ser cuantitativa. La medida por excelencia en casos de clasificación es la precisión o *accuracy*, definida como la proporción de muestras para las que el modelo ha producido una salida correcta. También en muchas ocasiones es útil la tasa de error, que mide la proporción de muestras para las que el modelo ha producido una salida incorrecta.

8.2 TIPOS DE APRENDIZAJE AUTOMÁTICO

En función de la experiencia E, podemos clasificar los algoritmos de AA en dos grandes grupos: algoritmos supervisados y no supervisados.

- **Aprendizaje Automático No Supervisado**: cuando los datos con los que trabajamos están formados por un conjunto de datos que representa diferentes características de cada una de las muestras. Normalmente, en este caso el objetivo suele ser encontrar alguna propiedad importante para la estructura del conjunto. Un ejemplo clásico es el *clustering* que consiste en agrupar los datos en subconjuntos que contengan características similares.
- **Aprendizaje Automático Supervisado**: cuando se tiene información de un conjunto con características y donde cada muestra tiene asociada una etiqueta. Así, el objetivo de estos problemas será encontrar la relación entre las características de una muestra y su etiqueta para poder etiquetar características futuras. Un ejemplo es el problema de clasificación.
- **Aprendizaje por Refuerzo**: cuando el conjunto de datos no contiene de manera explícita la salida correcta para cada entrada, sino que contiene una posible salida junto a una medida de como de adecuada esta salida es. Este tipo de aprendizaje es ideal para situaciones en las que no existe necesariamente un resultado correcto, como puede ser el problema de la conducción autónoma de vehículos.

8.3 PROBLEMA DE CLASIFICACIÓN

El problema de clasificación es el ejemplo más frecuente de aprendizaje supervisado. Consiste en, a partir de una población de entrada, clasificarla en diferentes subpoblaciones o clases. Se debe de conocer previamente el número de clases posibles, por lo que el problema consiste en decidir para cada elemento qué clase es la que mejor se ajusta a sus características observadas. Según el número de clases consideradas se puede distinguir entre:

- **Clasificación Binaria**: cuando la clasificación se reduce a tan solo dos etiquetas. Se puede ver como un problema de verdadero o falso, o pertenencia o no a un subgrupo de la población.
- **Clasificación Multiclase**: cuando el conjunto de etiquetas es mayor que dos. En este caso se debe elegir entre todas ellas cuál es la más adecuada.

Nosotros nos centraremos en los problemas de clasificación multiclase.

9

FUNDAMENTOS DEL DEEP LEARNING

En este capítulo nos centraremos propiamente en el *Deep Learning*. Para ello estudiaremos las redes neuronales prealimentadas, cómo funcionan y estudiaremos el aprendizaje basado en gradiente. Posteriormente hablaremos de redes convolucionales y concretamente de la arquitectura `EfficientNet`.

9.1 REDES NEURONALES PREALIMENTADAS

Las redes neuronales prealimentadas o Perceptrón Multicapa (*Multilayer Perceptron*) (MLP) son el modelo por excelencia del *deep learning* [10]. El objetivo de un MLP es el de aproximar una función f^*. Por ejemplo, para un clasificador, $y = f^*(x)$ lleva una entrada x a una categoría y. Un MLP define una correspondencia $y = f(x;\theta)$ y aprende el parámetro θ que lleva a la mejor aproximación de la función.

Este tipo de modelos se llaman prealimentados porque la información fluye a través de la función siendo evaluados desde x, a través de las computaciones intermedias usadas para definir f, y finalmente hasta el resultado y. Las redes prealimentadas son de vital importancia en el AA, ya que son el fundamento de muchas aplicaciones comerciales. Por ejemplo, las redes convolucionales usadas para reconocer objetos en fotografías son un tipo especializado de redes neuronales prealimentadas.

Las redes neuronales prealimentadas son llamadas redes porque son normalmente representadas como la composición de muchas funciones distintas. Se asocia el modelo con un grafo acíclico dirigido describiendo cómo las funciones se componen entre sí. Por ejemplo, podemos tener tres funciones $f^{(1)}$, $f^{(2)}$ y $f^{(3)}$ conectadas de forma secuencial, para obtener $f(x) = f^{(3)}(f^{(2)}(f^{(1)}(x)))$. Esta estructura secuencial es la estructura más común usada en redes neuronales. En este caso, $f^{(1)}$ es conocido como la primera capa de la red, $f^{(2)}$ la segunda capa y así. La longitud total se conoce como la profundidad del modelo. El nombre *deep learning* nace de esta terminología. La última capa se denomina capa de salida. Durante el entrenamiento de la red, intentamos que $f(x)$ se asemeje lo máximo posible a $f^*(x)$.

Por último, estas redes se llaman neuronales porque están ligeramente inspiradas en la neurociencia. La dimensión de cada capa oculta determina la anchura del modelo. Podemos ver cada capa como un conjunto de unidades que actúan en paralelo, cada una representando una función vector a escalar. Cada unidad recuerda a una neurona ya que recibe una entrada de otras unidades y devuelve su valor de activación.

Otra manera de entender las redes prealimentadas es comenzar con los modelos lineales y pensar en cómo solventar sus limitaciones. Los modelos lineales, tales como la regresión logística o lineal, son atractivos porque ajustarse eficientemente, ya sea de manera cerrada o mediante optimización convexa. Los modelos lineales también tienen el obvio problema de estar limitados a funciones lineales. Por lo tanto el modelo no puede entender la interacción entre dos variables cualesquiera de entrada.

Para extender los modelos lineales con el objetivo de representar funciones no lineales de x, en lugar de aplicar el modelo lineal a x podemos aplicárselo a una transformación de la entrada $\phi(x)$, donde ϕ es una transformación no lineal. Nos podemos plantear en que transformación ϕ aplicar a x.

1. Una opción es usar una ϕ muy genérica de dimensión muy alta, de manera que siempre tengamos capacidad suficiente para ajustar el conjunto de entrenamiento, pero la generalización al conjunto de evaluación suele ser bastante pobre.
2. Otra opción es crear una ϕ de manera manual. Antiguamente este solía ser el método más común. Requiere una gran cantidad de trabajo humano para cada tarea y no son reutilizables.
3. La estrategia que sigue el *deep learning* es en aprender ϕ. De esta manera, tenemos un modelo $y = f(x; \theta, w) = \phi(x; \theta)^T w$. Ahora tenemos un conjunto de parámetros θ que usamos para aprender ϕ dentro de una clase amplia de funciones, y parámetros w que llevan $\phi(x)$ a la salida deseada. Este es un ejemplo de red neuronal prealimentada, con ϕ definiendo una capa oculta. Esta técnica es la única de las tres que decide dejar de lado la convexidad del problema.

Este principio de mejorar modelos mediante el aprendizaje de características va mucho más allá de las redes neuronales prealimentadas y es un tema recurrente que se aplica a todo tipo de modelos en el *deep learning*.

9.2 APRENDIZAJE BASADO EN GRADIENTE

La principal diferencia entre los modelos lineales que hemos visto hasta el momento y las redes neuronales es que la no linealidad de estas últimas hace que la mayoría de funciones de pérdida interesantes se vuelven no convexas. Es por esto que las redes neuronales normalmente se entrenan usando optimizadores iterativos basados en gradientes que simplemente llevan la función de coste o pérdida a un valor muy bajo. En la optimización convexa tenemos convergencia desde cualquier conjunto

de parámetros iniciales, pero esto es algo que perdemos al usar métodos iterativos basados en gradiente.

9.2.1 *Funciones de Coste*

Es por esto, que un aspecto importante del diseño de una red neuronal profunda es la elección de la función de coste o pérdida. En la mayoría de casos, nuestro modelo paramétrico define una distribución $p(y|x;\theta)$ y por lo tanto aplicamos el principio de máxima verosimilitud. Esto significa usar la entropía cruzada entre los datos de entrenamiento y las predicciones del modelo como función de pérdida.

En otras ocasiones, en lugar de predecir una distribución de probabilidad completa sobre y, simplemente predecimos algún tipo de estadístico de y condicionado a x.

El función de coste total usada para entrenar una red neuronal normalmente combina una función principal con un término de regularización.

9.2.2 *Unidades de Salida*

La elección de la función de coste está íntimamente relacionada con la elección de la unidad de salida. La mayoría de ocasiones, simplemente calculamos la entropía cruzada entre la distribución de los datos y la del modelo. La elección de como representar la salida determina la forma de la función de entropía cruzada.

Cualquier tipo de unidad de red neuronal que puede ser usada como salida también puede ser usada como unidad oculta. Aquí, nos centraremos en el uso de estas unidades como salida del modelo, pero pueden ser usadas de forma interna también en un principio.

UNIDADES LINEALES

Un tipo sencillo de unidad de salida está basado en una transformación afín sin no linealidad. Normalmente se llaman unidades lineales.

Dadas características h, una capa lineal produce un vector $\hat{y} = W^T h + b$. Las capas de salida lineales normalmente se usan para producir la media de una distribución Gaussiana condicionada:

$$p(y|x) = \mathcal{N}(y;\hat{y}, I) \tag{102}$$

Maximizar el logaritmo de la verosimilitud es entonces equivalente a minimizar el error cuadrático medio.

UNIDADES SIGMOIDALES

Muchas tareas requieren predecir el valor de una variable binaria y. Es decir, un problema de clasificación de dos clases. Podemos intentar definir una distribución de Bernoulli sobre y condicionada a x. Como una distribución de Bernouilli está definida por tan solo un número, la red neuronal necesita predecir $P(y = 1|x)$. Para que este número sea una probabilidad válida, debe de estar en el intervalo $[0, 1]$.

Para esto usamos una función que siempre tiene un gradiente fuerte cuando el modelo se equivoca. Esta función es la función sigmoide definida como

$$\hat{y} = \sigma(w^T h + b) = \frac{1}{1 + e^{-(w^T h + b)}} \tag{103}$$

Podemos pensar en que la unidad de salida sigmoide tiene dos componentes. Primero, usa capa lineal y luego usa la función de activación sigmoide σ para convertir la salida anterior en una probabilidad.

Vamos a omitir la dependencia sobre x por un momento para discutir como definir una distribución de probabilidad sobre y usando la salida de la capa lineal z. La función sigmoide puede ser motivada mediante la construcción de una distribución de probabilidad $\tilde{P}(y)$ no normalizada, es decir, que no suma 1. Por lo tanto podemos dividir por una constante apropiada para obtener una distribución de probabilidad válida. Si comenzamos asumiendo que el logaritmo de las probabilidad no normalizadas son lineales en y y z, podemos usar la exponencial con el fin de obtener las probabilidad no normalizadas. Entonces normalizamos para ver que obtenemos una distribución de Bernoulli controlada por una transformación sigmoilad de z:

$$\log \tilde{P}(y) = yz, \tag{104}$$

$$\tilde{P}(y) = \exp(yz), \tag{105}$$

$$P(y) = \frac{\exp(yz)}{\sum_{y'=1}^{1} \exp(y'z)}, \tag{106}$$

$$P(y) = \sigma((2y - 1)z) \tag{107}$$

La variable z que define estas distribuciones sobre variables binarias se llama *logit*.

UNIDADES SOFTMAX

Cualquier momento que queramos representar una distribución de probabilidad sobre una variable discreta con n valores posibles, usaremos la función *softmax*. Puede ser vista como una generalización de la función sigmoide, que fue usada para representar una distribución de probabilidad sobre una variable binaria.

Para generalizar al caso de una variable discreta con n valores, necesitamos producir un vector $\hat{y}$, con $\hat{y}_i = P(y = i|x)$. No solo necesitamos que cada elementos del vector

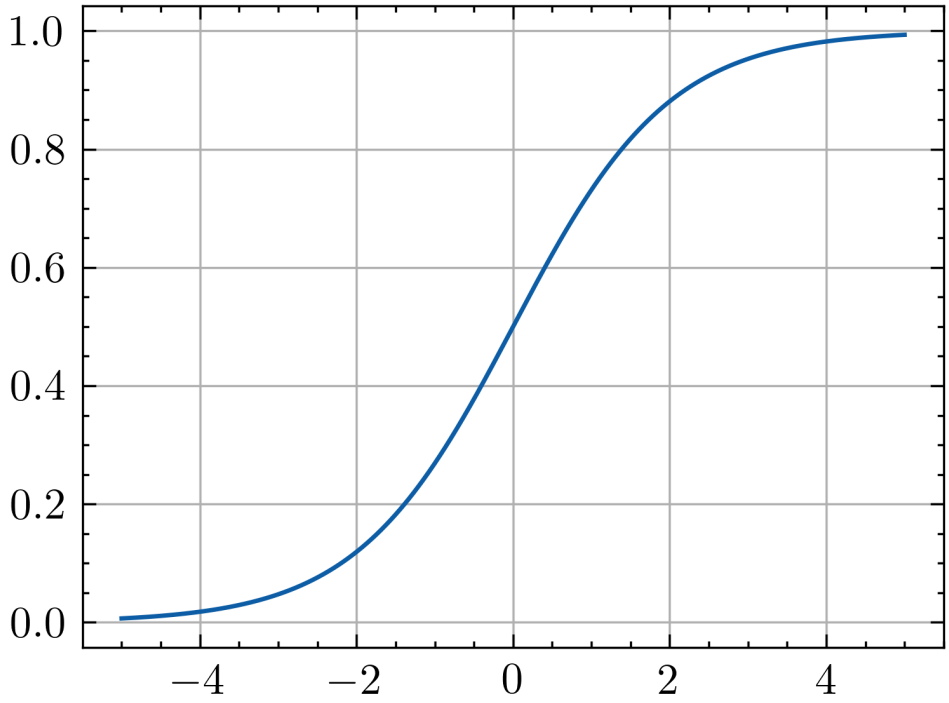


Figura 6: Gráfica de la función sigmoide. Elaboración propia.

esté entre 0 y 1, sino que además la suma de todo el vector sea igual a 1. Repitiendo el procedimiento usado para las sigmoides podemos generalizar la distribución. Primero, una capa lineal predice probabilidades logarítmicas no normalizadas:

$$z = W^T h + b \tag{108}$$

donde $z_i = \log \tilde{P}(y = i|x)$. Entonces, la función softmax puede exponenciar y normalizar z para obtener $\hat{y}$. Formalmente, la función softmax está dada por:

$$softmax(z)_i = \frac{\exp(z_i)}{\sum_j \exp(z_j)} \tag{109}$$

Al igual que en el caso de la función logística, el uso de la exponencial funciona bien cuando entrenamos el softmax para predecir el valor y usando máxima verosimilitud.

9.2.3 *Unidades ocultas*

Como hemos mencionado anteriormente, cualquier unidad de salida que hemos explicado también puede ser aplicada como unidad oculta. Aún así, existen otras funciones de activación para este caso. Cabe destacar que la investigación sobre el diseño de las unidades ocultas sigue activo. Nosotros nos centraremos en algunas de las más comunes.

UNIDADES LINEALES RECTIFICADAS (RELU) Y SUS GENERALIZACIONES

Las unidades lineales rectificadas usan la función de activación $g(z) = \text{máx}\{0, z\}$. Son fáciles de optimizar porque son muy parecidas a las unidades lineales. La única diferencia es que estas devuelven cero en la mitad de su dominio. Esto permite que las derivadas mediante una unidad linear rectificada se mantengan grandes siempre que la unidad está activa. Los gradientes además son consistentes. Además, la segunda derivada de una ReLU es 0 casi en todo punto, y la derivada primera es 1 donde la unidad está activa.

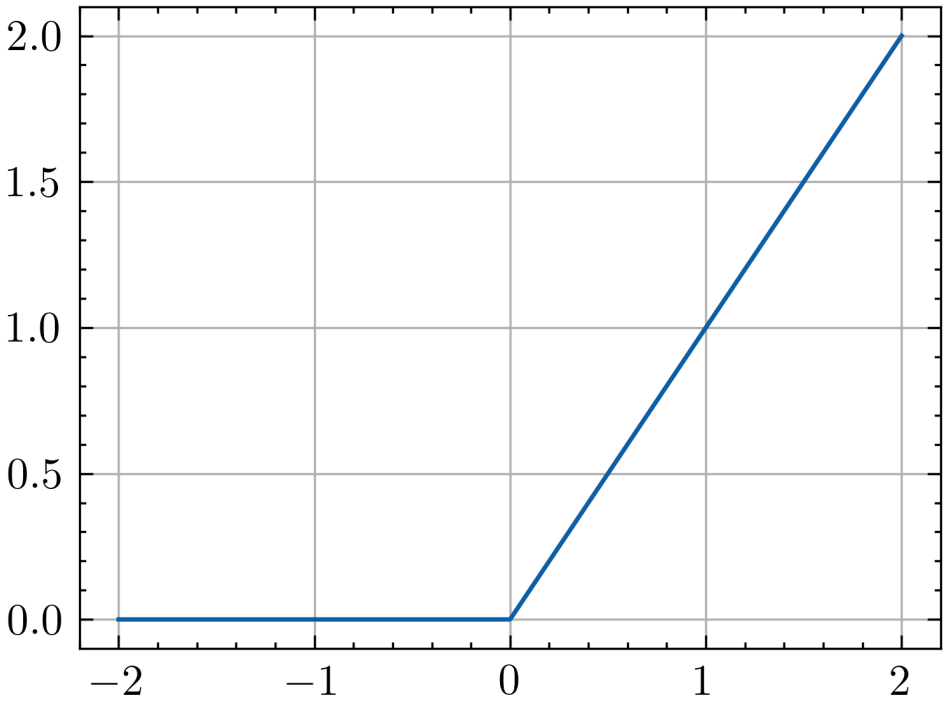


Figura 7: Gráfica de la función ReLU. Elaboración propia

Existen bastantes generalizaciones de las ReLU. La mayoría de estas generalizaciones funcionan de manera comparable o incluso mejor que las primeras. Muchas generalizaciones se centran en solventar el problema de que no pueden aprender por descenso por el gradiente cuando su activación es cero.

- **Rectificación de valor absoluto** consiste en fijar una pendiente de -1 cuando la salida de la ReLU es menor que 0, haciendo así $g(z) = |z|$.
- **Leaky ReLU** fija una pendiente muy pequeña como 0.01 cuando la salida de la ReLU es 0.
- **ReLU paramétrica** trata la pendiente ya mencionada como un valor a aprender.

SIGMOIDE LOGÍSTICA Y TANGENTE HIPERBÓLICA

Antes de que se introdujesen las ReLU, la mayoría de redes neuronales usaban la función logística sigmoide como función de activación o la tangente hiperbólica.

$$g(z) = \frac{1}{1 + e^{-z}} \tag{110}$$

$$g(z) = \tanh(z) \tag{111}$$

Estas funciones están estrechamente relacionadas pues $\tanh(z) = 2\sigma(2z) - 1$.

Ya hemos visto las unidades sigmoide como unidades de salida, usadas para predecir la probabilidad de que una variable binaria tome el valor 1. Sin embargo, las unidades sigmoidales se saturan en la mayor parte de su dominio, y solo son sensibles cuando su entrada es muy cercana a 0. La amplia saturación de las unidades sigmoidales hace que el aprendizaje por el gradiente sea muy difícil y es por ello que no se recomienda su uso como unidades ocultas. Solo se recomiendan cuando tengamos una función de coste capaz de deshacer la saturación de esta.

Las funciones de activación sigmoide son más comunes en otros escenarios que en MLP. Las redes recurrentes, muchos modelos probabilísticos, y algunos *autoencoders* tienen requisitos adicionales que hacen el uso de funciones sigmoidales más atractivos pese a la desventaja de la saturación.

9.3 REDES CONVOLUCIONALES

Una Red Neuronal Convolucional (Convolutional Neural Network) (CNN) [10] es un tipo especializado de red neuronal para procesar datos que tenga una topología de rejilla. Esto incluye datos en series temporales, que pueden ser pensados como una rejilla 1 dimensional a intervalos de tiempo regulares, e imágenes, que pueden ser pensadas como una rejilla 2 dimensional de píxeles. Las CNN han demostrado ser realmente exitosas en múltiples aplicaciones prácticas. El nombre de "red neuronal convolucional" indica que la red hace uso de la operación de convolución.

Vamos a explicar que es la convolución, cuál es la motivación para usarla en una red neuronal, descubriremos la técnica de *pooling* que es usada a día de hoy en la mayoría de las redes convolucionales y nombraremos algunas redes importantes de este tipo.

9.3.1 *La Operación de Convolución*

En su forma más general, la convolución es una operación que toma como argumento dos funciones de variable real. Sean $f, g : \mathbb{R} \to R$ dos funciones de variable real,

la convolución de ambas funciones se denota por $(f \star g)$ a su convolución y viene definida por:

$$s(t) = (f \star g) = \int_{-\infty}^{\infty} f(s)g(t-s)ds \tag{112}$$

Si usamos la terminología de las CNN, nuestro primer argumento, es decir f se le conoce como la entrada, y al segundo, g, será el núcleo o *kernel* de la convolución. Al resultado en ocasiones se le llama mapa de características o *feature map*.

Normalmente, cuando trabajamos con datos en un ordenador, no podemos tratar con un continuo sino con datos discretizados. Por lo tanto asumir que podemos evaluar las funciones en todo punto no es realista. Es por ello que definimos la operación de convolución discreta como será natural viendo la definición anterior:

$$s(t) = (f \star g) = \sum_{s=-\infty}^{\infty} f(s)g(t-s) \tag{113}$$

En el AA, las entradas normalmente suelen ser tensores de datos, y el *kernel* suele ser también un *kernel* de parámetros que son adaptados por el algoritmo de aprendizaje que usemos. Debido a que cada elemento de la entrada y del *kernel* debe de ser almacenado por separado, asumimos que estas funciones son cero en todos los puntos salvo en el conjunto finito del que almacenamos valores. En práctica, esto significa que podemos implementar la suma infinita como una suma finita de elementos de los tensores.

Además, también usaremos convoluciones sobre más de un eje a la vez. Por ejemplo, si usamos una imagen de dos dimensiones I como entrada, posiblemente también consideremos usar un *kernel* de dos dimensiones K:

$$S(i,j) = (I \star K)(i,j) = \sum_m \sum_n I(m,n)K(i-m, j-n) \tag{114}$$

Además, la convolución también es conmutativa por lo que podemos escribir también:

$$S(i,j) = (K \star I)(i,j) = \sum_m \sum_n K(m,n)I(i-m, j-n) \tag{115}$$

Normalmente se implementa la última fórmula pues hay menos variación en el rango de valores válidos de m y n.

9.3.2 *Motivación*

La convolución implica tres ideas importantes que pueden ayudar a mejorar un sistema de AA: compartir parámetros, interacciones dispersas y representaciones equivariantes. Además, también nos permite un método para trabajar con entradas de tamaño variable.

Las capas tradicionales de redes neuronales usan la multiplicación por una matriz de parámetros con un parámetro parámetro separado que describe la interacción entre cada unidad de entrada y cada unidad de salida. Esto significa que cada unidad de salida interacciona con cada unidad de entrada. Sin embargo, una CNN suele tener interacciones dispersas. Esto es cuando el *kernel* es más pequeño que la entrada (que suele ser el caso habitual). Por ejemplo, cuando procesamos una imagen, la imagen puede tener miles o millones de píxeles, pero podemos detectar características pequeñas pero útiles tales como bordes con *kernels* que son del tamaño de cien píxeles o menos. Esto significa que necesitamos almacenar menos parámetros, lo que reduce requisitos de memoria y mejora la eficiencia estadística del modelo. También implica que calcular el resultado del modelo necesita de menos operaciones. Estas mejoras en eficiencia suelen ser bastantes grandes. Si hay m entradas y n salidas, entonces la multiplicación de matrices necesita de $m \times n$ parámetros y los algoritmos tendrán una complejidad de $O(m \times n)$. Si limitamos el número de conexiones con cada salida a k, entonces solo necesitaremos $k \times n$ parámetros. Para muchas aplicaciones prácticas, es posible obtener un buen rendimiento en la tarea de AA mientras mantenemos k órdenes de magnitud por debajo de m.

Compartir parámetros se refiere al uso de un mismo parámetro para más de una función en un modelo. En las arquitecturas tradicionales de redes neuronales, cada elemento de la matriz de pesos se usa exactamente una vez cuando se calcula la salida de una capa. En una CNN, cada elemento del *kernel* se usa en cada posición de la entrada (excepto quizás en el borde de esta). Que se compartan los parámetros significa que en lugar de aprender un parámetro para cada ubicación de la entrada se aprende únicamente un conjunto de ellos. Esto no afecta al tiempo de ejecución de un paso de la red, pero si disminuye los requisitos de memoria del modelo. Por lo tanto, las convoluciones son dramáticamente más eficaces que las matrices densas en términos de memoria y eficiencia estadística.

En el caso de la convolución, el hecho de compartir parámetros implica que la capa tiene una propiedad llamada equivarianza a translaciones. Decir que una función es equivariante significa que si la entrada cambia, la salida se mantiene igual. Formalmente, una función $f(x)$ es equivariante a una función g si $f(g(x)) = g(f(x))$. En el caso de la convolución, si g es cualquier translación, esto es, desplaza la entrada, entonces la convolución es invariante a g. En el caso de imágenes, la convolución crea un mapa 2-D de dónde aparecen ciertas características en la entrada. Por ejemplo, es útil detectar bordes en la primera capa de una CNN. Los bordes aparecen más o menos en todas las partes de la imagen, así que tiene sentido compartir los parámetros para toda la imagen. En algunos casos, si estamos procesando imágenes recortadas para estar centradas respecto a algo, como por ejemplo, la cara de una persona, tal vez no nos interese compartir parámetros.

9.3.3 *Pooling*

Una capa de una *CNN* suele estar formada por tres etapas. En la primera etapa, la capa realiza convoluciones en paralelo para producir un conjunto de activaciones lineales. En la segunda etapa, cada activación lineal se ejecuta con una función de activación no lineal, tales como ReLU. En la tercera etapa, usamos una función de *pooling* para modificar la salida de la capa.

Una función de *pooling* reemplaza la salida de la red en una cierta localización con un estadístico de las salidas cercanas. Por ejemplo, el *max pooling* consiste en calcular la salida máxima dentro de un entorno rectangular. Otras funciones de *pooling* populares son calcular el promedio en un entorno rectangular, la norma L^2 de un entorno rectangular o la media ponderada basada en la distancia del píxel central.

En todos los casos, el *pooling* nos ayuda a hacer una representación aproximadamente invariante a pequeñas translaciones de la entrada. La invariancia a translaciones locales puede ser una propiedad útil si nos interesa más la presencia de algunas características que el saber exactamente dónde están. Por ejemplo, para determinar si una imagen contiene un rostro, no necesitamos saber la posición de los ojos con una precisión perfecta, solo saber que hay un ojo izquierdo a la izquierda de la cara y un ojo derecho a la derecha de la cara.

Para muchas tareas, el *pooling* es esencial para manejar entradas de tamaño variable. Por ejemplo, si queremos clasificar imágenes de un tamaño variable, la entrada a la capa de clasificación debe de tener un tamaño fijo. Esto normalmente se logra adaptando el tamaño de las regiones de *pooling* de manera que la capa de clasificación siempre reciba el mismo número de características sin importar el tamaño de la entrada. Por ejemplo, la última capa de *pooling* en una red puede estar definida para dar siempre cuatro características, una por cada cuadrante de la imagen, sin importar el tamaño de esta.

9.4 EFFICIENTNET

Durante el desarrollo de este trabajo haremos uso de la arquitectura de *EfficientNet* [33]. Esta arquitectura es un tipo de red convolucional centrada en escalar apropiadamente la profundidad, anchura y resolución de la red con el fin de obtener un mejor rendimiento.

9.4.1 *Escalado del modelo*

El problema que intenta resolver *EfficientNet* es encontrar un buen factor de escalado entre la anchura, profundidad y resolución de la red. Para ello, primero se debe de formular el problema de forma teórica.

La capa i de una red convolucional se puede definir como una función $Y_i = \mathcal{F}_i(X_i)$, donde $\mathcal{F}_i$ es el operador, Y_i es el tensor de salida y X_i es el tensor de entrada con forma (H_i, W_i, C_i), donde H_i y W_i son las dimensiones espaciales y C_i es la dimensión de los canales de la imagen. Una red convolucional se puede representar como una lista de capas compuestas: $\mathcal{N} = \mathcal{F}_k \circ \ldots \circ \mathcal{F}_2 \circ \mathcal{F}_1(X_1)$. En la práctica, muchas redes convoluciones se dividen en etapas y todas las capas en la misma etapa comparten la misma arquitectura. Por lo tanto se puede definir una red convolucional como:

$$\mathcal{N} = \circ_{i=1,\ldots,s} F_i^{L^i}(X_{(H_i,W_i,C_i)}) \tag{116}$$

donde $F_i^{L_i}$ denota que la capa F_i se repite L_i veces en la etapa i.

El objetivo de *EfficientNet* [33] consiste en expandir la longitud (L_i), anchura (C_i) y/o resolución (H_i, W_i) sin cambiar $\mathcal{F}_i$, la cual ya está previamente definida por una arquitectura base. Esto simplifica el problema de diseño pero sigue dejando un amplio espacio para explorar distintos parámetros para cada capa. Con el fin de reducir este espacio, los autores deciden restringir a que todas las capas deben de ser escaladas de manera uniforme con un ratio constante.

Los autores observan de manera empírica que las diferentes dimensiones en las que escalar no son independientes. Intuitivamente, para imágenes de mayor resolución, se debe de aumentar la profundidad de la red, con el fin de que campos receptores más grandes puedan ayudar a capturar características similares que incluyen más píxeles en imágenes más grandes. A su vez, también se debería de aumentar la anchura de la red cuando la resolución es mayor, con el objetivo de capturar patrones más finos con más píxeles. Por lo tanto se necesita coordinar y equilibrar las dimensiones sobre las que escalamos en lugar de escalar una única dimensión.

9.4.2 *La Arquitectura de EfficientNet*

Una vez dada la técnica de escalado para una red base, queda clara la importancia de esta última. Para obtener una arquitectura nueva, los autores han usado una búsqueda multi-objetivo de redes neuronales que optimiza tanto los *FLOPS* como la precisión. Concretamente usando como función a optimizar $ACC(m) \times [FLOPS(m)/T]^{\omega}$ donde $ACC(m)$ es la precisión del modelo m, $FLOPS(m)$ los *FLOPS* del modelo, T el objetivo de *FLOPS* a conseguir y $\omega = -0{,}007$ un hiperparámetro para compensar entre la precisión y los $FLOPS$. El resultado de esta búsqueda es un modelo llamado

`EfficientNet-B0`. Aplicando posteriormente la técnica de escalado logran crear los modelos `EfficientNet-B1` hasta `EfficientNet-B7` siendo este último un modelo estado del arte en el momento de publicación de [33], siendo 8.4 veces más pequeña y 6.1 veces más rápida que la mejor red convolucional existente previamente.

Parte III

PRELIMINARES EN APRENDIZAJE FEDERADO Y BLOCKCHAIN

10

FEDERATED LEARNING

10.1 INTRODUCCIÓN

El *deep learning* ha revolucionado el campo de la IA mediante la posibilidad de permitir a las máquinas a aprender y tomar decisiones como los humanos gracias a técnicas basadas en datos. El de desarrollo en redes de alta velocidad tales como 5G y avances en *edge computing* ha permitido el desarrollo de modelos y hardware capaces de procesar grandes cantidades de datos recolectados de múltiples dispositivos. En consecuencia, la privacidad se ha convertido en una gran influencia en el diseño, llevando los esfuerzos desde el AA centralizado al AA distribuido. Aún así, en el AA distribuido los costes de comunicación son inmensamente mayores que los costes de cómputo, haciendo el proceso de entrenamiento ineficiente. El FL o *Federated Learning* fue inventado para resolver estos problemas [18]. En esencia, FL es un paradigma de aprendizaje distribuido que permite a un modelo aprender a partir de datos distribuidos, sin la necesidad de estos de ser recolectados por un servidor central. Dado que los dados locales nunca abandonan los dispositivos donde son recogidos, se garantiza la privacidad de datos.

FL ha ganado atención debido a su capacidad de resolver preocupaciones sobre la privacidad y de mejorar la eficiencia de aprendizaje distribuido. Además, es altamente escalable ya que puede soportar una gran cantidad de participantes, cada uno con su respectivas fuentes de datos. Esto puede ser realmente útil en escenarios con una generación continua de datos como puede ser el IoT, sensores, etc. Como resultado, FL se ha convertido en un campo importante de la IA obteniendo el interés de investigadores, desarrolladores y científicos de datos en la comunidad del AA. El primer caso de uso exitoso del FL fue desarrollado por Google [21] para predecir la siguiente entrada de texto del usuario en miles de dispositivos Android, mientras se mantenía los datos de los dispositivos en local. Desde entonces, el FL se ha aplicado a un gran rango de aplicaciones en diversos campos, desde ingeniería industrial hasta el mundo de la salud [29].

10.2 FEDERATED LEARNING: ¿QUÉ Y POR QUÉ?

El AA basado en datos domina actualmente el campo de la IA. Desafortunadamente, la creciente demanda en término de volumen de datos y variedad ha resultado en varios problemas relacionado en la privacidad de los datos y en el procesamiento de grandes cantidades de datos. Entre ellos, los principales desafíos del AA que llevan a la aparición del FL están asociados con la privacidad, comunicación y acceso a los datos [18].

- **Privacidad de los Datos**: en el AA centralizado, los datos de los usuarios son recolectados y se almacenan en un servidor central, donde pueden ser vulnerables a una brecha de seguridad. Además, la creciente preocupación sobre la protección de los datos se manifiesta en el área legal. Por lo tanto, hay una urgente demanda en el desarrollo de métodos de IA que protejan la privacidad.
- **Costes de Comunicación y Latencia**: en el AA centralizado, los datos sin procesar son mandados al servidor central para ser procesados y usados para entrenar el modelo. Este intercambio de información puede ser costoso, especialmente cuando se trata con conjuntos de datos muy grandes. Además, la creciente cantidad de datos disponibles debido al drástico aumento de sensores IoT y dispositivos móviles generando enormes cantidades de datos supone un nuevo desafío relacionado con el almacenamiento y preprocesamiento de un flujo continuo de datos.
- **Limitaciones al Acceso de Datos**: en algunos casos, los datos pueden ser distribuidos entre diferentes instituciones u organizaciones, haciendo complicado el acceder o compartir datos entre ellas.

Con el objetivo de resolver los desafíos previamente mencionados, FL aparece como un paradigma de AA distribuido enfocado en desarrollar un modelo de AA sin explícitamente compartir ningún dato entre los participantes. Implica una red de clientes $\{C_1, C_2, \ldots, C_n\}$, que participa en dos fases principales:

1. **Fase de Entrenamiento del Modelo**: cada cliente (o propietario de datos) intercambia información sin revelar ningún dato de su conjunto para entrenar de forma conjunta un modelo de AA. Para lograr esto, cada cliente entrena un modelo local con sus datos y comparte la información de este modelo en lugar de sus datos de entrenamiento. Entonces, los modelos locales son agregados para crear un modelo global.
2. **Fase de Inferencia**: el modelo global es aplicado a nuevas instancias de datos.

Estos procesos pueden ser tanto síncronos como asíncronos dependiendo de la disponibilidad de los nodos y del modelo entrenado. Es importante notar que la privacidad no es la única razón para aplicar este método, ya que también se debería usar alguna estrategia para compartir los beneficios del modelo entrenado de forma colaborativa.

Una vez hemos descrito FL como un concepto general, un escenario de FL puede ser propuesto formalmente de la siguiente manera. Suponemos un conjunto de clientes $\{C_1, \ldots, C_n\}$ con sus respectivos conjuntos de datos de entrenamiento $\{D_1, \ldots, D_n\}$. Cada uno de estos clientes C_i cuenta un con modelo local L_i expresado como los parámetros $\{L_1, \ldots, L_n\}$. El FL se enfoca en aprender un modelo global G, usando datos distribuidos entre clientes mediante un proceso de aprendizaje iterativo conocido como **ronda de aprendizaje**. Para ello, en cada ronda de aprendizaje t, cada cliente entrena su modelo local sobre sus datos locales D_i^t, resultando en la actualización de los parámetros locales L_i^t a $\hat{L}_i^t$. Después, los parámetros globales G^t son calculados agregando los parámetros locales entrenados $\{\hat{L}_1^t, \ldots, \hat{L}_n^t\}$ usando una operación fija de agregación denotada por Δ. Luego los parámetros locales son actualizados con los parámetros globales agregados.

$$G^t = \Delta(\hat{L}_1^t, \ldots, \hat{L}_n^t) \tag{117}$$

$$L_i^{t+1} \leftarrow G^t, \quad \forall i \in \{1, \ldots, n\}. \tag{118}$$

Las actualizaciones entre los clientes y el servidor son repetidas durante el proceso de aprendizaje hasta que se alcanza algún criterio de parada. Así, el valor final de G resumirá el conocimiento modelado en los clientes.

10.3 ELEMENTOS CLAVE EN FEDERATED LEARNING

Una vez que el FL ha sido introducido de manera breve, podemos hablar del flujo de trabajo de un proceso de FL. Este se compone de los siguientes pasos:

1. **Entrenamiento local**: comienza con el entrenamiento local de cada uno de los modelos locales por cada uno de los clientes. Generalmente, todos estos modelos tienen una arquitectura común. Sin embargo, los aspectos relacionados con los hiperparámetros de entrenamiento (*epochs*, *batch size*, *learning rate*) puede diferir entre clientes. En esta fase los primeros elementos aparecen de manera natural:
   - **Datos Descentralizados**: los datos se encuentran distribuidos entre los distintos dispositivos o nodos, en lugar de en un mismo lugar de forma centralizada, lo cual es un beneficio cuando la privacidad es una preocupación. La distribución de los datos puede ser:
     a) **Homogénea o i.i.d.**: se asume que la distribución de datos entre los clientes es i.i.d., por lo que todos los clientes tienen la misma distribución.
     b) **Heterogénea o no-i.i.d.**: se asume que la distribución de datos entre los clientes no es i.i.d., esto es, que los datos de cada cliente sigue una distribución distinta. Formalmente se pueden distinguir entre tres tipos de heterogeneidad de la distribución de datos:

    - El espacio de características de nuestros clientes es distinto pero comparten el mismo objetivo.
    - El espacio de entradas es análogo pero el espacio de las etiquetas es distintos respecto a los mismos datos.
    - Ambos espacios son distintos.
  - **Modelo**: el entrenamiento del modelo es realizado en los datos descentralizados, donde cada dispositivo o nodo entrena su modelo y contribuye al proceso de entrenamiento, compartiendo los pesos de su modelo local. También mejora el modelo debido a una mejor generalización, ya que el modelo puede aprender de un rango más amplio de datos.
  - **Clientes**: estos nodos almacenan datos y entrenan modelos locales.
2. **Comunicación**: después del entrenamiento local, la comunicación permite la coordinación y agregación de las actualizaciones del modelo generadas por los participantes, permitiendo así un entrenamiento distribuido. Es un rol crucial en la protección de la privacidad y seguridad de los datos cuando se combina con varias técnicas como la Privacidad Diferencial (*Differential Privacy*) (DP). Destacamos los siguientes elementos clave en este paso:
  - **Planificación de Comunicaciones**: la comunicación puede ser tanto síncrona como asíncrona, dependiendo de la configuración. También puede haber un servidor central que maneja la recolección de todos los modelos locales, o puede estar distribuido a través de múltiples nodos en la red.
  - **Protocolos de Privacidad**: pese a que no se comparten los datos de entrenamiento durante las comunicaciones en FL, la información compartida es susceptible a brechas de privacidad o a corromper el proceso de aprendizaje en su totalidad. Por lo tanto, las comunicaciones son uno de los puntos débiles de FL respecto a los ataques. Por esta razón es normal combinarlas con otros mecanismos de privacidad.
3. **Agregación**: las actualizaciones locales generadas por cada nodo son combinadas mediante un operador específico de agregación y el resultado es incorporado para actualizar y crear un modelo global entrenado. El elemento clave en este paso es el **mecanismo de agregación**, que depende en la tarea que se plantea resolver. Pese a ello, el mas común es ***Federated Averaging* (FedAvg)** cuando el modelo se puede expresar como un vector de pesos. En otro caso, como podría ser el *clustering*, se debe de diseñar un agregador específico para combinar la información de cada nodo.
4. **Actualización local**: el último paso consiste en actualizar los modelos locales almacenados en los distintos nodos con el nuevo modelo global. El caso más sencillo es actualizar todos los modelos locales con el modelo global. Sin embargo, hay distintas estrategias de actualización que consisten en en combinar el modelo local y el global en lugar de reemplazar el uno por el otro directamente. Estos enfoques son usados para alcanzar características tales como la personalización de los clientes a sus datos locales.

10.4 ARQUITECTURAS DE FEDERATED LEARNING

La combinación de los elementos clave genera muchas arquitecturas de FL que definen su interrelación, ambas cliente-servidor y *peer-to-peer*:

- **Arquitectura Cliente-servidor**: hay un nodo que es el responsable de la coordinación y agregación de las actualizaciones del modelo que recibe el nombre de **servidor** y el resto de nodos responsables del entrenamiento de los datos locales con sus propios datos son llamados **clientes**. Es fácil de implementar pero requiere un nivel de confianza elevado en el servidor. Este grado de confianza es su principal debilidad y es por ello que es vulnerable a ataques.

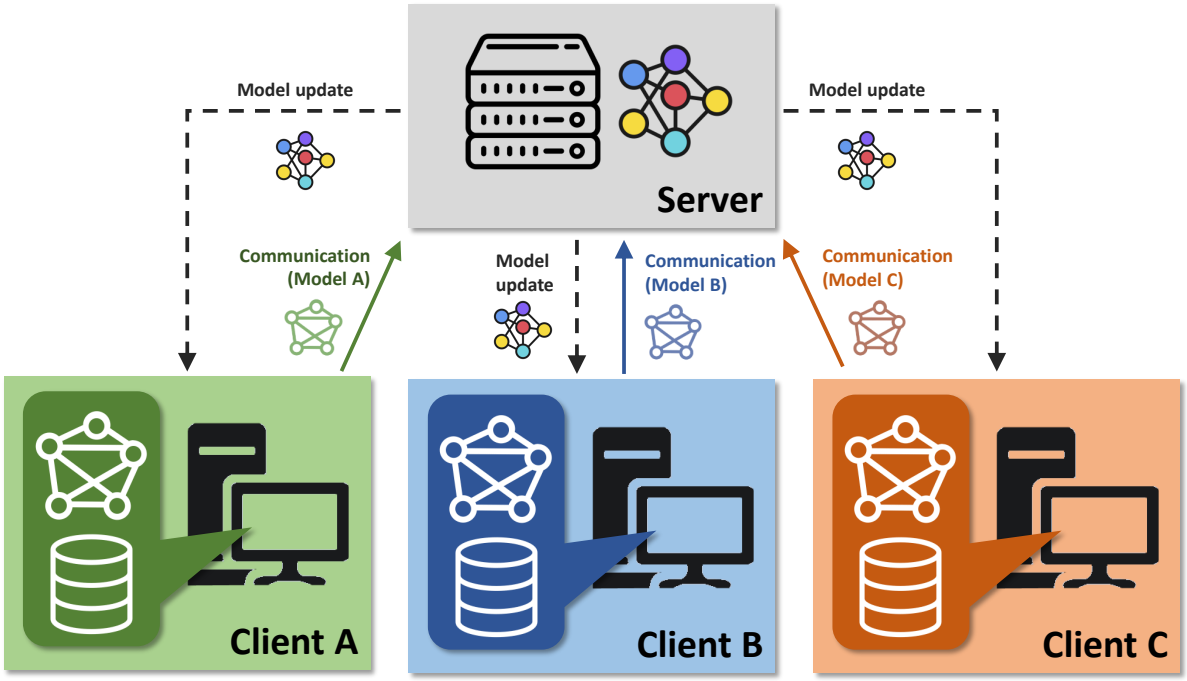


Figura 8: Representación de la arquitectura cliente-servidor en FL con 3 clientes. Fuente: [18].

- **Arquitectura *Peer-to-peer***: todos los nodos tienen datos locales de entrenamiento y agregan las actualizaciones de los demás nodos simultáneamente. No necesita de ningún nodo coordinador del proceso de aprendizaje fijo. Es complicada de implementar, y acarrea un aumento en los costes de comunicación, pero la principal ventaja es el elevado nivel de seguridad y privacidad de datos.

La arquitectura Cliente-Servidor es la más común en FL y consecuentemente nos referiremos a ella como la arquitectura por defecto cuando hablemos de FL.

10.5 CATEGORÍAS DE FEDERATED LEARNING

Hay múltiples categorías de FL según las propiedades de los elementos clave [29]. Las siguientes categorías están hechas en términos del espacio de características (X), el espacio de etiquetas (Y) y el espacio muestral (I).

- **Aprendizaje Federado Horizontal (*Horizontal Federated Learning*) (HFL)**: cuando los datos forman una partición entre los clientes basado en las muestras, que significa que cada cliente es propietario de diferentes muestras de el conjunto de entrenamiento general. Formalmente, lo podemos expresar como:

$$X_i = X_j, Y_i = Y_j, I_i \neq I_j, \quad \forall D_i, D_j, i \neq j \tag{119}$$

  donde los espacios de características y etiquetas de los clientes (i, j) están representados por (X_i, Y_i) y (X_j, Y_j) y asumiendo que son iguales, mientras que las muestras I_i y I_j no coinciden. D_i y D_j representan los datos de los clientes i y j. Es adecuado para entrenar modelos en datos recolectados por una gran cantidad de dispositivos similares, tales como *smartphones* o dispositivos IoT.
- **Aprendizaje Federado Vertical (*Vertical Federated Learning*) (VFL)**: cuando los datos forman una partición entre los clientes basado en las características, lo que significa que cda cliente es propietario del mismo conjunto de muestras pero con un conjunto de características distinto. Formalmente, lo podemos expresar como:

$$X_i \neq X_j, Y_i \neq Y_j, I_i = I_j, \quad \forall D_i, D_j, i \neq j \tag{120}$$

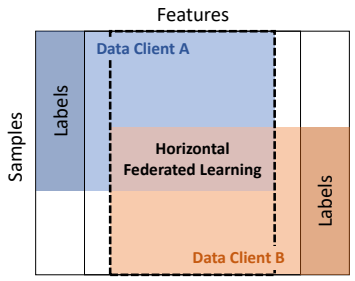
  Es adecuado para entrenar modelos en datos recolectados por una cantidad pequeña de dispositivos con un espacio de características distinto. Por ejemplo, puede ser usado para predecir resultados médicos basado en los datos recolectados por múltiples hospitales, donde cada hospital tiene un conjunto distinto de registros.
- ***Federated Transfer Learning* (FTL)**: cuando se transfiere conocimiento entre múltiples dominios sin que haya solapamiento entre muestras o características. Formalmente, lo podemos definir como:

$$X_i \neq X_j, Y_i \neq Y_j, I_i \neq I_j, \quad \forall D_i, D_j, i \neq j \tag{121}$$

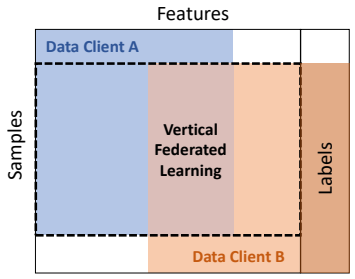
  En esta arquitectura, no se asume que la distribución de los datos tanto de entrenamiento como de validación sean los mismos y están definidos en el mismo espacio de características. Normalmente es usado en combinación con técnicas de *fine-tuning* con modelos entrenados sobre grandes conjuntos de datos centralizados.

10.6 PRIVACIDAD DE DATOS: TÉCNICAS AVANZADAS

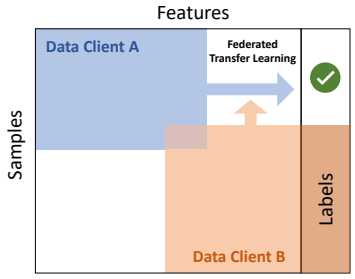
El FL ha sido desarrollado teniendo en cuenta la privacidad, esto es, los datos de los clientes se mantiene privado durante el entrenamiento del modelo. Sin embargo, es posible vulnerar estas garantías de privacidad mediante los modelos



(a) HFL



(b) VFL



(c) FTL

Figura 9: Representación de HFL, VFL y FTL en una arquitectura cliente servidor en FL [41].

intercambiados durante el proceso de aprendizaje, ya que los modelos locales de los clientes tienden a memorizar su conjunto de entrenamiento. Un nodo malicioso puede intentar recuperar parte del conjunto de entrenamiento de otros clientes, creando así una brecha de privacidad. Es por ello, se requieren técnicas de privacidad de datos para mejorar las promesas de privacidad de un modelo de FL. Consideramos que estas técnicas pueden ser desplegadas en múltiples elementos de la arquitectura de FL.

1. **Computación Segura y Cifrado Homomórfico:** la computación segura (*Secure Multiparty Computation*) se concentra en asegurar comunicaciones en las rondas de FL, principalmente en el mecanismo de agregación. Los canales de comunicación se mantienen seguros mediante el uso de cifrado homomórfico. Mientras estas técnicas evitan interferencias tanto internas como externas durante las rondas de FL, el modelo resultante sigue siendo vulnerable a ataques que se enfoquen en extraer información a partir del modelo agregado.

2. **Privacidad Diferencial:** la DP es una técnica que busca mejorar la privacidad. Se enfoca en ocultar la presencia de los individuos en el modelo. Esto se logra mediante la agregación calibrada de ruido aleatorio. Cuando se aplica en FL, la DP se puede usar en dos momentos distintos: a) aplicar la DP cuando se entrena el modelo localmente (DP local), y b) en el paso de agregación creando así una versión diferencialmente privada de FedAvg.

10.7 ATAQUES A FEDERATED LEARNING

El AA es vulnerable a ataques adversarios enfocados a reducir el rendimiento del modelo o a comprometer la privacidad de datos. Así mismo, el FL está igualmente expuesto dado que es un escenario específico de AA [29]. Algunos de esos ataques están basados en la manipulación maliciosa de los datos de entrenamiento, que son inaccesibles en el FL, y por lo tanto, no podemos confiar en el uso de técnicas de inspección de datos para detectar aquellos datos alterados. Es por ello que, uno de los grandes puntos débiles del FL es el ser vulnerable a ataques adversarios que pueden comprometer la integridad del modelo de aprendizaje o la privacidad de los datos.
Es por esto que esta sección se centrará en dar una serie de taxonomías en ataques de adversarios sobre el FL.

10.7.1 *Escenarios del Ataque*

Los escenarios del ataque en AA son una representación estructurada de la información, tales que permiten identificar y definir potenciales problemas de seguridad. Pueden ser definidos en términos de la información disponible y en el rango de acción del atacante. En este contexto, definimos los siguientes conjuntos de términos mutuamente exclusivos que nos permiten definir un modelo de ataque en FL.

ATACANTE EXTERNO VS. INTERNO: uno de los elementos clave de cualquier sistema distribuido es la comunicación entre distintas partes. La comunicación es muy vulnerable, ya que puede ser comprometida por agentes externos al sistema de aprendizaje, que son conocidos como atacantes externos o *outsiders*. Cuando el ataque es realizado por uno de los participantes del sistema distribuido, ya sea uno o más clientes, o incluso el servidor, se conoce como atacante interno o *insider*. Claramente, el alcance de los dos ataques es bastante diferente. Los ataques internos son son más dañinos y pueden estar enfocados a modificar el comportamiento del modelo o a inferir información de otros clientes, mientras que los ataques externos normalmente solo se enfocan a inferir información sobre los datos o del modelo resultante. Nos enfocamos en los ataques internos, de los que podemos destacar las siguientes categorías:

- **Ataques Bizantinos**: consisten en mandar actualizaciones arbitrarias al servidor, comprometiendo el rendimiento del modelo global.
- **Ataques sibilino**: consisten en ataques colaborativos, ya sea mediante la agrupación de varios ataques o simulando clientes ficticios con el fin de ser más dañinos.

CLIENTE VS. SERVIDOR: respecto a los ataques internos, en el HFL es normal diferenciar entre dos tipos de ataques, dependiendo si se están llevando a cabo por un cliente o por un servidor. La principal diferencia recae en la cantidad de información disponible. Mientras que los ataques realizados por clientes solo tienen información de uno o varios clientes, el servidor tiene información sobre la arquitectura del modelo y las actualizaciones de los clientes en cada ronda de aprendizaje. Incluso, en implementaciones criptográficas del medio de comunicación entre clientes y servidor, el servidor tiene más información que los clientes, ya que es el único con suficiente conocimiento para descifrar los modelos.

CONOCIMIENTO DEL ATACANTE: en escenarios centralizados, el atacante de caja blanca tiene acceso completo al modelo objetivo, incluyendo su arquitectura, los parámetros y el estado interno. Por el otro lado, el atacante de caja negra no tiene ningún acceso al modelo objetivo y además, puede tener algún tipo de información adicional sobre la arquitectura del modelo objetivo o su procedimiento de entrenamiento. Estas dos clasificaciones de conocimiento del atacante son demasiado generales para representar cualquier tipo de atacante, ya que no hay un término medio para considerar atacantes cuyo conocimiento en el caso de caja negra sea demasiado restringido o que en el caso de caja blanca no esté lo suficientemente acotado. Para ello, en [36] se introdujo un atacante de caja gris, que consiste en un atacante de caja negra con algún conocimiento estadístico específico no público que afecta a su víctima. En el FL, los atacantes de caja blanca, caja gris o caja negra pueden ser cualquier nodo, ya sea los clientes o el servidor. La mayoría de los ataques están relacionados con los datos pertenecientes a los clientes y a la comunicación entre el servidor federado y los clientes. Por lo tanto, también se requiere incluir información acerca del proceso de entrenamiento federado y de los datos privados de los clientes. Por lo tanto [29] define la siguiente clasificación de conocimiento de los atacantes adaptada a HFL y VFL:
En un sistema estándar de HFL, el atacante que tiene un cliente tiene **conocimiento por parte del cliente**:

- Acceso de caja blanca al modelo agregado.
- Acceso de caja blanca al modelo local entrenado por el cliente.
- Acceso al conjunto de datos del cliente.

Si el atacante tiene acceso a los datos locales de otros clientes o sus etiquetas, tiene **conocimiento extra por parte del cliente**.

Si el atacante es propietario del servidor federado tiene **conocimiento por parte del servidor**:

- Acceso de caja blanca al modelo agregado después de cada ronda de comunicación.
- Acceso de caja blanca a los modelos entrenados por los clientes o, alternativamente, acceso a sus gradientes.
- Los identificadores de los clientes agregados en cada ronda de comunicación.
- Las etiquetas de cada cliente y, opcionalmente, el tamaño de su conjunto de datos.

En sistema estándar de VFL, el atacante que tiene a un cliente tiene **conocimiento por parte del grupo**:

- Acceso de caja blanca a los parámetros relacionados con las características del cliente.
- Acceso al conjunto privado del cliente.
- El resultado parcial de los parámetros, cuando se solicita inferencia.

Además, si el atacante tiene información relacionada con las características de otros clientes, tiene **conocimiento extra por parte del grupo**.

Un atacante que tiene al coordinador de aprendizaje de un sistema de VFL tiene **conocimientos del lado de terceros**:

- Los gradientes compartidos por cada cliente.
- La pérdida calculada.
- Los resultados parciales de cada cliente, cuando se solicita inferencia.

Si solo un subconjunto de los conocimientos especificados está disponible, entonces el atacante tiene **conocimiento parcial**, y especificamos el subconjunto del conocimiento. Además, se espera que las defensas reduzcan el conocimiento del atacante, por lo tanto, en presencia de un método de defensa el atacante se supone que tiene **conocimiento parcial**.

Tanto en HFL como en VFL, si el atacante solo tiene acceso a los resultados del modelo federado, entonces tiene **conocimiento del lado externo**.

HONESTO PERO CURIOSO VS. MALICIOSO: un atacante malicioso (o activo) intenta interferir en el proceso de entrenamiento del modelo con el objetivo corromper el modelo, por ejemplo, dañando su rendimiento o inyectando una tarea secundaria. Por el otro lado, un atacante honesto pero curioso (o pasivo) no interfiere en el proceso de entrenamiento y sigue los protocolos de aprendizaje, pero intenta obtener información privada sobre otros clientes a partir de la información recibida.

COLUSIÓN VS. NO COLUSIÓN: la amenaza de colusión se basa en el hecho de que el atacante que controle más clientes tiene más poder en un sistema distribuido. Hay dos tipos de colusión:

1. **Servidor-Participantes**: el atacante controla algunos participantes benignos y el servidor, e intenta inferir información sobre el resto de clientes.
2. **Participante-Participante**: el atacante controla una fracción de clientes benignos e intenta inferir información sobre otros clientes, el servidor o intenta dañar el modelo.

10.7.2 *Ataques Adversarios en Federated Learning: taxonomías*

Los ataques adversarios representan uno de los problemas más difíciles en FL, debido a la alta cantidad de ataques existentes y a la dificultad de defenderse ante ellos. Además, la naturaliza distribuida del FL lo hace vulnerable a una gran variedad de ataques adversarios con distintos objetivos y usando distintas maneras de alcanzar estos objetivos. Es por ello que es difícil establecer una taxonomía común para todos los los tipos de ataques adversarios. Es por ello que en [29] se propone primero una amplia clasificación diferenciando entre:

- **Ataques al modelo federado**, que buscan cambiar su comportamiento.
- **Ataque a la privacidad de datos**, cuyo propósito es inferir información sensible del proceso de aprendizaje.

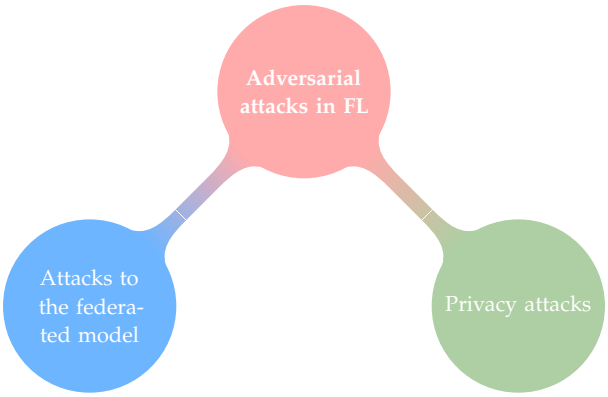


Figura 10: Categorización de los ataques adversarios en dos grandes categorías. Fuente: [29].

En este trabajo nos interesaremos sobretodo en los **ataques al modelo federado** y son en los que haremos hincapié en lo que sigue del trabajo.

10.7.3 *Ataques Adversarios al Modelo Federado*

Una de las principales limitaciones del FL, y más concretamente el HFL, en términos de ataques adversarios, es que los clientes tienen la habilidad de dañar el modelo mediante el envío de actualizaciones envenenadas, mientras que el servidor no puede inspeccionar los datos de entrenamiento almacenados por los clientes. Esto hace que

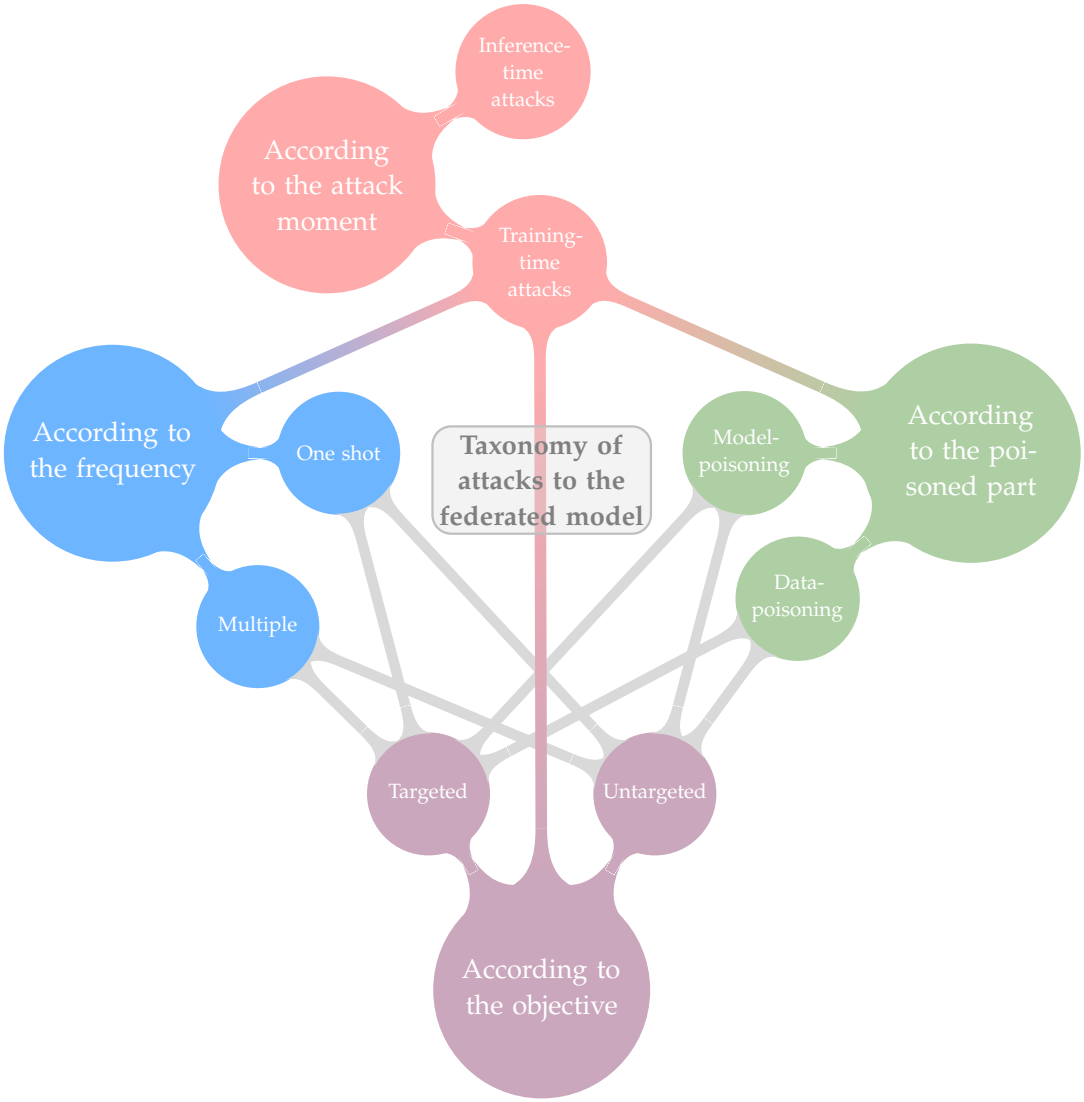


Figura 11: Representación de las taxonomías de los ataques según varios criterios. Los enlaces grises representan la posibilidad de combinar dos categorías. Fuente: [29].

los ataques adversarios al modelo federado sean uno de los retos más importantes en el FL.

En general, estos ataques son realizados por clientes y por tanto cuentan con características de caja blanca ya que el atacante tiene conocimiento por parte del cliente, ya sea que haya uno o varios clientes adversarios (atacantes). En algunas situaciones se considera que los atacantes tienen más información de caja blanca, como puede ser el método de agregación usado por el servidor, lo cual no es algo realista, por lo que [29] solo tiene en cuenta aquellos ataques que requieren información del cliente adversario.

Dentro de esta amplia categoría, se propone una taxonomía que clasifica un abanico de ataques según diferentes criterios. Así, cada tipo de ataque en la literatura pertenece a cuatro categorías distintas, una para cada criterio.

taxonomía según el momento del ataque

Esta taxonomía representa en que momento se realiza el ataque, lo cual determina completamente la capacidad de influenciar al modelo federado. Clasificamos en dos tipos de ataques:

- **Ataques durante entrenamiento**: esta fase incluye la recolección de los datos la preparación de estos para el entrenamiento. Son los ataques más comunes en la literatura ya que tienen la capacidad de alterar el modelo federado que es entrenado e inferir información sobre los datos de entrenamiento.
- **Ataques durante inferencia**: se llevan a cabo durante la fase de inferencia cuando el modelo ya ha sido entrenado. Se llaman también ataques exploratorios. Generalmente no pretenden alterar el modelo entrenado sino producir predicciones erróneas o recolectar información sobre las características del modelo.

taxonomía según el objetivo del ataque

Este es el criterio más usado en la literatura y por tanto el más importante a la hora de clasificar estos ataques. Podemos denotar dos grandes grupos de ataques según el objetivo del ataque:

1. **Ataques de *Backdoor***: el principal objetivo es inyectar una tarea secundaria o *backdoor* en el modelo. Es decir, un ataque de este tipo será exitoso si logra mantener el rendimiento de la tarea original mientras inyecta otra tarea. Estos ataques son muy silenciosos, ya que no suelen afectar al rendimiento de la tarea original, haciéndolos difíciles de detectar. Pese a que no suponen un riesgo a la principal tarea de aprendizaje, si representan un peligro para la integridad del sistema, ya que el atacante se aprovecha de la infraestructura federada para realizar alguna acción secundaria, logrando así una brecha de seguridad. La naturaleza de estos ataques es amplia, debido a la inmensa cantidad de tareas secundarias posibles. Se presenta una taxonomía basada en diferentes criterios:
    - **Estrategias basadas en Entrada-Instancia**: el objetivo es que el modelo etiqueta ciertas entradas concretas con una etiqueta concreta diferente de la original. Por ejemplo, en un sistema de reconocimiento facial que permite el acceso a una casa, identificar cinco personas de la entrada original, quienes originalmente no tenían acceso (etiqueta negativa como etiqueta original) como personas que sí tienen acceso (etiqueta positiva como etiqueta objetivo).
    - **Estrategias basadas en patrones**: el objetivo es que el modelo asocia un patrón particular con una etiqueta en concreto. Por ejemplo, en el ejemplo anterior, permitir acceso a cualquier persona que lleve una corbata de

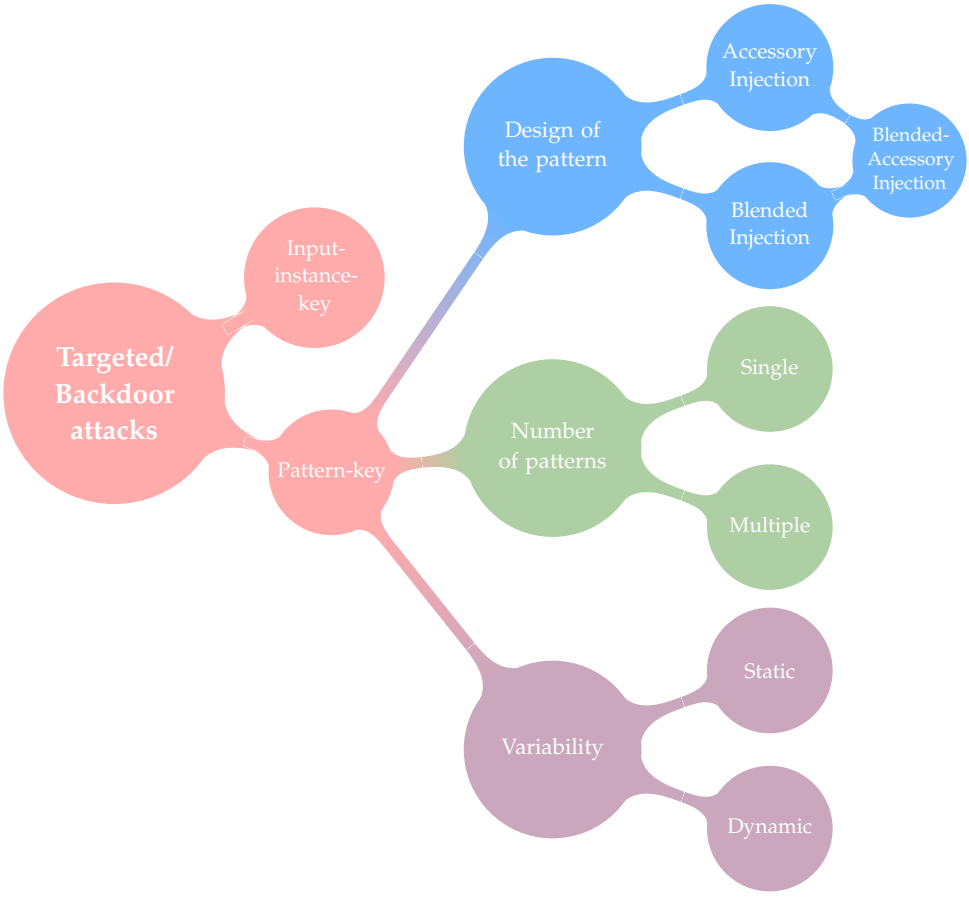


Figura 12: Representación de la taxonomía de ataques de *Backdoor*. Fuente: [29].

puntos. De esta manera, el sistema identificaría el patrón "corbata de puntos" con la etiqueta objetivo (etiqueta positiva). En la práctica, se escoge un patrón simple como una cruz o una marca parecida para la asociación.
Además, estos ataques también se pueden clasificar según diferentes criterios sobre el patrón inyectado.
Según el diseño del patrón se puede clasificar de la siguiente manera:

- **Estrategia de Inyección Mezclada**: esta estrategia generas instancias *backdoor* mezclando una entrada benigna con un patrón usando un ratio de mezcla. El patrón puede ser cualquier imagen, o incluso patrones aleatorios. La principal limitación es que el mecanismo requiere modificar la mezcla completa durante entrenamiento y evaluación, lo que no tiene por qué ser viable.
- **Estrategia de Inyección con Accesorio**: este ataque aparece como solución la estrategia de inyección mezclada y propone generar imágenes *backdoor* añadiendo patrones sobre algunas regiones de las imágenes originales. Es el equivalente a vestir un accesorio en la vida real.
- **Estrategia de inyección mezclada con accesorio**: combina ambas estrategias mencionadas anteriormente.

Según la cantidad de patrones:

- **Ataque de Patrón Único**: se refiere a cuando todos los clientes adversarios inyectan el mismo patrón al modelo. Tienden a tener más éxito ya que con un ataque colectivo con el mismo objetivo, pero a la vez son más fáciles de identificar en el servidor.
- **Ataque *Multi-Backdoor***: está compuesto de varios clientes adversarios coordinados (sibilino), donde cada cliente inyecta un patrón diferente o una parte de un patrón común en el modelo. Son más difíciles de detectar en el servidor debido a la distribución del patrón entre clientes. Sin embargo, es más complicado para los clientes inyectar tareas de *backdoor* en el moddelo debido a la diversidad de las tareas secundarias.

Según la variabilidad del patrón a lo largo del tiempo:

- **Ataque estático**: cuando el patrón se mantiene a lo largo del tiempo pese a la frecuencia del ataque. Esta situación es la más común.
- **Ataque dinámico**: el patrón cambia a lo largo del tiempo, lo cual se convierte en un reto tanto para las defensas, ya que el patrón a identificar cambia, y para los clientes adversarios, que sufren un aumento en coste computacional para adaptar las nuevas tareas secundarias.

2. **Ataques sin objetivo**: en contraposición con los ataques de *backdoor*, el único objetivo de los ataques sin objetivo es reducir el rendimiento del modelo en la tarea original. El escenario más extremo es conocido como **ataques Bizantinos**, en el que los clientes adversarios comparten actualizaciones generadas aleatoriamente o entrenadas sobre datos modificados de manera aleatoria, generando por lo tanto también actualizaciones aleatorias. Claramente, estos ataques son menos silenciosos que los ataques de **backdoor** y por lo tanto pueden ser detectados simplemente midiendo el rendimiento del modelo local en el servidor, aunque a veces es difícil distinguirlos de clientes con distribuciones de datos bastantes particulares.

TAXONOMÍA SEGÚN LA PARTE DEL ESQUEMA ENVENENADA

La mayoría de ataques durante el entrenamiento están basados en envenenar la información del cliente con la intención de corromper el modelo global. Dependiendo qué parte de la información del cliente se envenene, podemos diferenciar entre envenenamiento de datos y envenenamiento de modelo, aunque nos referiremos a ambos como ataques de envenenamiento.

- **Ataques de Envenenamiento de Datos**: se asume que el atacante tiene acceso a los datos de entrenamiento de uno o más clientes y es capaz de modificarlos. Según las características del envenenamiento, podemos distinguir los siguientes ataques:
  - **Ataque *label-flipping***: consiste en modificar las etiquetas de una parte del conjunto de datos. Puede ser con un objetivo, cambiando algunas etiquetas específicas, o sin objetivo, mediante barajando las etiquetas de manera aleatoria.

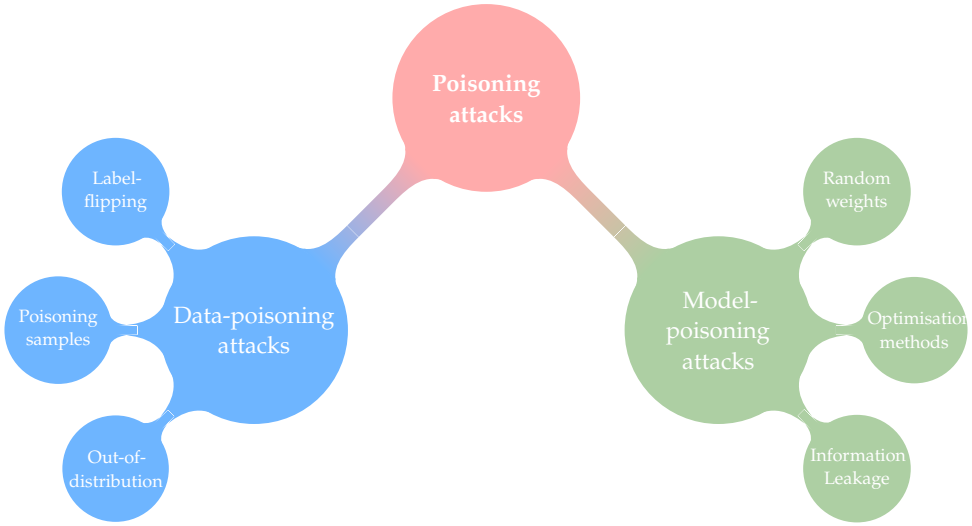


Figura 13: Representación de la taxonomía según la parte del esquema envenenada. Fuente: [29].

- **Ataque de envenenamiento de muestras**: este ataque consiste en modificar una parte de las muestras del conjunto de entrenamiento. El envenenamiento puede ser de distintos tipos, tales como añadir patrones en las muestras y asociarlos con una clase, o normalizar las muestras y añadir ruido uniforma con el objetivo de reducir el rendimiento del modelo.
- **Ataque fuera de distribución**: este ataque es similar al anterior, con la diferencia que las muestras envenenadas no son modificaciones de las originales sino muestras externas a la población original. Es posible usar muestras de otro dominio con las mismas características o muestras hechas por ruido aleatorio.

El objetivo de la mayoría de los ataques de envenenamiento de datos es reducir el rendimiento del modelo global y por tanto el modelo local de todos los clientes. Sin embargo, también existe el caso en el que el objetivo de los atacantes no es reducir el modelo global sino solo un subconjunto suyo. Es por ello por lo que podemos diferenciar tres tipos de ataques de envenenamiento de datos según el nivel de acceso que los atacantes tienen sobre los nodos objetivo:

- **Ataque directo**: los atacantes tienen acceso a los nodos objetivo, así que los envenenan directamente.
- **Ataque indirecto**: los atacantes no tienen acceso a los nodos objetivo, así que aplican mecanismos tales como entrenar sus propios modelos con datos envenenados para envenenar el modelo global, que luego será compartido con los nodos objetivo.
- **Ataque híbrido**: cuando se combinan ambos ataques mencionados.

En la literatura los ataques más comunes son los ataques directos.

- **Ataques de Envenenamiento de Modelo**: estos ataques consisten en envenenar directamente las actualizaciones del modelo que son mandadas por los clientes al servidor. Pese a que los ataques de envenenamiento de datos llevan a ata-

ques de envenenamiento de modelo, nos centraremos en aquellos ataques que modifican directamente las actualizaciones locales. Según cómo son generadas, distinguimos entre:

- **Generación de pesos aleatorios**: estos ataques se basan en generar pesos del modelo como un vector de la misma dimensión que los pesos del servidor generado aleatoriamente.
- **Métodos de optimización**: consisten en maximizar el rendimiento en una tarea de *backdoor*, mientras se minimizan las diferencias del modelo envenenado respecto al compartido con el servidor en la última ronda, maximizando así la efectividad y sigilo. Este ataque se realiza como un problema de optimización multiobjetivo. Este tipo de ataque es posiblemente el más eficaz para realizar ataques de *backdoor* en el modelo.
- **Brecha de información**: un caso particular de los ataques de envenenamiento de modelo es la brecha de información, donde el objetivo no es perjudicar al modelo global, sino la comunicación entre los atacantes a través de un protocolo seguro. Consiste en que varios clientes están coordinados dde tal manera que conocen reglas comunes y mediante la modificación de pequeñas partes de los pesos del modelo pueden comunicarse.

En el FL, asumiendo que la proporción de clientes adversarios es significativamente inferior a la de los benignos, el efecto del ataque se espera que sea disipado en el proceso de agregación. Por lo tanto, se aplican técnicas de *model-replacement*, que consisten en ponderar la contribución de los clientes adversarios usando técnicas de *boosting* con el objetivo de reemplazar el modelo agregado con sus actualizaciones locales. Formalmente, si consideramos la actualización del modelo global en la ronda t se computa según (122):

$$G^t = G^{t-1} + \frac{\eta}{n} \sum_{i=1}^{n} (L_i^t - G^{t-1}) \tag{122}$$

donde η es el *learning rate* del servidor y n la cantidad de clientes participando en la agregación. Entonces consideramos el modelo local del cliente adversario entrenado en los datos envenenados como en (123):

$$\hat{L}_{adv}^t = \beta (L_{adv}^t - G^{t-1}) \tag{123}$$

donde $\beta = \frac{n}{\eta}$ es el factor de *boosting*. Combinando entonces (122) y (123) tenemos[1] que:

$$G^t = G^{t-1} = \frac{\eta}{n}\frac{n}{\eta}(L_{adv}^t - G^{t-1}) + \frac{\eta}{n} \sum_{i=2}^{n} (L_i^t - G^{t-1}) \tag{124}$$

1 Asumimos que el adversario es el cliente 1

Además, debido a que el modelo de FL convergerá a una solución, se puede asumir que $L_i^t - G^{t-1} \approx 0$ para los clientes benignos. Por lo tanto se puede reescribir (124) como:

$$G^t \approx G^{t-1} + \frac{\eta}{n}\frac{n}{\eta}(L_{adv}^t - G^{t-1}) = L_{adv}^t \tag{125}$$

Si hay varios clientes adversarios, el factor de *boosting* se divide entre todos ellos.

Las técnicas de *boosting* dependen de saber el número de clientes participando en la agregación, lo cuál es una condición mucho más restrictiva de conocimiento por parte del cliente. En la práctica, el cliente estima este valor haciendo varias pruebas con distintos valores y analizando las actualizaciones devueltas por el servidor. Sin embargo, en la mayoría de trabajos experimentales se asume la peor situación en la que este conocimiento es sabido.

TAXONOMÍA SEGÚN LA FRECUENCIA

Debido a que el entrenamiento ocurre durante un periodo prolongado en el tiempo, los ataques durante entrenamiento pueden ser realizados en cualquier momento durante el entrenamiento y en una o varias ocasiones. Diferenciamos entre dos categorías:

- **Ataque *One-shot***: este ataque es realizado en un único momento durante el entrenamiento, en una ronda de aprendizaje en específico.
- **Ataque Múltiple o Adaptativo**: los ataques se realizan de manera continua durante el proceso de entrenamiento, ya sea durante todas las rondas de aprendizaje o sobre una porción de ellas.

10.8 MÉTODOS DE DEFENSA CONTRA ATAQUES ADVERSARIOS

Mientras que la diversidad y la complejidad de los ataques adversarios contra el FL está aumentando, aparecen nuevas defensas para mitigar sus efectos maliciosos. Aunque los ataques adversarios pueden ser divididos en categorías disjuntas, esto no se cumple para sus defensas ya que algunas son efectivas contra más de un tipo de ataque. Por lo tanto, en lugar de agrupar las defensas según el ataque contra el que defienden, [29] propone categorizarlas en tres grupos según el esquema federado en el que se implementan: el cliente, el servidor o el canal de comunicación.

10.8.1 *Defensas en el Servidor*

Normalmente se asume que se puede confiar en el servidor, ya que es un elemento federado controlado y accesible por expertos en FL, al contrario que los clientes que

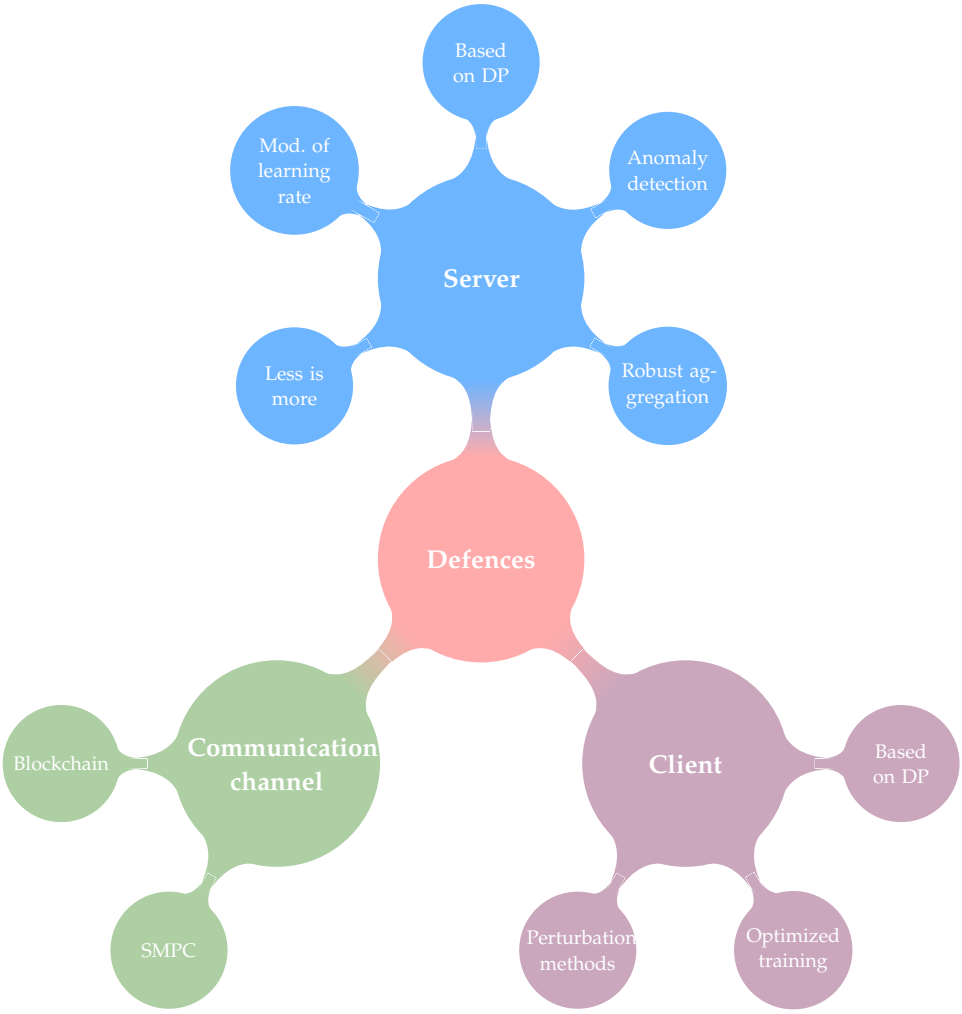


Figura 14: Representación de la taxonomía de las defensas contra ataques adversarios. Fuente: [29].

son elementos independientes e inaccesibles. Por lo tanto, la mayor parte de las defensas son implementadas en el servidor. Dentro de este tipo de defensas, se presenta la siguiente taxonomía. Debemos de tener en cuenta que algunas defensas pueden tener elementos de más de una categoría , aunque dada una defensa esta se ha clasificado en la categoría que mejor la representa según [29].

OPERADORES ROBUSTOS DE AGREGACIÓN

La técnica más común para defenderse ante ataques de envenenamiento es el usar estimadores que sean más robustos que la media a *outliers* o valores extremos, desde un punto de vista estadístico. Algunos agregadores, como FedAvg [20], son sensibles a *outliers*. Es por esta razón que se han propuesto muchos operadores de agregación basados en estimadores más robustos. Pueden destacarse los siguientes:

- *Mediana:* se basa en reemplazar la media aritmética por la mediana de las actualizaciones de los modelos.

- *Media Geométrica:* representa la tendencia central o el valor típico de la distribución mediante el uso del producto de sus valores. Se puede decir que escoge un vector que representa las actualizaciones de los modelos locales mediante voto de la mayoría.
- *Krum y Multikrum [2]:* este agregador está diseñado para prevenir ataques al modelo federado. Consiste en filtrar las actualizaciones que representan un comportamiento externo. Para ello se ordenan los clientes según las distancias geométricas de sus distribuciones y se escoge la más cercana a la mayoría como modelo agregado. Multikrum incorpora un parámetro que especifica la cantidad de clientes a ser agregados.
- *Bulyan:* pensado para evitar ataques de envenenamiento, combina el operador de Multikrum con la media recortada.

DETECCIÓN DE ANOMALÍAS

Estos métodos de defensa consisten en identificar clientes adversarios mediante datos anómalos en la distribución y retirarlos de la agregación. Para este propósito, se aplican técnicas de detección de anomalías multivariante o univariante en AA.

Uno de los mecanismos propuestos es *AUROR*, un mecanismo contra ataques de envenenamiento basado en *K-Means* con $k = 2$, por lo tanto distinguiendo entre *clusters* benignos y maliciosos. El principal problema de esta propuesta es que si se da la presencia de distribuciones no-i.i.d. entre los clientes podría fallar a la hora de identificar los *clusters*.

El principal problema con técnicas de detección de anomalías es que las actualizaciones del modelo tienden a ser de una dimensión muy alta, ya que vienen de redes neuronales la mayoría de ocasiones. Es por ello que aparecen varias propuestas que intentan reducir la dimensionalidad de las datos ya sea basándose en análisis de componentes principales o en técnicas similares, aunque conllevan también una pérdida de información en el proceso.

DEFENSAS BASADAS EN PRIVACIDAD DIFERENCIAL

Aunque la privacidad es algo que se escapa a los ataques adversarios al modelo, la DP ha demostrado ser una defensa viable contra este tipo de ataques. Sin embargo, también se sabe que la DP puede impactar negativamente al rendimiento de los modelos bajo circunstancias de un desequilibrio en las distribuciones datos, que es lo esperado a ocurrir en la mayoría de escenarios federados.

MODIFICACIÓN DEL *learning rate*

Una de las ventajas del servidor es que es él el que establece el peso entre la versión previa del modelo y de las actualizaciones de los modelos locales según

$$G^t = G^{t-1} + \eta\Delta(L_1^t, \ldots, L_n^t) \tag{126}$$

donde G^t es el modelo global en la ronda t, η es el *Learning Rate* (lr), Δ el operador de agregación y L_i^t las actualizaciones del cliente i en la ronda t. Otra posibilidad es la de descomponer η en un vector de lr con una componente por cada dimensión. Por lo tanto en servidor controla la participación de cada dimensión en las actualizaciones del modelo.

Se propone *Robust Learning Rate* como un mecanismo de defensa basado en ajustar el lr del servidor, por dimensión, en cada ronda de aprendizaje según la información de las actualizaciones de los clientes. Para cada dimensión, se examina si los clientes acuerdan la dirección de las actualizaciones usando algún tipo de umbral predefinido. Si el acuerdo es mayor que el requerido por el umbral, se mantiene el lr, en caso contrario se cambia el signo del lr.

10.8.2 *Defensas en el Cliente*

Las defensas en el servidor asumen que el servidor es un recolector de datos y un agregador fiable. Sin embargo, asumir esto puede ser demasiado, por lo tanto hay una necesidad de defensas cuando no asumimos un servidor fiable. En esta situación se despliegan defensas en el cliente y por lo tanto, al menos una parte de los clientes se supone que es benigna. Si bien las defensas en el servidor protegían a los clientes como conjunto, estas se consideran más fuertes pues defienden a cada cliente individualmente.

BASADAS EN PRIVACIDAD DIFERENCIAL

Generalmente, este tipo de defensas están pensadas para defenderse contra ataques de privacidad en el servidor, aunque pueden proteger a los clientes de ataques adversarios. La DP local es la principal defensa basada en DP en el cliente. Por lo tanto muchos autores han propuesto mejor a la DP local mediante la disminución de su uso, es decir, usando el menor ruido necesario. Se ha estudiado su efectividad contra ataques adversarios y se ha relacionado el impacto negativo al rendimiento de esta técnica con la reducción de la efectividad del ataque adversario.

MÉTODOS DE PERTURBACIÓN

Son una técnica alternativa a la DP para defenderse contra ataques a la privacidad. Su principal objetivo es el introducir ruido a las partes más vulnerables del modelo federado, tales como los parámetros compartidos o al conjunto de datos local de cada cliente, para reducir la cantidad de información que el atacante puede extraer. Algunos trabajos proponen podar los gradientes que estén por debajo de cierto umbral o perturbar los datos locales de los clientes con una red neuronal.

ENTRENAMIENTO OPTIMIZADO

La optimización del entrenamiento de los clientes benignos puede ser un método para proteger al sistema federado de ataques adversarios. Por ejemplo, se propone realizar *fine-tuning* del modelo en los clientes benignos con el objetivo de aumentar el impacto de estos clientes en la agregación. Para decir que clientes son benignos, se hace uso de *matching networks*, que consiste en medir la similitud entre las actualizaciones de los modelos y el modelo anterior. En casos experimentales, se logra filtrar tareas de *backdoor* aunque conlleva un impacto negativo en el rendimiento de la tarea original.

10.8.3 *Defensas en el Canal de Comunicación*

Estas defensas permiten a varios clientes realizar una tarea global, asumiendo la presencia de actores maliciosos que intentan perjudicar a la tarea. Estos actores pueden ser vistos como los atacantes en nuestro caso, que intentan realizar algún tipo de ataque adversario mencionado previamente. Mientas que se mantiene la privacidad de los datos usados para realizar la tarea global, el resultado de esta se revela a algunos grupos, sino a todos. Por lo tanto la privacidad del resultado no está asegurada, aunque se detienen algunos ataques de privacidad ya que el atacante pierde acceso a los resultados intermedios de la tarea global tales como parámetros o gradientes usados por los clientes.

FEDERATED LEARNING BASADO EN BLOCKCHAIN

Esta será la defensa más importante de nuestro trabajo y en la que nos centraremos más adelante en este trabajo en los próximos capítulos. La tecnología *Blockchain* permite un esquema de FL descentralizado sin riesgos de un único punto de fallo y una escalabilidad mejorada. Aún así también implica algunos riesgos inherentes a las *Blockchain* como ataques del 51 % o ataques de *forking* entre otros. Sin embargo veremos más adelante como pueden ser realmente útiles como defensas ante ataques adversarios al modelo.

11

BLOCKCHAIN

Durante la historia de la humanidad, el ser humano siempre ha realizado transacciones entre ellos siendo un tema delicado la confianza en prójimo. Es por ello que se recurren a intermediarios para resolver este problema. Este es el caso de los bancos para actividades financieras, servidores centrales para almacenar datos y computación o incluso gobiernos para mantener la sociedad. La confianza puesta en los intermediarios es asumida: asumimos que van hacer lo que se suponen que tienen que hacer. Sin embargo esto no es así en la práctica. Las máquinas fallan, las personas cometen errores.

En este contexto, surge la tecnología *blockchain* [5]. Una tecnología descentralizada en la que no actúa ningún intermediario. La *blockchain* intenta solventar las debilidades del intermediario centralizado en cuatro grandes aspectos: confianza, seguridad, privacidad y transparencia. Mientras que un servidor como único punto de contacto puede ser atacado y los datos almacenados pueden ser alterados de manera maliciosa, esto queda solventado con la *blockchain*.

11.1 ¿QUÉ ES LA *blockchain*?

Una vez introducida la motivación para la *blockchain* y su potencial, nos centraremos en qué es. Se puede definir de manera informal basándonos en qué nos ofrece: una tecnología para registrar transacciones y procesamiento que es segura (no puede haber ni pérdida ni alteración de los datos) y *trustless* (no hace uso de ningún intermediario en el que confiar). Una definición más completa sería verla como un sistema de computación descentralizada con cinco componentes:

- **Red descentralizada**: la *blockchain* confía en una red descentralizada de ordenadores, llamados nodos, que contribuyen con recursos de computación para ayudar a almacenar y procesar transacciones. Estos ordenadores trabajan de manera autónoma y se comunican mediante una red *Peer-to-Peer* (P2P). La mayoría de las redes *blockchain* incluyendo a Bitcoin [13] adoptan una topología P2P no estructurada, es decir, cada nodo escoge a sus vecinos de manera arbi-

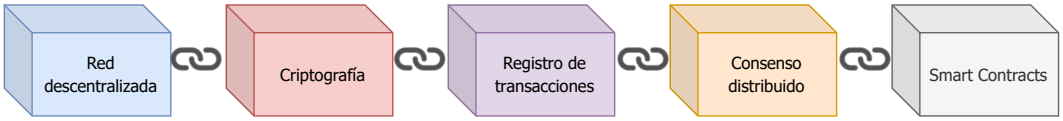


Figura 15: Diagrama indicando los 5 componentes de una *blockchain*. Fuente: elaboración propia.

traria. Otra redes como Ethereum [3] usan una estructurada. Una topología no estructurada puede ser menos eficiente pero más fácil de mantener.

- **Criptografía**: los métodos de criptografía usados en la *blockchain* dan demostraciones matemáticas de que la *blockchain* debe de funcionar como se supone. Se usan *hashes* criptográficos para vincular bloques de datos en la cadena de forma que no se permita ninguna modificación después de registrarlos en la *blockchain*. Cada transacción se encripta con criptografía de clave pública para asegurar que quien crea la transacción es verificable usando una firma digital y que solo quien se indica como receptor puede recibirla. La confidencialidad de la transacción se logra gracias al uso de *Zero Knowledge Proofs* [44], un mecanismo criptográfico que permite validar la veracidad de una afirmación sin conocer información sensible sobre esta. La elección de la criptografía a usar determina el rendimiento y garantías de la *blockchain*.
- **Registro de transacciones**: Como tecnología de almacenamiento, la *blockchain* es un registro digital que almacena las transacciones en bloques de manera cronológica.
- **Consenso distribuido**: cuando se tiene que tomar alguna decisión, por ejemplo, si una transacción es válida, no hay ninguna autoridad central para decidir. En su lugar, la decisión se hace basándose en un consenso entre los nodos participantes. Por lo tanto, la red *blockchain* debe de tener un protocolo de consenso para asegurarse de que cada transacción o bloque añadido a la *blockchain* es la única opción válida que es aceptada por todos los nodos. La elección del mecanismo de consenso es la decisión más importante a la hora de diseñar una red *blockchain*, por ello más adelante nos centraremos este componente más adelante.
- ***Smart Contracts***: una *blockchain* se puede ver como un ordenador no convencional que realiza ciertas tareas. En lugar de integrar unidades de procesamiento (CPUs) como un ordenador normal, la *blockchain* es un ordenador descentralizado que usa cientos o miles de ordenadores alrededor del mundo. Las aplicaciones que se ejecutan en la *blockchain* son conocidas como *smart contracts*.

11.2 LA ESTRUCTURA DE CADENA

Normalmente, salvo en casos poco populares, el registro de la *blockchain* sigue una estructura de cadena [11]. Los datos están organizados en bloques de datos: $\{b_1, b_2, \dots\}$.

Cuando se necesita almacenar nuevas transacciones, estas se colocan en un nuevo bloque que se añadirá después del último bloque ya existente de la cadena. Además de almacenar la información de la transacción, el estado de la *blockchain* si procede, e información necesaria en una cabecera, el bloque tiene dos atributos fundamentales:

- **ID del bloque**: este es el *hash* del contenido del bloque usando una función de *hash* criptográfica H. Esta función está predefinida y se conoce de manera pública.
- **El hash anterior**: es el ID del bloque anterior.

Notemos que el ID del bloque no tiene por qué ser necesariamente almacenado en el bloque ya que se puede calcular a partir del contenido del bloque.

El *hash* anterior es una información crítica para mantener la integridad de la cadena. Si cualquier parte de cualquier bloque cambiase después de registrarse en la *blockchain*, este cambio será detectado. Esto es porque para añadir un nuevo bloque a la *blockchain* se debe de cumplir un proceso llamado validación del bloque [5]. Un nuevo bloque b_{i+1} es válido si:

1. El *hash* anterior es consistente, esto es: $b_{i+1}^{prev} = H(b_i)$.
2. Todas las transacciones de b_{i+1} son válidas.
3. El bloque b_i es válido.

La verificación en el paso 1 necesita calcular el *hash* de b_i y compararlo con b_{i+1}^{prev}. El paso 3 necesita ejecutar el mismo proceso de validación para verificar la validez del bloque b_i. Consecuentemente, para validar el bloque b_{i+1}, se necesita comprobar todos los *hash* anteriores de todos los bloques b_j con $j \leq i+1$. Si un bloque anterior, digamos, b_{j-1} ha sido modificado, entonces cuando calculemos su *hash*, $H(b_{j-1})$, encontraremos que es distinto del valor b_j^{prev} almacenado en el bloque b_j. Esto resulta en una infracción y como resultado el nuevo bloque b_{i+1} se considerará inválido y no se añadirá a la *blockchain*.

Una consecuencia de haber modificado el bloque b_j es que la *blockchain* no podrá crecer más. Uno podría pensar que en ese caso la *blockchain* se vuelve inútil ya que detiene a toda la *blockchain*. Esto sería cierto si la red *blockchain* estuviese compuesta de un único ordenador. En la práctica, la red *blockchain* está compuesta de varios ordenadores [5], donde los datos de la *blockchain* están duplicados en cada nodo. Para que un nodo se asegure de que su copia de la *blockchain* es correcta, necesita comparar su copia con la de sus vecinos para escoger la *blockchain* más larga como la correcta [13]. Antes de realizar esta comparación, el nodo necesita verificar la validez de la copia de cada vecino, lo que implica validar todos los bloques de la copia. Por lo tanto, si la copia de la *blockchain* de algún nodo incluye alguna infracción, esta copia no será considerada válida. Por lo tanto, los nodos honestos de la red nunca usarán una copia errónea [13].

11.3 CÓMO CONSEGUIR UN CONSENSO

El consenso ya era un área de investigación en computación con 30 años de estudio antes de que la tecnología *blockchain* se volviese popular. Comenzó en los años 70 con el proyecto de la NASA, "*Software Implemented Fault Tolerance* (SIFT)" [9], enfocado a construir un sistema de control de naves resistente. El reto consistía en replicar el sistema en múltiples máquinas de manera que el sistema se pudiese resistir fallos de múltiples de ellas. Más adelante este reto se formuló como el ahora conocido "Problema de los Generales Bizantinos" [15]. Este problema popularizó el término "Fallo Bizantino (*Byzantine Fault*)" para indicar una condición de un sistema distribuido donde algunos nodos no son confiables y pueden aparentar normales o maliciosos de manera arbitraria, pudiendo camuflarse entre ellos de manera que no haya información consistente para que los otros nodos declaren su mal funcionamiento.

Un sistema sistema Tolerante a Fallos Bizantinos (*Byzantine Fault Tolerance*) (BFT) [9] debe de evitar el fallo completo y por tanto los nodos deben de acordar una estrategia común y seguir este consenso, sabiendo que algunos nodos pueden fallar o actuar de manera maliciosa. Para describir BFT de manera formal, consideremos un sistema de difusión donde un nodo necesita difundir un mensaje (valor) a todos los demás nodos de manera par a par. Al principio, en nodo emisor recibe un valor de entrada m. El protocolo de difusión debe concluir con que cada nodo i tenga un valor de salida m_i. El emisor y los receptores pueden ser honestos o deshonestos. Este protocolo logra BFT si cumple dos requisitos:

- **Consistencia**: todos los nodos honestos i y j deben de tener el mismo valor: $m_i = m_j$.
- **Validez**: si el emisor es honesto, todos los nodos honestos i deben de tener el mismo valor: $m = m_i$.

Un sistema puede ser consistente pero no válido, cuando todos los nodos honestos tienen el mismo valor de salida pero no es el mismo que el del emisor. Un sistema puede ser válido pero no consistente, cuando el emisor no es honesto y transmite valores distintos a los nodos. Por tanto, ninguno de los requisitos es condición suficiente.

Una *blockchain* es un sistema BFT. Para solventar las inconsistencias debido al trabajo autónomo e independiente de los nodos, la solución estándar es que cada nodo esté de acuerdo en que la copia más larga de la *blockchain*, es decir, la que tiene más bloques, es la versión globalmente correcta. Debido a que las copias más cortas que la correcta no se usan, los nodos quieren tener sus copias lo más actualizadas posibles ya que si no podrían malgastar esfuerzos en añadir bloques a la *blockchain* equivocada. Como consecuencia, aunque a veces algunas transacciones sean registradas en distintas copias de la *blockchain* en nodos distintos, varias transacciones pueden ser añadidos al último bloque de la *blockchain* existente en distintos nodos, o aunque

distintos nodos pueden tener copias distintas de la *blockchain*, al final estos nodos tendrán la misma copia de la *blockchain*.

Sin embargo esto es solo en teoría. Si un consenso ocurre demasiado tarde, las inconsistencias mencionadas previamente causarán que el sistema rinda de manera incorrecta. Por lo tanto, necesitamos minimizar la probabilidad de estas inconsistencias y minimizar el tiempo que se tarda en alcanzar el consenso de la *blockchain*. Para esto, se han usado distintos mecanismos de consenso en *blockchain*. Entre los más comunes están la Prueba de Trabajo (*Proof of Work*) (PoW) o la Prueba de Participación (*Proof of Stake*) (PoS).

11.4 ALGORITMOS DE CONSENSO

Para hacer una comunicación confiable entre los nodos y mantener el estado correcto a través del sistema incluso en presencia de nodos maliciosos o incluso ante fallos de la red, se usan los conocidos algoritmos de consenso brevemente expuestos anteriormente. El algoritmo de consenso es el responsable para mantener copias consistentes del estado actual de la *blockchain* en todos los nodos, validar nuevas transacciones y actualizar el estado actual mientras se alcanza un acuerdo entre los nodos. Nos centraremos en explicar los dos algoritmos de consenso más populares, PoW y PoS.

11.4.1 *Proof Of Work*

El algoritmo de PoW normalmente necesita un nodo de prueba y otro de verificación. El nodo de prueba realiza una tarea computacionalmente intensiva para encontrar una solución a un problema de una cierta dificultad. El resultado entonces se presenta al verificador que consume significantemente menos recursos para verificar el resultado. La asimetría y la excesiva cantidad de recursos necesitados por el nodo de prueba sirve para lograr dos propósitos:

1. Mitiga ataques sibilinos a nivel de consenso. Lanzar un ataque sibilino necesita a un adversario creando múltiples entidades maliciosas o a varias entidades coordinadas entre sí. Sin embargo, por diseño, la cantidad de recursos computacionales es importante para el algoritmo de PoW, no la cantidad de nodos.
2. La carga de trabajo en sí se convierte en un seguro contra ataques de bifurcación. La longitud de la cadena es prácticamente proporcional a la cantidad de recursos gastados minando bloques. Si el adversario quiere modificar una transacción del pasado, necesita primero tener más del 51 % de los recursos computacionales dentro de la red, bifurcar una nueva cadena a partir del bloque objetivo, y minar los bloques hasta que la nueva cadena se vuelva la más larga.

Normalmente los algoritmos de PoW emplean tareas que computacionales que hacen uso de la CPU o de la GPU. La red más famosa que usa este tipo de algoritmo es la de Bitcoin. El algoritmo necesita encontrar un número llamado *nonce* tal que, cuando es *hasheado* con los contenidos del bloque (usando SHA-256d, esto es, SHA-256 aplicado dos veces), el resultado sea un número menor que un valor de dificultad. Cuando un *nonce* adecuado se encuentra y es aprobado por los verificadores, el nodo que lo ha encontrado obtiene una recompensa. El proceso de encontrar este número se llama minar, y a los nodos que lo buscan se llaman mineros.

11.4.2 *Proof Of Stake*

Para solventar las desventajas de algoritmos de PoW tales como minado centralizado y consumo energético, los algoritmos de PoS están ganando fuerza. Un algoritmo de PoS intenta validar transacciones y logra consenso en la red sin el uso de tareas computacionalmente intensas. En lugar de "trabajar", un nodo verificador tiene que bloquear su "participación", que es una parte de su riqueza en la red.

En un algoritmo de PoS, los propietarios de la participación bloquean su "participación" y se vuelven elegibles para participar en la validación de la transacción y en la creación de un nuevo bloque. Estos no minan nuevas monedas a diferencia de los algoritmos de PoW. En su lugar, recolectan las tasas de transacción o intereses proporcionales a su participación como recompensa. Si el nodo se identifica como malicioso, su participación y recompensa pueden ser confiscadas, lo que sirve como un aliciente económico para evitar nodos maliciosos.

Las ventajas de un algoritmo de PoS sobre uno de PoW son las siguientes:

- **Descentralización**: en los algoritmos de PoW los mineros pueden tener un incremento exponencial en recursos computacionales cuando se invierte en equipamiento centrado en el minado. En una red de PoS, la ganancia es directamente proporcional a la cantidad de participación que el nodo decide invertir.
- **Eficiencia energética**: los algoritmos de PoW consumen mucha energía. En el caso de Bitcoin, sus mineros consumen más energía que países como Portugal, Singapur o la República Checa. Los algoritmos de PoS no necesitan que los nodos resuelvan tareas computacionalmente intensas cuando se crean nuevos bloques. Por lo tanto pueden ser altamente eficientes a nivel energético.

Parte IV

PROPUESTA: KRUM FEDERATED CHAIN

12

BLOCKCHAIN APLICADO A FEDERATED LEARNING

Debido a las propiedades de la tecnología *blockchain* han sido múltiples los trabajos para combinar FL y esta tecnología [45] con el fin de resolver problemas causados mayormente por la centralización en FL. La combinación de *blockchain* y FL es ampliamente usada en múltiples campos [14, 38] y ha mostrado un gran éxito que ha llevado a muchas propuestas. Aún así, como veremos, es un campo todavía muy reciente y por tanto sin ningún estándar establecido al que ceñirnos.

12.1 BLOCKCHAIN COMO SOLUCIÓN A PROBLEMAS DEL FEDERATED LEARNING

Una de las principales aplicaciones de la tecnología *blockchain* en FL es la propuesta de soluciones a algunos problemas derivados del aprendizaje distribuido en FL. A continuación exponemos algunos de estos problemas vigentes en esquemas federados, muchos de ellos causados por la centralización de los esquemas clásicos de FL y ofrecemos una breve vista sobre como el uso de tecnología *blockchain* nos puede ayudar a solventar estos problemas o, al menos, mejorar las soluciones actuales.

12.1.1 *Falta de incentivos*

Cuando un cliente participa en una tarea de FL, consume una cantidad de recursos, incluyendo computación, ancho de banda de la red y batería. Además son varios los riesgos de seguridad que suele llevar el esquema federado como hemos visto en secciones anteriores. Esto hace que en algunos casos los clientes no quieran formar parte de la tarea de FL a no ser que obtengan un beneficio.

Aunque los mecanismos de incentivo son un campo de investigación activo en FL [37, 40], la mayoría de soluciones propuestas necesitan una entidad central en la que se pueda confiar para observar el comportamiento de los clientes y dar sus recompensas. Este problema proviene de la centralización del esquema de FL y por tanto es un buen candidato a ser resuelto mediante el uso de *blockchain*. El problema se puede resolver haciendo uso de los mecanismos de recompensa económicos que integran múltiples

redes *blockchain*. Por ejemplo, Zhao et al. [43] propuso un sistema de reputación basado en *blockchain* para FL para dispositivos IoT con el fin de entrenar modelos en los datos de los clientes.

12.1.2 *Heterogeneidad estadística*

La heterogeneidad estadística se refiere a las distintas distribuciones de datos de los clientes. Esto aumenta de manera drástica la complejidad del modelado del problema, análisis teórico y evaluación empírica de las soluciones. Por una parte, es difícil modelar los datos heterogéneos cuando se entrenan modelos federados en conjuntos no idénticamente distribuidos en los clientes. Por otro lado, no es fácil analizar la convergencia en procesos de entrenamiento en los que los datos son heterogéneos. En *blockchain* se ha trabajado en algoritmos de optimización distribuidos logrando resolver las limitaciones de las soluciones existentes. Por ejemplo, el algoritmo de promedio con pesos según las distancias basado en *blockchain* propuesto por Zhang et al. [42] ha demostrado tener una buena precisión a la hora de tratar el problema de heterogeneidad de datos en FL. Además, se pueden desplegar redes *blockchain*.

12.1.3 *Heterogeneidad de sistemas*

La heterogeneidad de sistemas se refiere a las diferencias sustanciales en capacidad de computación, ancho de banda de red, batería, capacidad de almacenamiento, etc., de los dispositivos en la red de FL. Además, las restricciones del tamaño de la red de cada dispositivo o a nivel de sistema normalmente solo soportan un pequeño porcentaje de estos estando activos inmediatamente. Por lo tanto, una arquitectura de FL debe de tolerar *hardware* heterogéneo y ser lo suficientemente robusta como para descartar dispositivos en la red de comunicación. El uso de tecnología *blockchain* también puede ayudar a abordar este problema. Un ejemplo es la propuesta de Zhang et al. [42] de un sistema de FL basado en *blockchain* para la detección de errores en dispositivos industriales IoT, el cual puede lograr la integridad verificable de los datos de los clientes.

12.1.4 *Seguridad del modelo*

Como se mostró en la Sección 10.7 un esquema de FL no es totalmente seguro y pueden participar clientes maliciosos con el fin de destruir los modelos finales. Además los modelos pueden ser filtrados por servidores no fiables llevando así también a problemas de privacidad. Las soluciones actuales no pueden eliminar la posibilidad de que un servidor central robe datos o modifique el modelo para dañarlo [45]. Los esquemas basados en *blockchain* no requieren de un servidor central para la agregación

del modelo lo que reduce riesgos en el sistema distribuido. Además, la verificación de identidad, trazabilidad, durabilidad, anonimidad y alta escabilidad de esta tecnología también permite garantizar la seguridad del modelo.

12.1.5 *Privacidad de los datos*

Aunque en FL se compartan modelos y no datos, la comunicación de las actualizaciones del modelo entre clientes y servidor permite la extracción de información sensible a terceros o al servidor central. Además, algunos clientes maliciosos también pueden inferir información a partir de parámetros compartidos. Algunos métodos actuales pasan por el uso de DP o métodos de cifrado como vimos en la Sección 10.8.2, aunque tienen el problema de perjudicar el rendimiento del modelo. En métodos de *blockchain* se ha propuesto con éxito [21] la incorporación de DP local a nivel de cliente. También la seguridad criptográfica de la *blockchain* permite el cifrado del modelo global en la red dejando solo a los clientes con la clave para descifrar este, evitando así ataques externos.

12.1.6 *Coste de las comunicaciones*

Dado que el cómputo de entrenamiento está repartido entre dispositivos conectados a través de internet, las comunicaciones son un cuello de botella en una red federada. Una red federada puede contener gran cantidad de dispositivos diferentes, como millones de teléfonos móviles. Debido a los recursos limitados, como el bando de ancha de la red, energía o potencia, la comunicación en la red puede ser incluso más lenta que el cómputo local. Por lo tanto, es necesario desarrollar un método de comunicación eficaz. Se ha probado que usando un mecanismo de consenso adecuado y ajustando los parámetros básicos de la *blockchain* se puede solventar este problema. Por ejemplo, en el trabajo propuesto por Kim et al. [14] el sistema optimiza la latencia de fin a fin del modelo ajustando la tasa de generación de bloques de la red. Además, también propusieron un método para calcular la latencia mediante el ajuste de la dificultad en PoW. Majeed and Hong [19] han empleado una *blockchain* con permisos mejorando la eficiencia de transmisión significantemente.

12.2 MECANISMOS DE CONSENSO EN BLOCKCHAIN APLICADO A FEDERADO

Los mecanismos de consenso son un elemento clave de la arquitectura de la *blockchain*. Aunque PoW es todavía el principal método usado en este campo y el más estudiado en la literatura, tanto a nivel teórico como de propuestas de arquitecturas, no se considera práctico desplegar PoW directamente en un sistema de FL por motivos de eficiencia. Por lo tanto, esto ha llevado al diseño de varios métodos de consenso. Al-

gunas propuestas usan PoS como mecanismo de consenso [1, 27] debido a su bajo consumo energético aunque requieren el uso de algún mecanismo de incentivo económico, por lo que su uso es restringido. Una de las propuestas más interesantes es PoFL [28].

El mecanismo de consenso PoFL [28] fue desarrollado como solución eficiente a nivel energético para la integración de la tecnología blockchain con FL. Inspirado por el concepto de los mecanismos de consenso *Proof-of-Useful Work* [31], PoFL mantiene la idea fundamental de PoW: hacer que los participantes resuelvan problemas computacionalmente intensivos para lograr un consenso. Sin embargo, a diferencia de PoW, que utiliza la potencia computacional para tareas sin ningún valor inherente (por ejemplo, encontrar un *nonce* en la red de Bitcoin [5]), PoFL decide usar estos recursos para entrenar un modelo federado.

12.2.1 *El mecanismo de consenso PoFL*

Es importante saber que PoFL hace uso una arquitectura de *pooled-mining*. En esta estructura, los participantes de la red son divididos entre diferentes *pools*, cada una supervisada por un minero asignado. Estas *pools* operan de manera independiente, entrenando sus respectivos modelos federados sin ninguna comunicación entre *pools*. La *pool* que entrene el modelo con mayor precisión en un conjunto de datos de prueba previamente determinado será considerada la ganadora en la ronda de consenso. Como consecuencia, el modelo ganador se integra en la blockchain y es retransmitido a través de toda la red a los clientes. El proceso dentro de cada *pool* se ilustra en la figura 16:

1. El minero difunde el modelo inicial a los clientes.
2. Los clientes entrenan su copia local del modelo usando sus datos privados. Este paso incluye el calcular las actualizaciones locales basadas en la diferencia entre el modelo recibido y el modelo entrenado.
3. Los clientes mandan las actualizaciones locales al minero. El minero agrega estas actualizaciones para crear un nuevo modelo mejorado. El minero evalúa el rendimiento del modelo agregado contra una meta de rendimiento previamente establecida. Si se alcanza esta meta, se llega a consenso en esa ronda.
4. Después de llegar al consenso, el minero añade el modelo mejorado a la blockchain y lo retransmite a todos los mineros, y por tanto a todos los clientes participando. Si no se llega a consenso (no se consigue la meta), el proceso se reinicia desde el paso 2, con los clientes entrenando hasta que se alcance la meta deseada.

Formalmente, podemos modelar una red de blockchain usando PoFL como un conjunto de *pools* $\{P_1, P_2, \ldots, P_n\}$, donde cada *pool* P_i contiene un minero m_i y un conjunto de clientes $\{C_1^i, \ldots, C_{j_i}^i\}$. En cada ronda de aprendizaje t, cada *pool* P_i computa un modelo federado denotado por L_i^t mediante un proceso de entrenamiento federado coor-

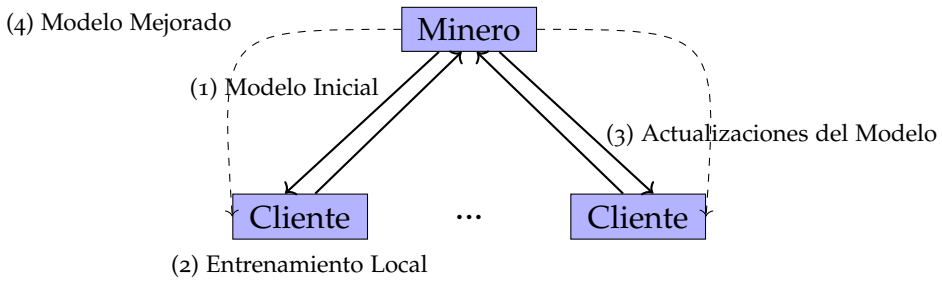


Figura 16: Diagrama mostrando como funciona el procedimiento de *pooled-mining*. Fuente: Elaboración propia.

dinado por e minero m_i. Entonces, cada minero m_i calcula la precisión de su modelo L_i^t en un conjunto de datos determinado $\mathcal{D}$, que denotaremos por $accuracy(L_i^t, \mathcal{D})$. El minero m_i* ganará el consenso si

$$accuracy(L_i^t, \mathcal{D}) \leq accuracy(L_{i*}^t, \mathcal{D}) \quad \forall i \in \{1, \ldots, n\}. \tag{127}$$

Después de llegar a consenso, el modelo L_{i*}^t es difundido a todos los mineros m_j con $j \neq i*$ los cuales comprobarán que la precisión calculada es correcta. Después de una verificación correcta, el minero m_{i*} creará un nuevo bloque b_t en que se establecerá $G^{t+1} = L_{i*}^t$ y la añadirá a la blockchain. Finalmente, todos los mineros descargarán la nueva versión de la blockchain y difundirán el nuevo modelo global G^{t+1} a todos sus clientes, asegurándose así que todos los clientes tienen acceso al último modelo.

Aunque son varios los mecanismos propuestos la mayoría de estudios no están dedicados a escenarios de FL por lo que su implementación puede no ser idónea para nuestro caso. Actualmente existe una alta demanda para desarrollar mecanismos de consenso eficientes a nivel energético para la redes *blockchain* en FL.

12.3 ARQUITECTURAS EN BLOCKCHAIN APLICADO A FL

Debido a la amplia cantidad de arquitecturas propuestas, no han sido pocos los trabajos que han intentado categorizarlas. Una de las categorizaciones más interesantes es la de Zhu et al. [45] que propone categorizar las arquitecturas según el conjunto de mineros y clientes de la red en tres tipos: desacoplado, acoplado y superpuesto. En lo que sigue M hará referencia al conjunto de los mineros de la red y C al conjunto de los clientes de la red.

12.3.1 *Modelo Desacoplado*

Se dice que nuestra red sigue un modelo desacoplado cuando dado un nodo, este solo trabaja en un esquema federado o en *blockchain*, pero nunca en ambos. Formalmente

se puede expresar como $C \cap M = \emptyset$. De este modo, los los clientes únicamente se encargarán de entrenar los modelos locales y de mandar las actualizaciones a los mineros. Estos últimos serán los encargados agregar los modelos, ejecutar el mecanismo de consenso, validar transacciones y escribir en la *blockchain*.

Este modelo tiene múltiples ventajas como la facilidad a la hora de su implementación o bajos requisitos energéticos para muchos nodos dado que no tienen que ejecutar el mecanismo de consenso todos. Sin embargo sí implica un mayor coste en comunicaciones, dificultad a la hora de implementar mecanismos de incentivo o seleccionar clientes adecuados.

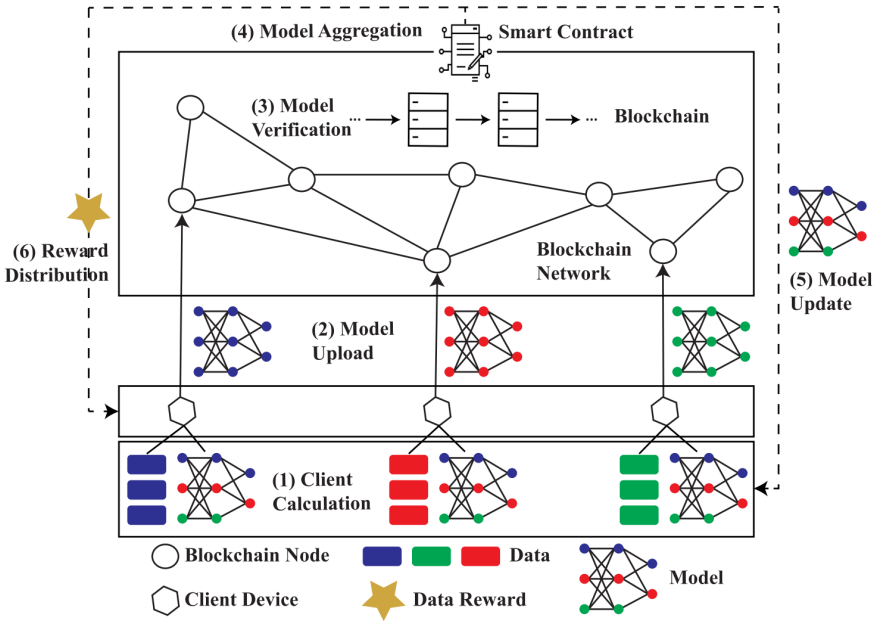


Figura 17: Esquema de una arquitectura según el modelo desacoplado. Fuente: [45].

12.3.2 *Modelo Acoplado*

En el modelo acoplado, todos los nodos son responsables de participar en el esquema de FL y de *blockchain*. Formalmente se puede expresar como $C = M$. En este modelo, los nodos se pueden llamar nodos compuestos y realizan tanto el entrenamiento local como la agregación, generación de transacciones y verificación, sumando a esto la generación de bloques y su validación.

Esto lleva a que los nodos tengan pocos costes de comunicaciones debido a la topología de la red. Además debido a la integración total de la *blockchain* y el esquema federado, el modelo acoplado es también simple de configurar y de diseñar. Sin embargo, cada nodo realiza tareas intensivas a la hora de entrenar el modelo y de ejecutar el mecanismo de consenso. Otro problema es que cada nodo en el esquema tiene más

responsabilidades que en otros esquemas, creando así más preocupaciones a nivel de privacidad de datos o seguridad del modelo.

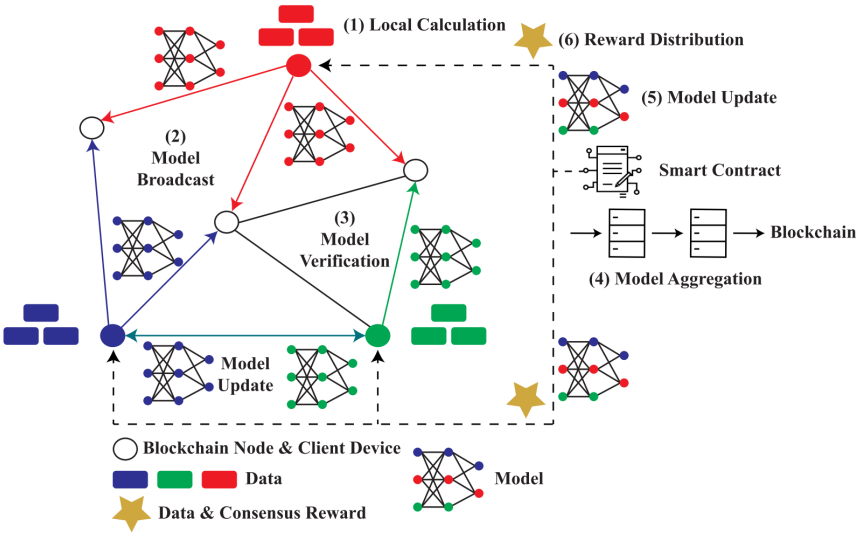


Figura 18: Esquema de una arquitectura según el modelo acoplado. Fuente: [45].

12.3.3 *Modelo superpuesto*

En el modelo superpuesto encontramos tres tipos de nodos: nodos de *blockchain*, nodos de FL y nodos compuestos. Formalmente lo podemos expresar como $M \cap C \neq \emptyset, M \neq C$. Pese a las distintas funciones, un nodo puede cambiar su tipo a lo largo del tiempo. El modelo superpuesto equilibra el modelo acoplado y desacoplado y toma la ventajas de ambos. Sin embargo esto se ve reflejado en complejidad a la hora de configurar la red.

La asignación de las funciones de los nodos se puede optimizar basado en los recursos disponibles, nivel de seguridad, etc. Sin embargo, a parte de las desventajas ya mencionadas, el modelo superpuesto tiene complicaciones a la hora de diseñar mecanismos de consenso e incentivo. Esto se debe a la dificultad intrínseca de la red.

Como veremos más adelante, la mayoría de propuestas para combinar FL y *blockchain* consistirán en modelos acoplados o desacoplados que seguirán un estilo similar al de una red como *Bitcoin*. Estas redes cuentan con lo que se conoce como mecanismo de *Gossip*. Este mecanismo es un protocolo que permite a los mineros intercambiar información con el fin de que un minero m_i pueda publicar en un bloque las transacciones recibidas o generadas por el minero m_j. Es por ello que la mayoría de arquitecturas se pueden ver como un esquema federado centralizado en el que en cada ronda t el agregador central es un minero m_{i_t}. Este acercamiento nos permite beneficiarnos de

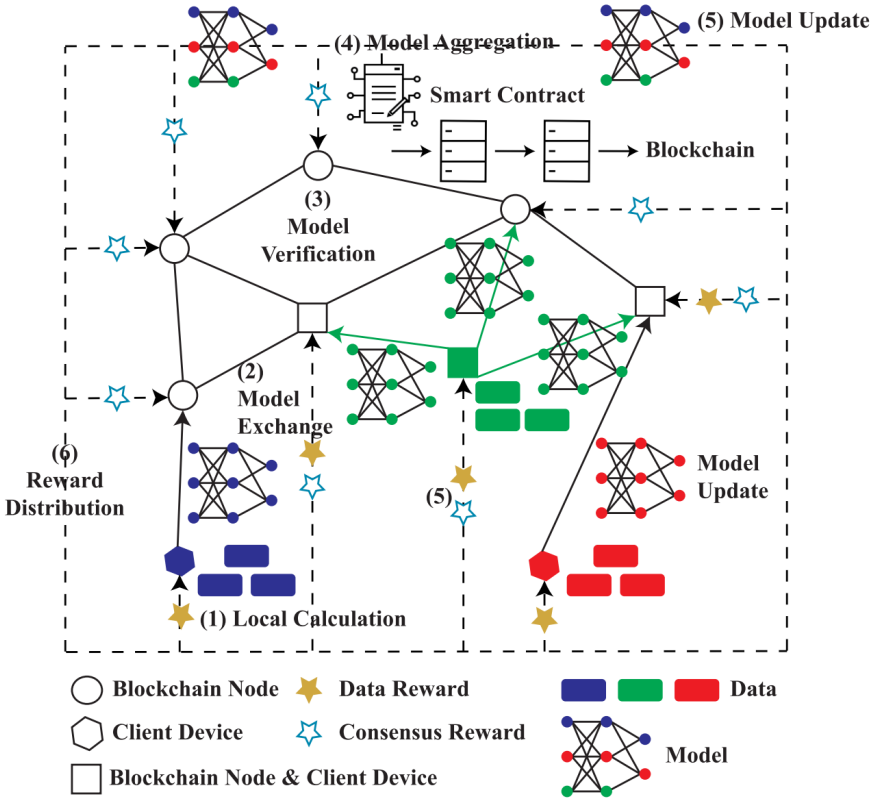


Figura 19: Esquema de una arquitectura según el modelo superpuesto. Fuente: [45].

todas las ventajas expuestas al principio del capítulo y con un diseño sencillo a nivel de red.

13

PROPUESTA: KRUM FEDERATED CHAIN (KFC)

Una vez introducidos todos los conceptos necesarios, en esta sección exponemos nuestra propuesta de mecanismo de defensa frente a ataques adversarios en FL llamada KFC. La propuesta parte de la hipótesis de que el mecanismo de consenso PoFL puede ser una defensa contra ataques adversarios en FL. Además, de las vulnerabilidades identificadas respecto a la configuración de los ataques, proponemos KFC, una defensa basada en la combinación de Krum y blockchain con el mecanismo de consenso PoFL para defender esquemas de FL contra cualquier configuración de ataques adversarios.

13.1 HIPÓTESIS: POFL COMO MECANISMO DE DEFENSA

Se puede apreciar que el mecanismo de consenso PoFL exhibe una potencial defensa contra ataques adversarios. Los ataques bizantinos consisten en perjudicar el rendimiento del modelo federado y los ataques de *backdoor* involucran a un adversario dentro de una *pool* intentando manipular el proceso de entrenamiento del modelo para introducir una tarea secundaria. En el segundo caso, el hecho de estar optimizando varios objetivos pueden llevar a una ligera degradación en el rendimiento del modelo en la tarea original.

Mediante el uso de un mecanismo de consenso basado en el rendimiento del modelo y *pooled-mining*, PoFL desalienta estos tipos de ataques. Las *pools* que contengan a clientes adversarios son probables de producir modelos con una precisión inferior en el conjunto de datos preestablecido en comparación con las *pools* sin clientes maliciosos. Como consecuencia, el mecanismo de consenso PoFL favorecería a las *pools* con el modelo con mejor rendimiento, filtrando de manera eficaz al modelo con la tarea inyectada y por tanto mitigando el ataque.

13.2 KRUM FEDERATED CHAIN

Es importante saber que la potencial resistencia de PoFL contra ataques adversarios depende de la condición de que siempre exista una *pool* sin ningún cliente malicioso en ella. Esta condición puede ser considerada demasiado optimista y exigente para el mundo real. Por lo tanto, proponemos la arquitectura KFC como una alternativa a PoFL con una seguridad más robusta. Aunque KFC mantiene la misma arquitectura fundamental y principios de eficiencia energética que PoFL, incorpora mecanismos de seguridad adicionales para mitigar ataques adversarios incluso en la presencia de actores maliciosos en todas las *pools* de la red.

KFC hace uso del operador de agregación Krum [2], expuesto previamente en la sección 7, para mejorar su resistencia contra ataques al modelo federado. Recordamos que este operador funciona ordenando actualizaciones de clientes según la distancia geométrica entre las distribuciones de actualizaciones de modelo respectivas. Para ello y recuperando la notación de la Sección 7, a cada cliente i se le asigna la puntación

$$s(i) = \sum_{i \to j} ||V_i - V_j||^2 \tag{128}$$

donde $i \to j$ denotaba el hecho de que el vector V_j pertenecía al conjunto de los $n - f - 2$ vectores más cercanos al vector V_i, con $i \neq j$. A continuación, selecciona la actualización más cercana a la mayoría, lo que efectivamente filtra a los outliers. Esto es, el vector V resultado de la regla de agregación $KR(V_1, \ldots, V_n)$ se define como aquel vector V_{i_*} tal que

$$s(i_*) \leq s(i) \quad \forall i \in \{1, \ldots, n\}. \tag{129}$$

Los clientes adversarios que intentan manipular el modelo tienen más probabilidades de generar actualizaciones que se desvían significativamente de la norma, lo que los hace más propensos a ser identificados y excluidos por el operador Krum.

Así, KFC consiste en la combinación de la arquitectura base de PoFL con la regla de agregación Krum para reforzar la seguridad a nivel de *pool*. Esta decisión de diseño permite a KFC verse beneficiado por la resistencia a ataques adversarios innata de PoFL debido al uso de *pooled-mining* y su mecanismo de consenso. Además, Krum refuerza esta resistencia mediante el filtrado de *outliers* dentro de las distribuciones de las actualizaciones dentro de cada *pool*. Por lo tanto, no hace falta mantener la hipótesis de la existencia de una *pool* sin ningún tipo de actor malicioso. Este acercamiento combinado hace uso de distintos tipos de mecanismos de defensa en distintas partes del esquema federado, llegando así a una solución más robusta para FL bajo condiciones adversarias.

14

ENTORNO EXPERIMENTAL

Con el fin de evaluar y medir el rendimiento de KFC y PoFL, hemos calculado la precisión o *accuracy* del modelo federado resultante en múltiples conjuntos de datos expuestos en la Sección 14.1 donde el objetivo era resolver un problema de clasificación de imágenes. Para esto hemos desplegado modelos basados en CNN en un entorno de FL explicados en la Sección 14.2. Finalmente, explicamos los ataques cubiertos en la Sección 14.3, los escenarios en los que se realizan en la Sección 14.4 y las métricas usadas para la evaluación en la Sección 14.5.

14.1 CONJUNTOS DE DATOS PARA LA EVALUACIÓN

Dado que PoFL, y por tanto KFC, necesitan un conjunto de validación con el fin de medir la precisión de cada modelo de la red, usamos el 20 % del conjunto de test de cada conjunto de datos con este fin dejando el 80 % restante como conjunto de test. Hemos usado tres conjuntos de datos en la evaluación, los cuales describimos a continuación:

1. El conjunto de datos EMNIST (*Extended Modified NIST*), presentado en 2017 en [4], es una extensión del famoso conjunto de datos MNIST [16]. La clase *EMNIST Digits* contiene un subconjunto balanceado del conjunto de dígitos. Esta contiene 28,000 muestras de cada dígito. Por lo tanto, el conjunto total consiste en 280,000 muestras, de las cuales 240,000 son usadas para el entrenamiento del modelo y 40,000 como muestras de test (usamos 8,000 como muestras de validación y las 32,000 restantes para test). Para los experimentos hemos decidido establecer el número de clientes en 200 y el de mineros en 3.
2. El conjunto de datos Fashion MNIST [39] fue diseñado para ser un reemplazo más exigente del conjunto original MNIST. Contiene un conjunto de 10 clases distintas extraídas del conjunto de imágenes de Zalando con 7,000 muestras de cada clase. Así, el conjunto de datos consiste de 70,000 muestras, de las cuales 60,000 son usadas para entrenamiento y 10,000 para test (usamos 2,000 para validación y las 8,000 restantes para test). Hemos establecido el número de clientes a 200 y el número de mineros a 3.

	EMNIST	Fashion MNIST	CIFAR-10
Train	240,000	60,000	50,000
Test	32,000	8,000	8,000
Validation	8,000	2,000	2,000

Tabla 1: Cantidad de muestras en cada conjunto distribuidas según su uso.

3. El conjunto de datos CIFAR-10 es un subconjunto etiquetado del conjunto *80 million tiny images* [35]. Consta de 60,000 imágenes a color de resolución 32x32 de 10 clases, es decir, con 6,000 imágenes por clase. 50,000 imágenes son usadas para entrenamiento y 10,000 como validación, lo que corresponde a 1,000 muestras por clase (usamos 2,000 para validación y las 8,000 restantes para test). Establecemos el número de clientes a 100 (con el fin de tener más datos por cliente pues el modelo a entrenar es mucho más complejo) y el de mineros a 2.

14.2 MODELOS DE CLASIFICACIÓN DE IMÁGENES

Dado que todos los problemas que abordamos son problemas de clasificación, y que el objetivo principal de la propuesta no es maximizar el rendimiento si no probar mecanismos de defensa, usamos modelos de clasificación que obtengan buen rendimiento en estos problemas sin ataques. Para ello hemos considerado dos modelos:

- En los conjuntos EMNIST y Fashion MNIST hemos usado una red CNN estándar compuesta de dos capas convolucionales seguida cada una por una capas de *max-pooling*, y un clasificador compuesto de dos capas densas con funciones de activación ReLU.
- En el conjunto más exigente CIFAR-10, hemos decidido usar una red de la familia ya expuesta previamente EfficientNet, concretamente una `EfficientNet-B0`. Hemos realizado un *fine-tuning* usando unos pesos preentrenados de la red en el conjunto *ImageNet*.

14.3 ATAQUES ADVERSARIOS

A continuación se detallan los ataques implementados, basados en los ya introducidos en la Sección 10.7.3. Con el fin de que las actualizaciones de los clientes adversarios no sean mitigadas al ser agregadas debido a la cantidad de clientes benignos, usamos técnicas de *model replacement* para aumentar la eficacia del ataque.

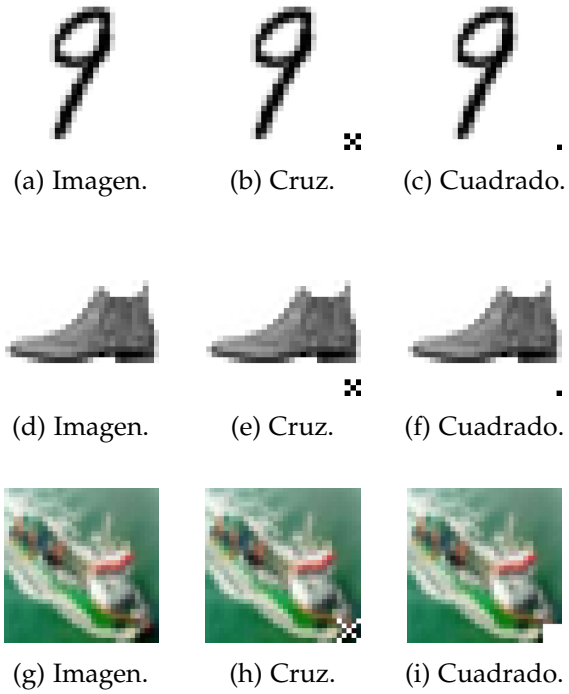
14.3.1 *Ataques bizantinos*

Implementamos un ataque bizantino basado en *random label flipping* [29] consistente en cambiar de forma aleatoria las etiquetas de todas las muestras del cliente adversario a una etiqueta distinta de la original. El único objetivo de este ataque es el de perjudicar el rendimiento del modelo federado confundiendo al modelo con las muestras mal etiquetadas.

Figura 20: Ejemplo de ataque *random label flipping* en el conjunto de datos EMNIST. Las etiquetas de las muestras son cambiadas por una nueva etiqueta aleatoria manteniendo la muestra original. Fuente: Elaboración propia.

14.3.2 *Ataques backdoor basado en patrones*

También realizamos un ataque de *backdoor* basado en patrones, en el que todos los clientes conocen el patrón completo y lo usan en su proceso de entrenamiento. Establecemos una etiqueta objetivo y un patrón. El número de tareas de *backdoor* es el número de clientes adversarios. Con el fin de demostrar que el comportamiento observado no depende del patrón usado, usamos dos patrones: (1) un píxel blanco en la esquina inferior izquierda para EMNIST y Fashion MNIST o un cuadrado blanco de 5x5 en CIFAR-10, y (2) una cruz blanca de longitud 3 para EMNIST y Fashion MNIST y de longitud 5 para CIFAR-10. La diferencia de tamaño en los patrones se debe al uso del modelo de *EfficientNet* pues este usa un recorte central en la imagen. Se pueden ver los patrones en la Figura 21.



(a) Imagen. (b) Cruz. (c) Cuadrado.

(d) Imagen. (e) Cruz. (f) Cuadrado.

(g) Imagen. (h) Cruz. (i) Cuadrado.

Figura 21: Ejemplos de muestras originales (a, d y g) y envenenadas con patrón de cruz (b, e y h), y de cuadrado (c, f and i) en los conjuntos de datos EMNIST, Fashion-MNIST y CIFAR, respectivamente.

14.4 ESCENARIOS DE LOS ATAQUES

Con el objetivo de poner a prueba nuestras propuestas en distintos entornos consideramos dos escenarios distintos:

1. **Escenario A:** Solo hay un único cliente adversario participando en una ronda de aprendizaje t.
2. **Escenario B:** La cantidad de clientes adversarios en una ronda de aprendizaje se establece a la cantidad de mineros en la red. En el caso de que la arquitectura a comprobar haga uso de *pooled-mining*, esto es, a cada minero se le asigna un subconjunto de los clientes y no hace uso de ningún tipo de recurso del subconjunto de otro minero, nos aseguramos que cada subconjunto contenga un cliente adversario.

14.5 MÉTRICAS USADAS PARA LA EVALUACIÓN

Consideraremos las métricas según el escenario que estemos cubriendo:

14.5.1 *Métricas en el caso sin atacantes*

En esta situación nos encontramos ante un caso genérico de clasificación de imágenes. Por lo tanto, usaremos la precisión o *accuracy* en el conjunto de test de nuestro conjunto de datos para medir el rendimiento del modelo.

14.5.2 *Métricas para ataques bizantinos*

Cuando medimos una defensa contra un ataque bizantino, el objetivo es obtener el mejor rendimiento del modelo en la tarea planteada. Es por ello que comprobamos la precisión o *accuracy* del modelo en el subconjunto de test del conjunto de datos que estamos considerando. Claramente, cuanto mayor sea este valor, mejor será la defensa contra este tipo de ataques.

14.5.3 *Métricas para ataques de backdoor*

A la hora de defendernos de un ataque de *backdoor*, debemos de considerar dos aspectos distintos. Estos son, el rendimiento del modelo resultante en la tarea original y en la tarea inyectada. La meta es reducir el impacto del ataque lo máximo posible sin comprometer el rendimiento del modelo en la tarea original. Para ello consideramos dos conjuntos de test:

1. **Conjunto de test original**. El conjunto de test original usado para medir el rendimiento en términos de la precisión o *accuracy* en la tarea original.
2. **Conjunto de test de *backdoor***. Este conjunto representa el ataque con el fin de medir el rendimiento de la tarea de *backdoor*. Consiste del conjunto de test original pero envenenado usando el patrón, con el fin de medir la capacidad de generalización del ataque.

Una defensa será más efectiva cuanto más consiga prevenir el ataque (menor rendimiento en el test de *backdoor*) minimizando la pérdida de rendimiento en la tarea original (mayor rendimiento en el test original).

Dado que los resultados pueden ser altamente heterogéneos en cada ronda de aprendizaje y con el fin de mostrar resultados robustos, mostramos tanto la métrica de la última ronda (*accuracy*) como la media de cada una de estas métricas en las últimas diez rondas de aprendizaje ($accuracy_{10}$). Además, dado que los resultados entre mineros de una misma red pueden diferir en el caso de PoFL y KFC, mostramos las métricas del mejor minero en términos del rendimiento en la tarea original la cual se supone que puede ser medida en un caso real mediante el conjunto de validación usado en estas arquitecturas.

14.6 ARQUITECTURAS DE REFERENCIA

Comparamos KFC y PoFL con las siguientes arquitecturas de FL, que representan las referencias clásicas y más estudiadas para *blockchain* aplicado al FL:

1. **Cliente-Servidor (C-S)**. No hace ningún uso de *blockchain*. Es uno de los escenarios de FL más comunes donde un servidor central actúa como agregador y organiza todo el proceso de aprendizaje.
2. ***Proof of Work***. El mecanismo de consenso donde los mineros compiten en una carrera computacional para resolver un puzzle. El ganador actúa como el agregador para una ronda dada. Consiste en un modelo desacoplado.
3. ***Proof of Stake***. Una arquitectura similar a la anterior pero donde se decide usar PoS como mecanismo de consenso como alternativa más eficiente a nivel energético para la red. Consiste en un modelo desacoplado.

En todas las arquitecturas de referencia anteriores y en PoFL usamos FedAvg [20] como el operador de agregación el cual es muchas veces considerado el operador de agregación por defecto para FL y el más estudiado en la literatura. Además, dado que PoW y PoS utilizan la misma arquitectura *blockchain* subyacente hemos decidido combinar los resultados de ambos con el fin de obtener una mayor comprensión y que llamaremos PoW/S.

14.7 DETALLES DE IMPLEMENTACIÓN

Los experimentos han sido realizados usando el framework de simulación de FL `flex` [12] y la librería de AA `pytorch` [25]. Con el fin de simular el comportamiento de una *blockchain* en un escenario federado se ha implementado una librería compañera a `flex` llamada `flex-block` [12][1]. Además, se adjunta el código realizado[2] para asegurar la reproducibilidad de los experimentos realizados.

1 https://github.com/FLEXible-FL/flex-block
2 https://github.com/mariogmarq/kfc-experiments

15

ANÁLISIS DE LOS RESULTADOS

En esta sección se exponen los resultados de los experimentos explicados en la Sección 14. En primer lugar, en la Sección 15.1 analizamos el funcionamiento de PoFL como mecanismo de defensa. De las debilidades detectadas en esta sección surge KFC, cuyo rendimiento como defensa analizamos, de forma comparativa con la propuesta anterior, en la sección 15.2.

15.1 ANÁLISIS DE POFL COMO MECANISMO DE DEFENSA

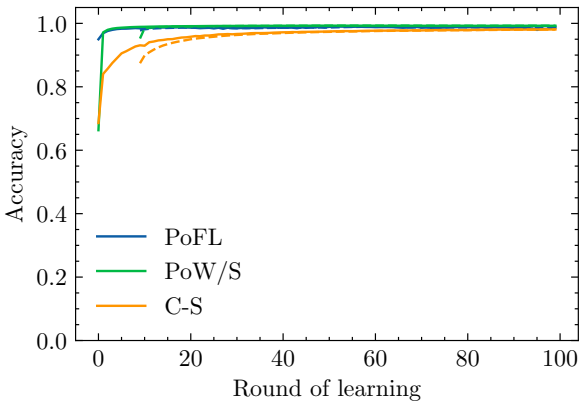
A continuación veremos el rendimiento de PoFL tanto en el caso en el que no hay ningún atacante en la red (vea Sección 15.1.1) como en los escenarios A (vea Sección 15.1.2) y B (vea Sección 15.1.3).

15.1.1 *Rendimiento sin atacantes*

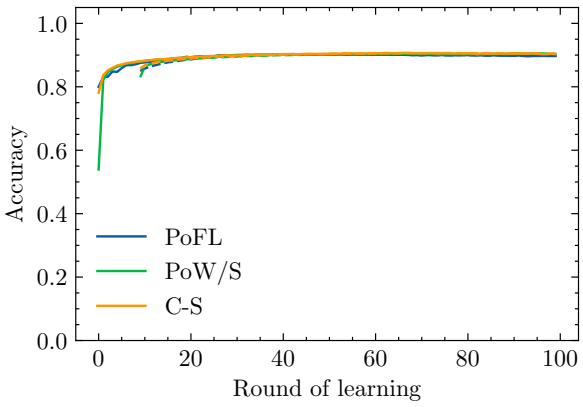
Al considerar el caso de que no haya ningún atacante en la red podemos observar tanto en la Tabla 2 como en la Figura 22 que tanto nuestra arquitecturas de referencia como aquella basada en PoFL obtienen un rendimiento en términos de precisión similar en todos los conjuntos de datos. Esto nos muestra que PoFL es una buena arquitectura para el aprendizaje y una buena alternativa a nuestras arquitecturas de referencia. Esto indica que seleccionar únicamente un modelo basándose en un criterio de eficiencia en un conjunto de validación obtiene resultados similares a los de considerar todas las actualizaciones posibles.

	accuracy			*accuracy*$_{10}$		
	C-S	**PoW/S**	**PoFL**	**C-S**	**PoW/S**	**PoFL**
EMNIST	0.9810	**0.9924**	0.9869	0.9805	**0.9924**	0.9845
Fashion MNIST	0.9030	**0.9033**	0.8972	**0.9041**	0.9040	0.8973
CIFAR-10	**0.9688**	0.9677	0.9569	**0.9686**	0.9676	0.9564

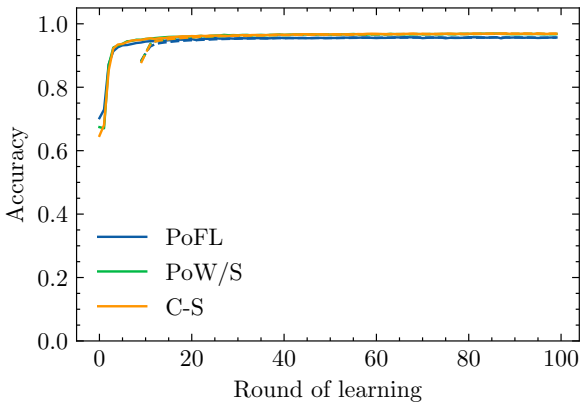
Tabla 2: *accuracy* y *accuracy*$_{10}$ comparando PoFL con las arquitecturas de referencia sin ningún atacante.



(a) EMNIST.



(b) Fashion MNIST.



(c) CIFAR.

Figura 22: *accuracy* (línea) y *accuracy*$_{10}$ (línea discontinua) de las tareas originales EMNIST (a), Fashion MNIST (b) y CIFAR (c), sin ningún atacante presente.

15.1.2 *Escenario A: Un solo atacante*

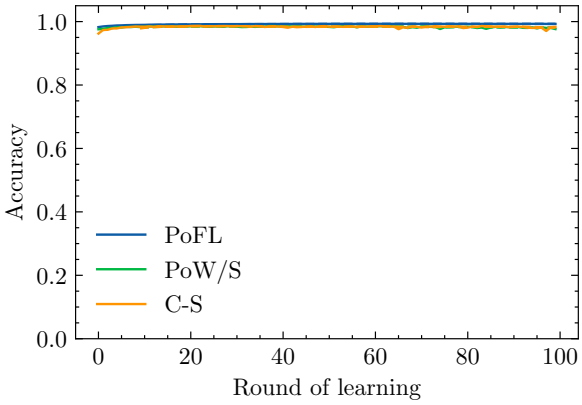
En esta sección analizamos el rendimiento de PoFL en el **escenario A**. Analizando los resultados expuestos en las Tablas 3 y 4 junto a las Figuras 23 y 24, aunque se abarcan diferentes tipos de ataques, se puede observar las siguientes conclusiones comunes:

- PoFL demuestra una superioridad en rendimiento en todos los conjuntos de datos bajo ambos ataques, obteniendo la mayor precisión durante todo el proceso de entrenamiento. Además los resultados logrados por PoFL son similares a los obtenidos para el caso donde no considerábamos atacantes, resaltando así su capacidad como mecanismo de defensa.
- Para nuestra tarea de *backdoor*, representada en la Tabla 3 y Figura 23, PoFL logra mitigar el ataque obteniendo una precisión mínima que nos sugiere la exitosa supresión de la tarea inyectada. Además, PoFL además de mitigar el ataque consigue mantener un buen rendimiento en la tarea original, asegurando tanto la mitigación de ataques como el objetivo original. Por otra parte, nuestras arquitecturas de referencia no ofrecen ningún tipo de resistencia obteniendo una precisión casi perfecta en la tarea secundaria.
- Si observamos el ataque bizantino, cuyos resultados aparecen en la Tabla 4 y Figura 24, vemos como PoFL vuelve a obtener los mismos resultados mientras que nuestras arquitecturas de referencia ven cómo su rendimiento es degradado presentando además múltiples perturbaciones en las medidas.

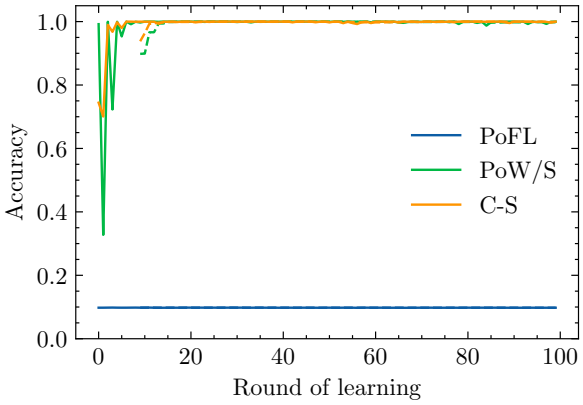
		accuracy			*accuracy*$_{10}$		
		C-S	**PoW/S**	**PoFL**	**C-S**	**PoW/S**	**PoFL**
EMNIST	Original	0.9821	0.9793	**0.9930**	0.9825	0.9809	**0.9930**
	Backdoor	0.9999	0.9994	**0.0980**	0.9997	0.9996	**0.0979**
Fashion	Original	0.8462	0.8414	**0.8961**	0.8416	0.8486	**0.8986**
MNIST	Backdoor	0.9955	0.9969	**0.0871**	0.9884	0.9878	**0.0989**
CIFAR-10	Original	0.9103	0.9038	**0.9397**	0.9021	0.8762	**0.9283**
	Backdoor	0.9676	0.9704	**0.1163**	0.9348	0.9736	**0.0849**

Tabla 3: *accuracy* y *accuracy*$_{10}$ comparando PoFL con las arquitecturas de referencia en el **escenario A** ante un ataque de backdoor.

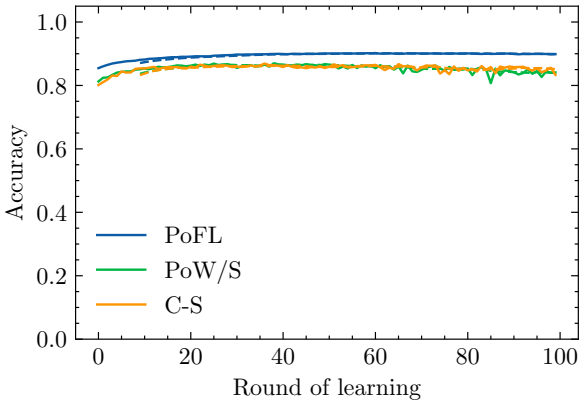
Basándonos en estas observaciones, los resultados experimentales ofrecen pruebas convincentes de que PoFL constituye una arquitectura de *blockchain* aplicada a FL robusta contra ataques de *backdoor* y bizantinos dentro del escenario investigado. Estos hallazgos confirman que el mecanismo de consenso basado en el rendimiento empleado por *PoFL*, junto con la arquitectura de *pooled-mining*, filtra de manera eficaz



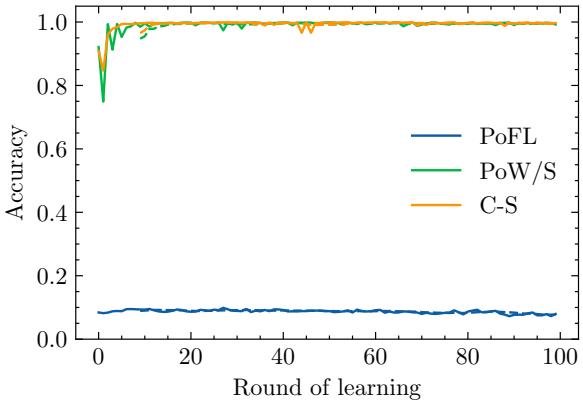
(a) EMNIST (Tarea original).



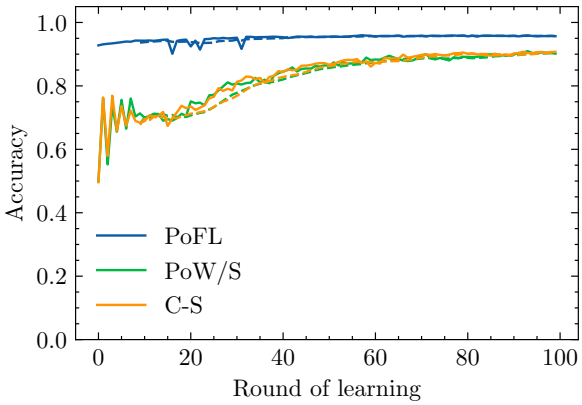
(b) EMNIST (Tarea backdoor).



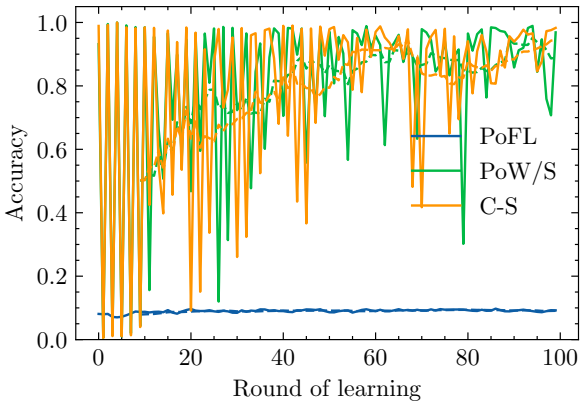
(c) Fashion EMNIST (Tarea original).



(d) Fashion EMNIST (Tarea backdoor).



(e) CIFAR (Tarea original).



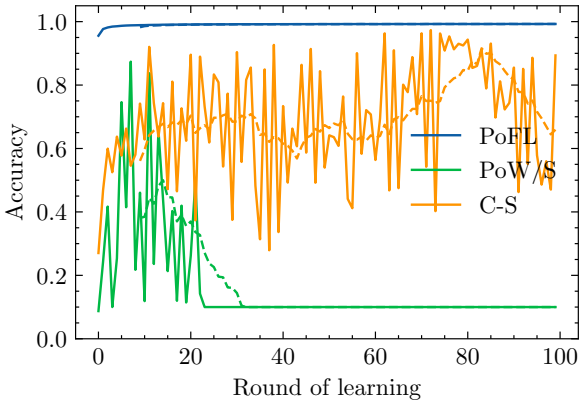
(f) CIFAR (Tarea backdoor).

Figura 23: *accuracy* (línea) y $accuracy_{10}$ (línea discontinua) de las tareas originales y backdoor en EMNIST (a y b), Fashion MNIST (c y d) y CIFAR (e y f), respectivamente, en el **escenario A**.

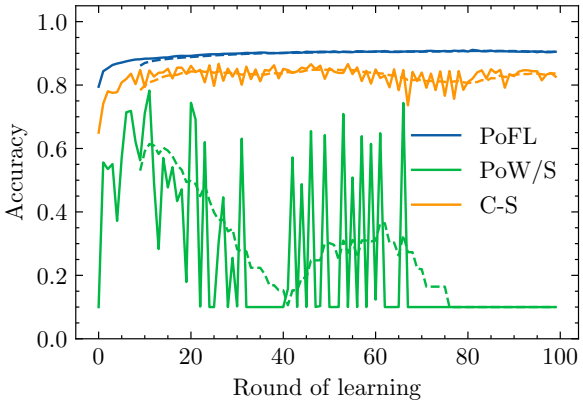
las actualizaciones maliciosas inyectadas por clientes adversarios debido a que estos tienden a crear actualizaciones con mala calidad.

	accuracy			*accuracy*$_{10}$		
	C-S	**PoW/S**	**PoFL**	**C-S**	**PoW/S**	**PoFL**
EMNIST	0.8931	0.1000	**0.9926**	0.6594	0.1030	**0.9925**
Fashion MNIST	0.8264	0.1041	**0.9051**	0.8363	0.0978	**0.9049**
CIFAR-10	0.2868	0.6870	**0.9628**	0.4196	0.6685	**0.9604**

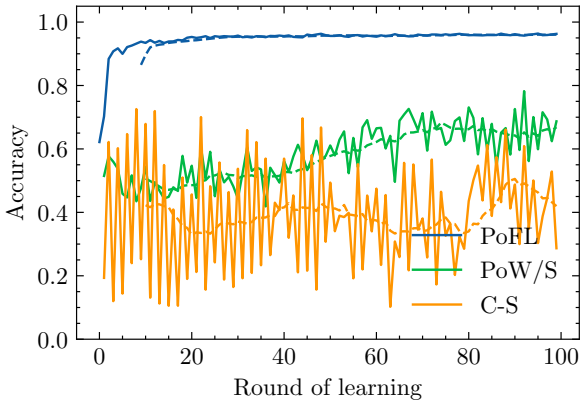
Tabla 4: *accuracy* y *accuracy*$_{10}$ comparando PoFL con las arquitecturas de referencia en el **escenario A** ante un ataque bizantino.



(a) EMNIST.



(b) Fashion EMNIST.



(c) CIFAR.

Figura 24: *accuracy* (línea) y *accuracy*$_{10}$ (línea discontinua) de las tareas originales EMNIST (a), Fashion MNIST (b) y CIFAR (c), bajo un ataque bizantino en el **escenario A**.

		accuracy			*accuracy*$_{10}$		
		C-S	**PoW/S**	**PoFL**	**C-S**	**PoW/S**	**PoFL**
EMNIST	Original	0.9863	**0.9889**	0.8986	0.9866	0.9888	**0.9108**
	Backdoor	0.9999	1.0000	**0.7536**	0.9997	1.0000	**0.8002**
Fashion MNIST	Original	0.8593	**0.8729**	0.8133	0.9997	1.0000	**0.8002**
	Backdoor	0.9955	0.9977	**0.8199**	0.9891	0.9963	**0.9181**
CIFAR-10	Original	0.9101	**0.9436**	0.8485	0.9032	**0.9402**	0.8223
	Backdoor	0.9676	0.9922	**0.8535**	0.9837	0.9928	**0.9151**

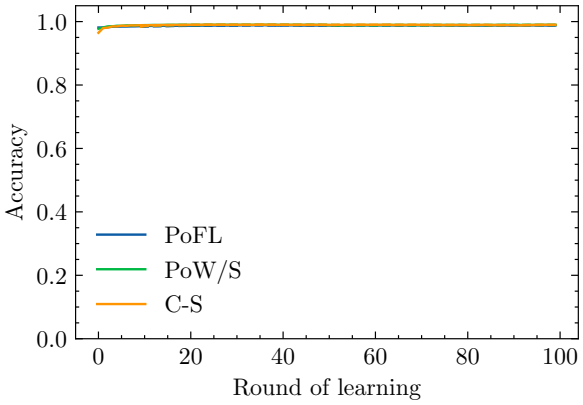
Tabla 5: *accuracy* y *accuracy*$_{10}$ comparando PoFL con las arquitecturas de referencia en el **escenario B** ante un ataque de backdoor.

15.1.3 *Escenario B: Todos los mineros son atacados*

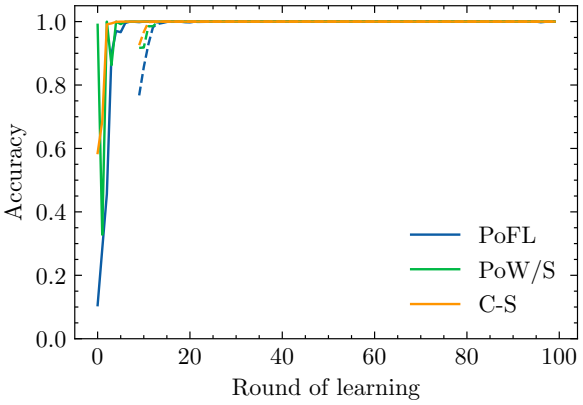
A continuación estudiamos el rendimiento de PoFL como defensa en el **escenario B**. Los resultados han sido representados tanto en las Tablas 5 y 6 como en las Figuras 25 y 26. Tras su observación llegamos a las siguientes conclusiones:

- A diferencia del escenario anterior, donde PoFL aparecía como la arquitectura con mayor rendimiento, en este caso exhibe el peor rendimiento en la mayoría de las tareas. En todas ellas muestra un comportamiento inconsistente con grandes fluctuaciones durante todo el proceso de entrenamiento. Nuestras arquitecturas de referencia, sin embargo, mantienen una precisión similar en todas las pruebas en comparación con el escenario anterior.
- En la tarea de *backdoor*, acudiendo a la Tabla 5 y Figura 25, vemos como PoFL tiene el peor rendimiento en la tarea original en los tres conjuntos. Además, aunque inferior a las arquitecturas de referencia, PoFL muestra un alto rendimiento en la tarea secundaria indicando el éxito del ataque de *backdoor* bajo estas circunstancias.
- Respecto al ataque bizantino, observando la Tabla 6 y Figura 26, vemos que el rendimiento de PoFL es similar al de las arquitecturas de referencia, denotando ser vulnerable ante este tipo de ataques.

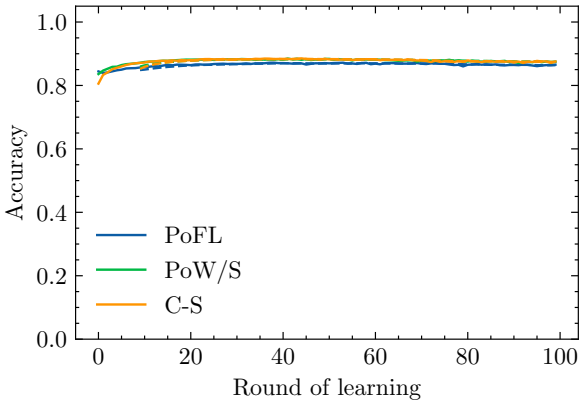
En conclusión, los experimentos muestran un panorama preocupante sobre la idoneidad de PoFL en este escenario. Como se ha demostrado en las observaciones anteriores, PoFL exhibe una vulnerabilidad significativa a los ataques de *backdoor*. No solo logra una precisión preocupantemente alta en las tareas de *backdoor* en promedio, sino que su rendimiento en la tarea original también sufre un detrimento sustancial. Además, muestra un deterioro grave ante ataques bizantinos, obteniendo resultados similares a aquellos obtenidos por nuestras arquitecturas de referencia. Esto se debe



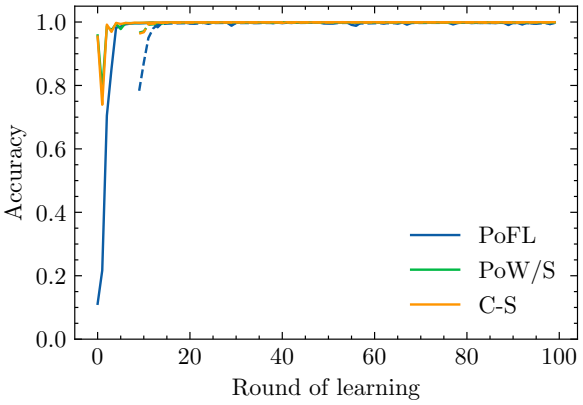
(a) EMNIST (Tarea original).



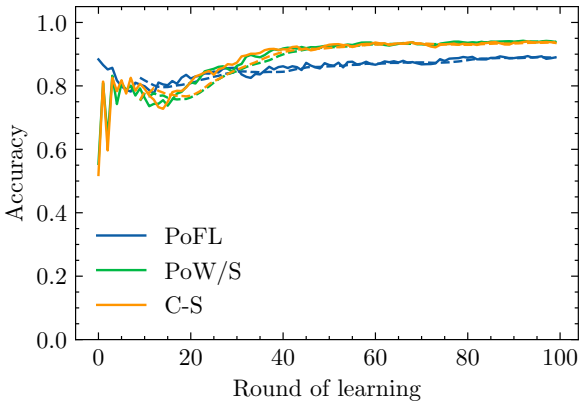
(b) EMNIST (Tarea backdoor).



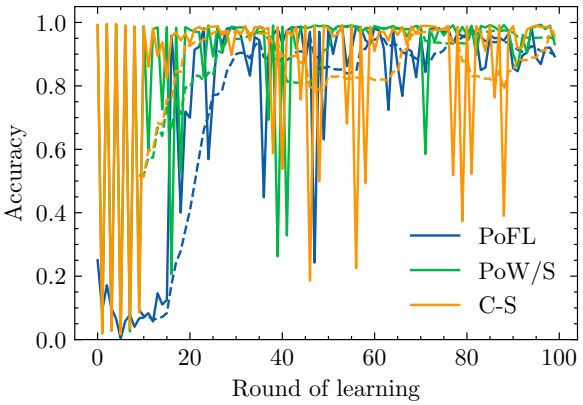
(c) Fashion EMNIST (Tarea original).



(d) Fashion EMNIST (Tarea backdoor).



(e) CIFAR (Tarea original).



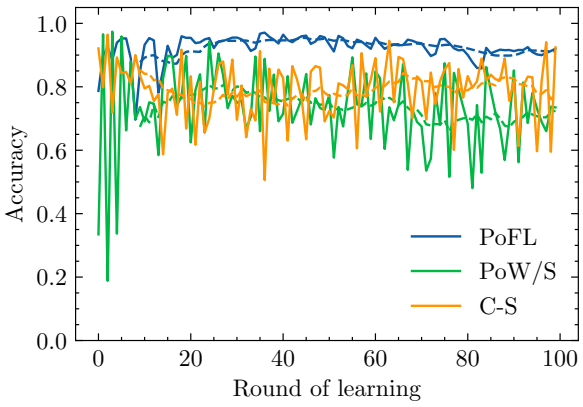
(f) CIFAR (Tarea backdoor).

Figura 25: *accuracy* (línea) y $accuracy_{10}$ (línea discontinua) de las tareas originales y backdoor en EMNIST (a y b), Fashion MNIST (c y d) y CIFAR (e y f), respectivamente, en el **escenario B**.

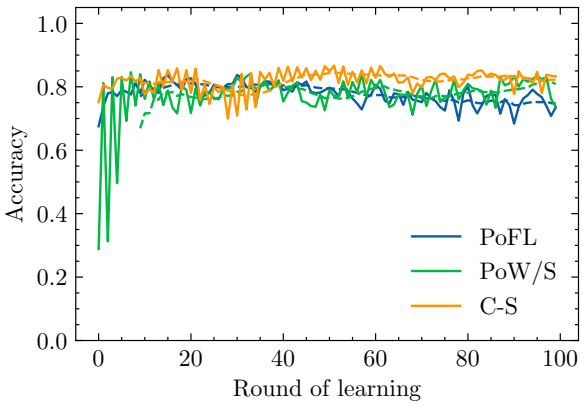
a que cuando el rendimiento de todas las *pools* se ve comprometido, el mecanismo de consenso es incapaz de filtrar las actualizaciones maliciosas.

	accuracy			*accuracy*$_{10}$		
	C-S	**PoW/S**	**PoFL**	**C-S**	**PoW/S**	**PoFL**
EMNIST	**0.9232**	0.7351	0.9202	0.7770	0.7246	**0.9120**
Fashion MNIST	**0.8329**	0.7372	0.7344	**0.8222**	0.8087	0.7476
CIFAR-10	0.6168	**0.7967**	0.5648	0.6692	**0.7545**	0.5648

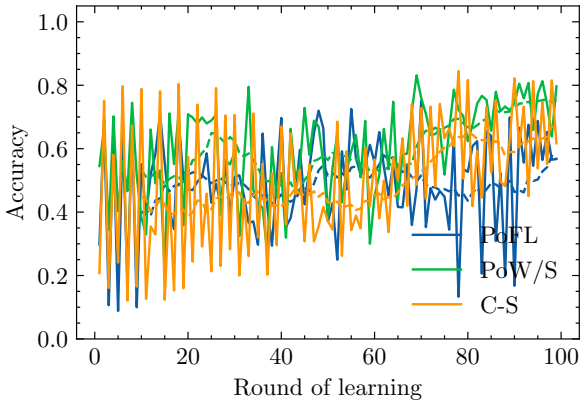
Tabla 6: *accuracy* y *accuracy*$_{10}$ comparando PoFL con las arquitecturas de referencia en el **escenario B** ante un ataque de bizantino.



(a) EMNIST.



(b) Fashion EMNIST.



(c) CIFAR.

Figura 26: *accuracy* (línea) y *accuracy*$_{10}$ (línea discontinua) de las tareas originales EMNIST (a), Fashion MNIST (b) y CIFAR (c), bajo un ataque bizantino en el **escenario B**.

Esta vulnerabilidad de PoFL motiva el diseño de KFC, como la combinación de PoFL y un mecanismo de defensa en cada minero, lo que lo hace resistente a ataques adversarios incluso cuando todos los mineros están siendo atacados. Abordamos esta cuestión en la siguiente sección.

15.2 ANÁLISIS DE RENDIMIENTO DE KFC

En esta sección analizaremos los resultados de KFC en los escenarios A (vea Sección 15.2.1) y B (vea Sección 15.2.2). Además, los compararemos con aquellos de PoFL obtenidos en la Sección 15.1.

15.2.1 *Escenario A: Un solo atacante*

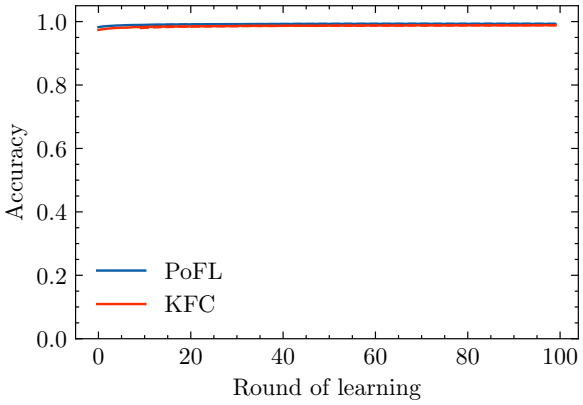
Primero nos centraremos en el **escenario A** donde PoFL ya se mostraba como una defensa válida. Los resultados expuestos en las Tablas 7 y 8 junto a las Figuras 27 y 28, nos indican que KFC también es una defensa eficaz para ambos tipos de ataques. Si observamos con detalle, vemos que el rendimiento de KFC tanto en la tarea original del ataque de *backdoor* como en el ataque bizantino es ligeramente inferior al de PoFL. Esto es un resultado esperado y conocido del mecanismo de agregación Krum [2], debido a seleccionar únicamente al cliente que minimiza la distancia geométrica, lo que causa una pérdida de precisión en la tarea original pero también en la tarea de *backdoor*. Podemos concluir por tanto que KFC se presenta en este escenario como una arquitectura viable para implementar FL y *blockchain* mientras ofrece una capa de seguridad, siendo igual de viable que PoFL pues ambas tienen sus características.

		accuracy		*accuracy*$_{10}$	
		PoFL	**KFC**	**PoFL**	**KFC**
EMNIST	Original	**0.9930**	0.9889	**0.9930**	0.9887
	Backdoor	0.0980	**0.0964**	0.0979	**0.0970**
Fashion	Original	**0.8961**	0.8645	**0.8986**	0.8645
MNIST	Backdoor	0.0871	**0.0768**	0.0989	**0.0813**
CIFAR-10	Original	**0.9397**	0.8870	**0.9283**	0.8839
	Backdoor	0.1163	**0.1025**	**0.0849**	0.0981

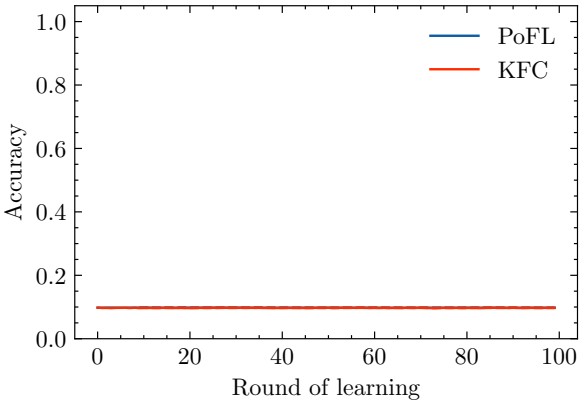
Tabla 7: *accuracy* y *accuracy*$_{10}$ comparando PoFL y KFC en el **escenario A** ante un ataque de backdoor.

15.2.2 *Escenario B: Todos los mineros son atacados*

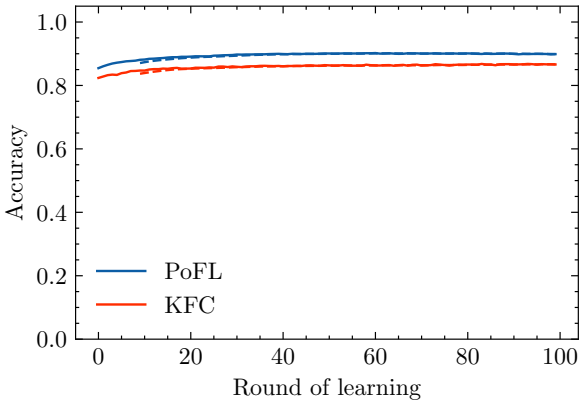
En esta sección volvemos al escenario más exigente en el que ya no se puede asumir la existencia de una *pool* no siendo atacada. Los resultados de KFC en este escenario se muestran en las Tablas 9 y 10 y Figuras 29 y 30. Un análisis detallado de los resultados nos lleva a las siguientes conclusiones comunes:



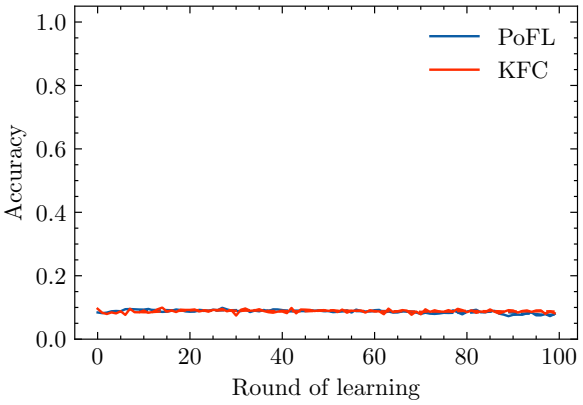
(a) EMNIST (Tarea original).



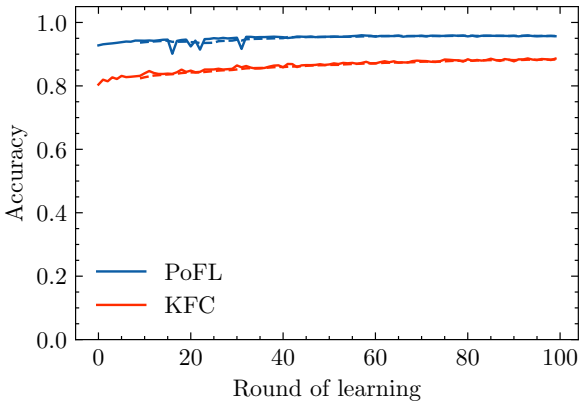
(b) EMNIST (Tarea backdoor).



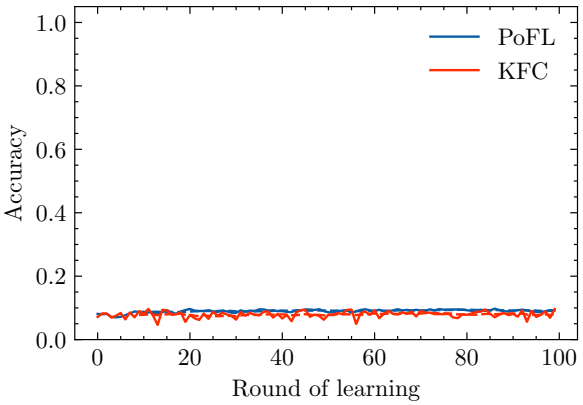
(c) Fashion EMNIST (Tarea original).



(d) Fashion EMNIST (Tarea backdoor).



(e) CIFAR (Tarea original).



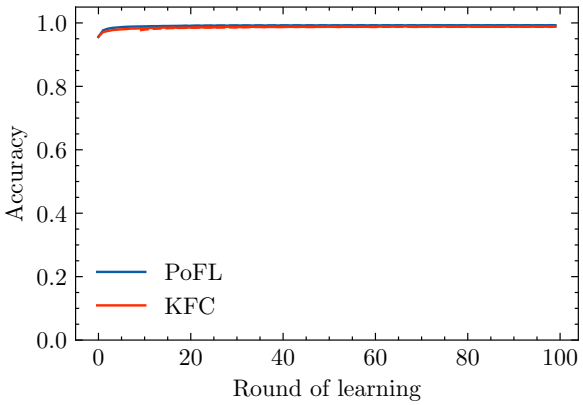
(f) CIFAR (Tarea backdoor).

Figura 27: *accuracy* (linea) y $accuracy_{10}$ (linea discontinua) de la tarea original y backdoor en EMNIST (a y b), Fashion MNIST (c y d) y CIFAR (e y f), respectivamente, en el **escenario A**.

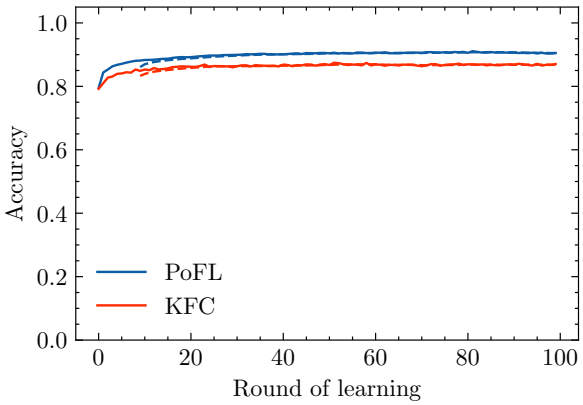
- La capacidad de KFC para mantener la precisión en la tarea original de *backdoor* en comparación con PoFL demuestra su resistencia en el nuevo escenario donde todos los mineros están bajo ataque.

	accuracy		*accuracy*$_{10}$	
	PoFL	**KFC**	**PoFL**	**KFC**
EMNIST	**0.9926**	0.9882	**0.9925**	0.9880
Fashion MNIST	**0.9051**	0.8705	**0.9049**	0.8684
CIFAR-10	**0.9628**	0.8901	**0.9604**	0.8875

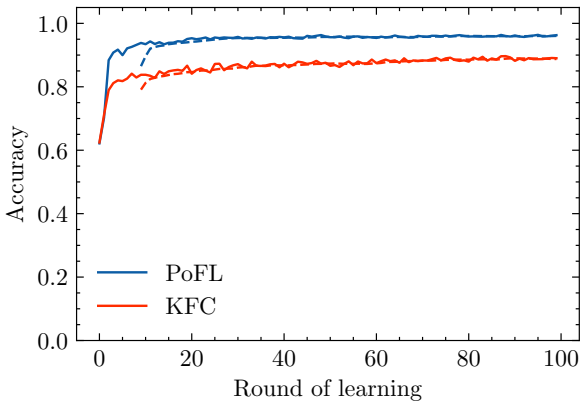
Tabla 8: *accuracy* y *accuracy*$_{10}$ comparando PoFL y KFC en el **escenario A** ante un ataque bizantino.



(a) EMNIST.



(b) Fashion EMNIST.



(c) CIFAR.

Figura 28: *accuracy* (línea) y *accuracy*$_{10}$ (línea discontinua) de las tareas originales EMNIST (a), Fashion MNIST (b) y CIFAR (c), bajo un ataque bizantino en el **escenario A**.

- Al el contrario que PoFL, KFC demuestra una resistencia excepcional ante todos los ataques, logrando una precisión mínima en la tarea de *backdoor* y mantener el rendimiento en el ataque bizantino, mitigando efectivamente los ataques.

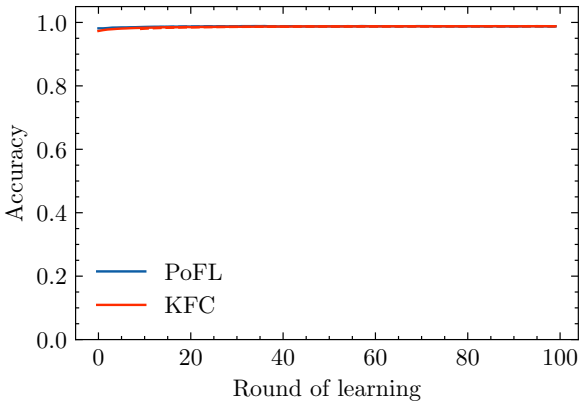
		accuracy		*accuracy*$_{10}$	
		PoFL	**KFC**		**KFC**
EMNIST	Original	0.8986	**0.9921**	0.9108	**0.9922**
	Backdoor	0.7536	**0.0995**	0.8002	**0.0994**
Fashion MNIST	Original	0.8133	**0.9000**	0.8002	**0.9000**
	Backdoor	0.8199	**0.0958**	0.9181	**0.0894**
CIFAR-10	Original	**0.8485**	0.8214	0.8223	**0.9123**
	Backdoor	0.8535	**0.1163**	0.9151	**0.1016**

Tabla 9: *accuracy* y *accuracy*$_{10}$ comparando PoFL y KFC en el **escenario B** ante un ataque de backdoor.

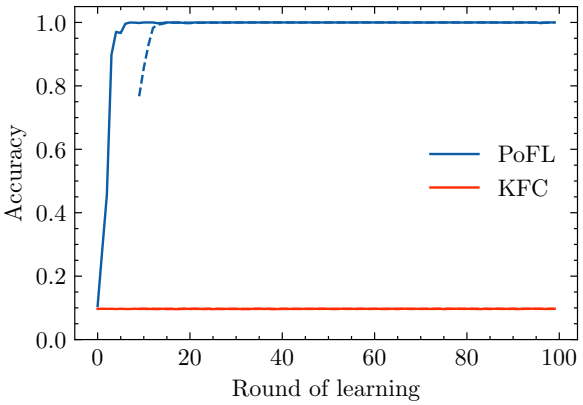
En resumen, KFC destaca como mecanismo de defensa. Los resultados nos indica como logra mitigar ambos tipos de ataques considerados, logrando reducir el impacto de la *backdoor* mientras se observa un rendimiento alto en la tarea original y en el ataque de tipo bizantino, superando a PoFL y nuestras arquitecturas de referencia en los entornos de la experimentación. Estos hallazgos confirman con firmeza que KFC supera a PoFL como la mejor opción en escenarios donde todas las *pools* podrían estar comprometidas. KFC logra eficazmente el doble objetivo de defender contra los ataques de *backdoor* en FL: maximizar el rendimiento de la tarea original y minimizar el impacto de la tarea inyectada. Además, su rendimiento contra ataques bizantinos se muestra excelente al presentar un comportamiento estable y una alta precisión. Esto se debe a que el mecanismo de agregación Krum permite filtrar actualizaciones maliciosas dentro de cada *pool* y logrando así que el mecanismo de consenso basado en rendimiento ofrezca un modelo no comprometido.

	accuracy		*accuracy*$_{10}$	
	PoFL	**KFC**	**PoFL**	**KFC**
EMNIST	0.9202	**0.9881**	0.9120	**0.9881**
Fashion MNIST	0.7344	**0.8709**	0.7476	**0.8690**
CIFAR-10	0.5681	**0.8856**	0.5648	**0.8914**

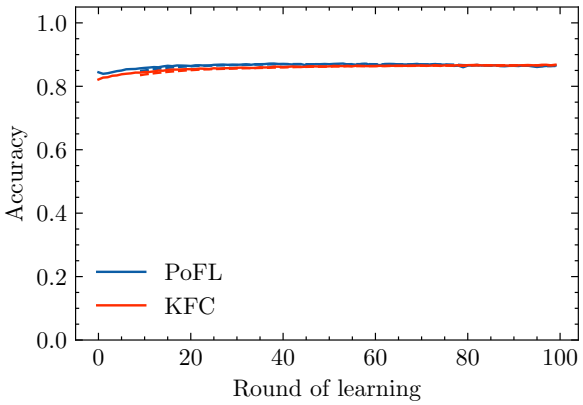
Tabla 10: *accuracy* y *accuracy*$_{10}$ comparando PoFL y KFC en el **escenario B** ante un ataque bizantino.



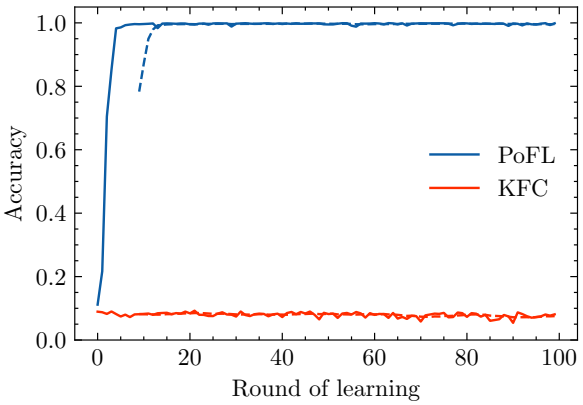
(a) EMNIST (Tarea original).



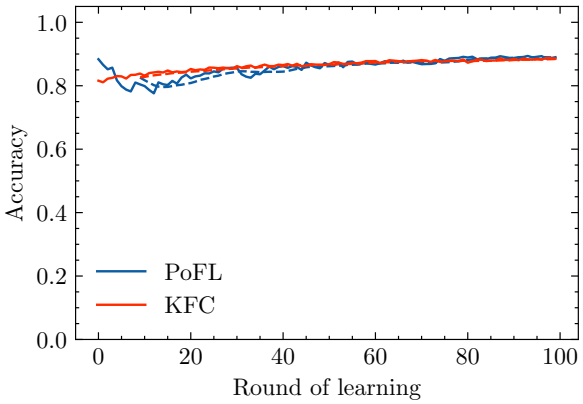
(b) EMNIST (Tarea backdoor).



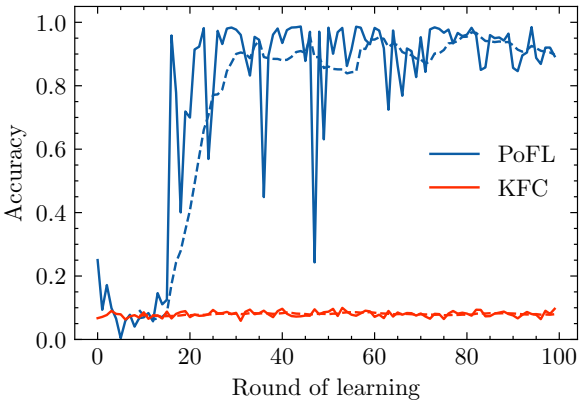
(c) Fashion EMNIST (Tarea original).



(d) Fashion EMNIST (Tarea backdoor).

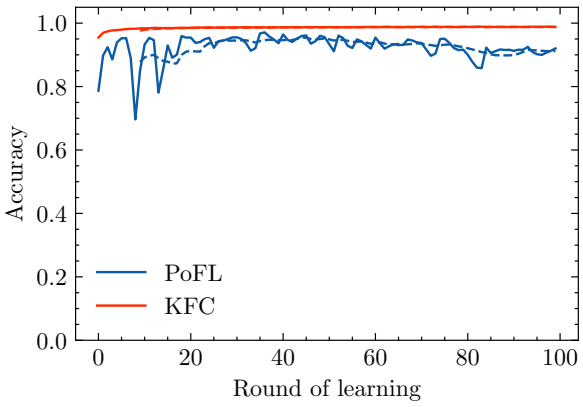


(e) CIFAR (Tarea original).

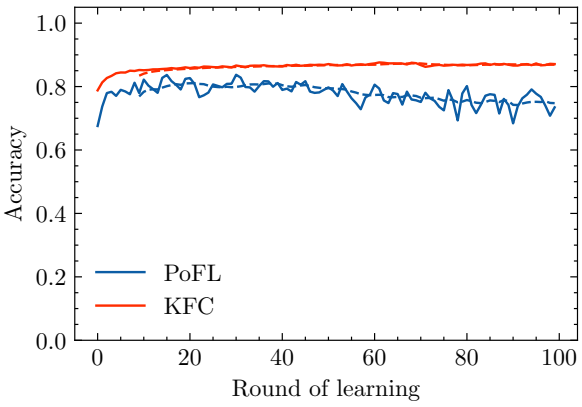


(f) CIFAR (Tarea backdoor).

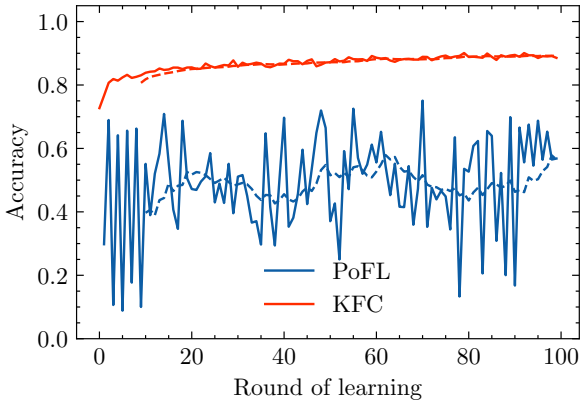
Figura 29: *accuracy* (linea) y $accuracy_{10}$ (linea discontinua) de la tarea original y backdoor en EMNIST (a y b), Fashion MNIST (c y d) y CIFAR (e y f), respectivamente, en el **escenario B**.



(a) EMNIST.



(b) Fashion EMNIST.



(c) CIFAR.

Figura 30: *accuracy* (línea) y $accuracy_{10}$ (línea discontinua) de las tareas originales EMNIST (a), Fashion MNIST (b) y CIFAR (c), bajo un ataque bizantino en el **escenario B**.

Parte V

CONCLUSIONES Y TRABAJO FUTURO

16

CONCLUSIONES

En este capítulo se resumen las conclusiones principales del trabajo. Algunas de ellas darán lugar a posibles líneas de desarrollo de trabajo futuro.

En el contexto de los sistemas de IA, la seguridad y la privacidad son aspectos fundamentales que no pueden ser subestimados. La creciente preocupación popular debido al creciente uso de estos sistemas ha llevado a la aparición del concepto de sistemas de *Trustworthy AI* y *High-Risk AI Systems* por parte de organismos como la Unión Europea. Estos requisitos nos llevan a investigar maneras de garantizar la fiabilidad de los sistemas. Usualmente, aquellos sistemas de IA que utilizan FL manejan datos sensibles que, si no se protegen adecuadamente, pueden ser vulnerables a una variedad de ataques, incluyendo la filtración de datos privados y la manipulación maliciosa del modelo.

Es por ello que durante este trabajo nos hemos centrado en estudiar cómo la tecnología *blockchain* puede ayudarnos a proteger un esquema federado. Se exponen a continuación los resultados observados:

- Se ha observado cómo la combinación de las tecnologías *blockchain* y FL ayudan a resolver muchos problemas presentes en el paradigma de aprendizaje actual. Aunque hayamos hecho hincapié en aquellos relacionados con la seguridad del modelo, hemos visto también cómo el uso de la *blockchain* ayuda a tener una mejor escalabilidad de la red o a no depender de una entidad central entre otros beneficios. Esto demuestra el enorme potencial que tiene aún por explotar este campo en el que no existe un estándar claro y está lleno de constantes innovaciones y propuestas.
- Los experimentos avalan la hipótesis planteada de que PoFL resulta ser un mecanismo de defensa viable contra ataques al modelo en el escenario en el que haya menos clientes bizantinos que mineros. Esta conclusión puede resultar sorprendente pues PoFL fue concebido originalmente como un método de consenso para resolver el problema de la eficiencia energética de la *blockchain*, por lo que esta resistencia resulta un efecto secundario del diseño del método.
- Nuestra propuesta KFC, basada en la combinación de PoFL con un agregador más robusto, demuestra ser una defensa sólida contra los ataques al modelo en

escenarios más difíciles donde el número de atacantes es igual o mayor al de mineros. Además, al basarse en PoFL, sigue manteniendo el diseño centrado en la eficiencia energética, algo clave para cualquier arquitectura de *blockchain* que se quiera aplicar en la realidad. Por lo tanto, se puede considerar una mejora considerable sobre PoFL y una gran opción para integrar *blockchain* y FL.

- Durante la experimentación realizada se han considerado múltiples modelos de AA de distinta complejidad y en múltiples conjuntos de datos con tamaño y características variadas. Es por ello y por la consistencia de los resultados en las distintas configuraciones por lo que se puede afirmar que las conclusiones extraídas de esta experimentación son robustas y generalizables a otros entornos federados.

Estas observaciones nos muestran cómo la incorporación de la tecnología *blockchain* puede ofrecer una capa adicional de seguridad mediante la descentralización y la inmutabilidad de los datos, lo que reduce el riesgo de manipulación y aumenta la transparencia. Sin embargo, la tecnología *blockchain* por sí sola no es suficiente para garantizar una protección completa. Por ello es crucial implementar mecanismos adicionales, como métodos robustos de agregación y algoritmos de consenso seguros, para defenderse contra ataques sofisticados y garantizar la integridad y confidencialidad de los datos y el modelo. Un ejemplo de estos mecanismos adicionales es nuestra propuesta KFC.

Estos resultados obtenidos son prometedores y novedosos. Es por ello, que se ha elaborado un artículo científico y ha sido presentado al congreso *European Conference on Artificial Intelligence*[1] recopilando los experimentos y resultados expuestos en esta memoria.

1 https://www.ecai2024.eu/

17

TRABAJO FUTURO

El trabajo realizado, al estar en un área en auge y tratar con un problema realmente importante (la privacidad de los datos en IA), tiene un amplio recorrido. A continuación planteamos el posible trabajo a realizar a raíz de las conclusiones extraídas de esta memoria:

- Con el fin de solventar el problema presente de PoFL, que radica en una falta de protección dentro de una *pool*, hemos decidido usar Krum. Sin embargo, como hemos visto este no es el único mecanismo posible para agregar esta capa adicional de seguridad. Por lo tanto, se pueden realizar variaciones de KFC donde se usen mecanismos alternativos a Krum.
- Los ataques al modelo probados han sido los ataques bizantinos y los de *backdoor*. Sin embargo, existe una mayor variedad de ataques al modelo que no han sido probados en esta memoria, como por ejemplo aquellos ataques realizados durante la inferencia del modelo.
- Una línea futura de trabajo es ver cómo la *blockchain* puede ayudar a mitigar ataques de privacidad a los datos en un esquema federado. Estos ataques constituyen un gran interés pues están directamente vinculados con el problema de la privacidad. Se podría tomar como base los resultados encontrados en esta memoria con el fin de construir, deseablemente, una defensa conjunta frente ataques a la privacidad de los datos y la integridad al modelo federado, el cual es uno de los mayores retos que existen a día de hoy en el paradigma del FL.

En conclusión, este trabajo profundiza en el estado actual de la aplicabilidad de *blockchain* y FL ofreciendo una solución novedosa a uno de los principales problemas actuales. Es por ello que este problema puede causar un gran impacto ya sea ofreciendo arquitecturas más seguras en entornos en los que ya se aplica la combinación nombrada o abriendo paso a futuras investigaciones que sigan la línea aquí planteada.

BIBLIOGRAFÍA

[1] Sana Awan, Fengjun Li, Bo Luo, and Mei Liu. Poster: A reliable and accountable privacy-preserving federated learning framework using the blockchain. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, CCS '19, page 2561–2563, New York, NY, USA, 2019. Association for Computing Machinery.

[2] Peva Blanchard, El Mahdi El Mhamdi, Rachid Guerraoui, and Julien Stainer. Machine learning with adversaries: byzantine tolerant gradient descent. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, page 118–128, 2017. ISBN 9781510860964.

[3] Vitalik Buterin. A next generation smart contract & decentralized application platform. 2015.

[4] Gregory Cohen, Saeed Afshar, Jonathan Tapson, and André van Schaik. Emnist: Extending mnist to handwritten letters. In *2017 International Joint Conference on Neural Networks (IJCNN)*, pages 2921–2926, 2017.

[5] Bhaskar Krishnamachari Duc A. Tran, My T. Thai. *Handbook on Blockchain*. Springer Cham, 2023.

[6] Natalia Díaz-Rodríguez, Javier Del Ser, Mark Coeckelbergh, Marcos López de Prado, Enrique Herrera-Viedma, and Francisco Herrera. Connecting the dots in trustworthy artificial intelligence: From ai principles, ethics, and key requirements to responsible ai systems and regulation. *Information Fusion*, 99:101896, 2023. ISSN 1566-2535.

[7] Paul Garthwaite, Ian Jolliffe, and Byron Jones. *Statistical Inference*. Oxford University Press, 2002.

[8] Andrew Gelman, John B. Carlin, Hal S. Stern, and Donald B. Rubin. *Bayesian Data Analysis*. 2004.

[9] J. Goldberg, W. H. Kautz, and P. M. Melliar-Smith. Development and analysis of the software implemented fault-tolerance (sift) computer. 1984.

[10] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016.

[11] Huaqun Guo and Xingjie Yu. A survey on blockchain technology and its security. *Blockchain: Research and Applications*, 3(2):100067, 2022. ISSN 2096-7209.

[12] Francisco Herrera, Daniel Jiménez-López, Alberto Argente-Garrido, Nuria Rodríguez-Barroso, Cristina Zuheros, Ignacio Aguilera-Martos, Beatriz Bello, Mario García-Márquez, and M. Victoria Luzón. Flex: Flexible federated learning framework, 2024.

[13] Anthony Dewayne Hunt. Bitcoin: A peer-to-peer electronic cash system. 2008.

[14] Hyesung Kim, Jihong Park, Mehdi Bennis, and Seong-Lyun Kim. Blockchained on-device federated learning. *IEEE Communications Letters*, 24(6):1279–1283, 2020.

[15] Leslie Lamport, Robert Shostak, and Marshall Pease. The byzantine generals problem. *ACM Trans. Program. Lang. Syst.*, 4(3):382–401, jul 1982. ISSN 0164-0925.

[16] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[17] David G. Luenberger. *Optimization by Vector Space Methods*. John Wiley & Sons, 1969.

[18] M. Victoria Luzón, Nuria Rodríguez-Barroso, Alberto Argente-Garrido, Daniel Jiménez-López, Jose M. Moyano, Javier Del Ser, Weiping Ding, and 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.

[19] Umer Majeed and Choong Seon Hong. Flchain: Federated learning via mec-enabled blockchain network. In *2019 20th Asia-Pacific Network Operations and Management Symposium (APNOMS)*, pages 1–4, 2019.

[20] Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Aguera y Arcas. Communication-Efficient Learning of Deep Networks from Decentralized Data, 20–22 Apr 2017.

[21] H. Brendan McMahan, Daniel Ramage, Kunal Talwar, and Li Zhang. Learning differentially private recurrent language models. In *International Conference on Learning Representations*, 2018.

[22] Thomas M. Mitchell. *Machine Learning*. McGraw-Hill, 1997.

[23] Yaser S. Abu Mostafa, Malik Magdon Ismail, and Hsuan Tien Lin. *Learning from data: a short course*. Yaser S. Abu Mostafa, Malik Magdon Ismail, Hsuan Tien Lin, 2012.

[24] Council of European Union. Regulation of the european parliament and of the council laying down harmonised rules on artificial intelligence (artificial intelligence act) and amending certain union legislative acts, 2021.

[25] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Yang, Zach DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library, 2019.

[26] Boris Polyak. *Introduction to Optimization*. Optimization Software, Inc., Publications Division, 1987.

[27] Davy Preuveneers, Vera Rimmer, Ilias Tsingenopoulos, Jan Spooren, Wouter Joosen, and Elisabeth Ilie-Zudor. Chained anomaly detection models for federated learning: An intrusion detection case study. *Applied Sciences*, 8(12), 2018.

[28] Xidi Qu, Shengling Wang, Qin Hu, and Xiuzhen Cheng. Proof of federated learning: A novel energy-recycling consensus algorithm. *IEEE Transactions on Parallel and Distributed Systems*, 32(8):2074–2085, 2021.

[29] Nuria Rodríguez-Barroso, Daniel Jiménez-López, M. Victoria Luzón, Francisco Herrera, and Eugenio Martínez-Cámara. Survey on federated learning threats: Concepts, taxonomy on attacks and defences, experimental study and challenges. *Information Fusion*, 90:148–173, 2023.

[30] Sheldon M. Ross. *Introduction to Probability Models*. Academic Press, 2019.

[31] Nareman Sabry, Bahaa Shabana, Mohamed Handosa, and M. Z. Rashad. Adapting blockchain's proof-of-work mechanism for multiple traveling salesmen problem optimization. *Scientific Reports*, 13(1):14676, 2023.

[32] Peter Spellucci. *Numerische Verfahren der nichtlinearen Optimierung*. Birkhäuser Basel, 1993.

[33] Mingxing Tan and Quoc V. Le. Efficientnet: Rethinking model scaling for convolutional neural networks. *CoRR*, abs/1905.11946, 2019.

[34] Scott Thiebes, Sebastian Lins, and Ali Sunyaev. Trustworthy artificial intelligence. *Electronic Markets*, 31, 10 2020.

[35] Antonio Torralba, Rob Fergus, and William T. Freeman. 80 million tiny images: A large data set for nonparametric object and scene recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 30(11):1958–1970, 2008.

[36] Stacey Truex, Ling Liu, Mehmet Emre Gursoy, Lei Yu, and Wenqi Wei. Demystifying membership inference attacks in machine learning as a service. *IEEE Transactions on Services Computing*, 14(6):2073–2089, 2021.

[37] Tao Wang, Yunjian Xu, Chathura Withanage, Lan Lan, Selin Damla Ahipaşaoğlu, and Costas A. Courcoubetis. A fair and budget-balanced incentive mechanism for energy management in buildings. *IEEE Transactions on Smart Grid*, 9(4):3143–3153, 2018.

[38] Jiasi Weng, Jian Weng, Jilian Zhang, Ming Li, Yue Zhang, and Weiqi Luo. Deepchain: Auditable and privacy-preserving deep learning with blockchain-based incentive. *IEEE Transactions on Dependable and Secure Computing*, 18(5):2438–2455, 2021.

[39] Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms, 2017.

[40] Jia Xu, Jinxin Xiang, and Dejun Yang. Incentive mechanisms for time window dependent tasks in mobile crowdsensing. *IEEE Transactions on Wireless Communications*, 14(11):6353–6364, 2015.

[41] Qiang Yang, Yang Liu, Yong Cheng, Yan Kang, Tianjian Chen, and Han Yu. *Federated learning*. Morgan & Claypool Publishers, 2019.

[42] Weishan Zhang, Qinghua Lu, Qiuyu Yu, Zhaotong Li, Yue Liu, Sin Kit Lo, Shiping Chen, Xiwei Xu, and Liming Zhu. Blockchain-based federated learning for device failure detection in industrial iot. *IEEE Internet of Things Journal*, 8(7): 5926–5937, 2021.

[43] Yang Zhao, Jun Zhao, Linshan Jiang, Rui Tan, Dusit Niyato, Zengxiang Li, Lingjuan Lyu, and Yingbo Liu. Privacy-preserving blockchain-based federated learning for iot devices. *IEEE Internet of Things Journal*, 8(3):1817–1829, 2021.

[44] Lu Zhou, Abebe Diro, Akanksha Saini, Shahriar Kaisar, and Pham Cong Hiep. Leveraging zero knowledge proofs for blockchain-based identity sharing: A survey of advancements, challenges and opportunities. *Journal of Information Security and Applications*, 80:103678, 2024. ISSN 2214-2126.

[45] Juncen Zhu, Jiannong Cao, Divya Saxena, Shan Jiang, and Houda Ferradi. Blockchain-empowered federated learning: Challenges, solutions, and future directions. *ACM Comput. Surv.*, 55(11), feb 2023.