Universidad Politécnica
de Madrid

Criptografía de Curvas Elípticas y Redes Neuronales

Javier García Muñoz

Director

Prof. Ángel González Prieto

Codirector

Prof. Raúl Lara Cabrera

Proyecto final presentado para optar al grado de
Ingeniería del Software

Departamento de Sistemas Informáticos

Escuela Técnica Superior de Ingeniería de Sistemas Informáticos

Madrid, 2022

"We can only see a short distance ahead;
but we can see plenty there that needs to be done."

Alan Turing

ABSTRACT

Criptografía de Curvas Elípticas y Redes Neuronales

por Javier García Muñoz

Abstract en Español

El objetivo de este proyecto es analizar el comportamiento de la criptografía de curvas elípticas sobre cuerpos finitos frente al aprendizaje automáticos de la redes neuronales. Para ello, nos hemos centrado en el estudio de los fundamentos matemáticos necesarios para poder comprender las propiedades de diferentes conjuntos de elementos, como son los grupos, anillos y cuerpos. Estas colecciones, junto al Problema del Logartimo Discreto, sirven de base para entender el comportamiento y las características de la criptografía de curvas elípticas. Por último, se ha realizado una introducción al mundo del aprendizaje automático, en concreto al ámbito de las redes neuronales artificiales. La unión de ambas partes, nos lleva finalmente a un caso práctico en el que ponemos a prueba los conocimientos adquiridos y analizamos como responde esta rama de la inteligencia artificial frente a dos algoritmos de cifrado: el cifrado César y el cifrado ElGamal sobre curvas elípticas.

Palabras clave: *Criptografía, cuerpos finitos, curvas elípticas, redes neuronales.*

Abstract in English

The aim of this project is to analyse the performance of elliptic curve cryptography on finite fields against machine learning of neural networks. To this end, we have focused on the study of the mathematical foundations necessary to be able to understand the properties of different sets of elements, such as groups, rings and fields. These collections, together with the Discrete Logarithm Problem, serve as a basis for understanding the behaviour and characteristics of elliptic curve cryptography. Finally, an introduction to the world of machine learning, in particular to the field of artificial neural networks, was given. Both theoretical parts come together in a practical case in which we test the knowledge acquired and analyse how this branch of artificial intelligence responds to two encryption algorithms: the Caesar cipher and the ElGamal cipher on elliptic curves.

Key words: *Cryptography, finite fields, elliptic curves, neural networks.*

Agradecimientos

Este proyecto cierra una gran etapa de mi vida, en la que he crecido en todos los ámbitos. Sin ayuda nunca hubiera podido alcanzar esta meta, y estoy realmente agradecido a todas las personas que de una u otra forma, me han apoyado a lo largo de estos años.

A mi tutor Ángel. Por su paciencia, dedicación y vocación. He tenido muchos profesores a lo largo de mi etapa académica, pero ninguno con su pasión y ganas de enseñar y de continuar aprendiendo.

A mis padres y mi hermana, porque ellos pusieron las primeras piedras de mi camino y siempre han estado y estarán para ayudarme.

A Lucía, porque apareció a medio camino, pero ha logrado sacar mi mejor versión y motivarme a buscar nuevas metas.

A mis amigos, por hacerme desconectar de la realidad y disfrutar en todo momento.

Índice general

Introducción

Durante siglos el ser humano se ha visto en la necesidad de realizar comunicaciones secretas para preservar la privacidad de la información transmitida, con el objetivo de que solo las personas autorizadas pudieran leer dicha información. Podemos encontrar multitud de ejemplos en la historia, desde el cifrador del César utilizado por los romanos, hasta la máquina Enigma, utilizada por las fuerzas armadas alemanas durante la Segunda Guerra Mundial. Con la llegada de la Era Digital y el uso masivo de las comunicaciones digitales, ha aumentado el peligro de que esas comunicaciones sean interceptadas, y para garantizar la seguridad, se han desarrollado numerosos algoritmos, protocolos y sistemas que protegen la información.

La criptografía es la técnica utilizada para el cifrado de dicha información para dotar de seguridad a las comunicaciones. La palabra criptografía proviene del griego, donde *Kryptos* significa oculto y *graphia* significa escritura, y su definición es *arte de escribir con clave secreta o de un modo enigmático.* Hay dos tipos fundamentales de criptosistemas en función del tipo de cifrado: **simétricos** que son aquellos que emplean la misma clave para cifrar y descifrar, y **asimétricos** que emplean diferentes claves (pública y privada).

Uno de los problemas matemáticos en los que se basan algunos sistemas criptográficos es el del logaritmo discreto. Su complejidad se debe a que dados dos enteros a y m módulo n, calcular a^m es computacionalmente sencillo, pero conociendo $x = a^m$, es muy difícil calcular el exponente m. En algunos casos no hay ningún algoritmo eficaz conocido para resolver el problema en general y la complejidad en el caso promedio resulta tan difícil como en el peor de los casos.

A lo largo del capítulo 1, se estudiarán las bases matemáticas necesarias para la correcta comprensión del trabajo. Este capítulo está dividido en tres partes: grupos, anillos y cuerpos.

Los **grupos** son estructuras algebraicas formadas por un conjunto de elementos y una operación definida sobre dicho conjunto, que es cerrada, asociativa, con un elemento neutro y con inverso para todos los elementos que lo forman. A su vez, un grupo estará formado por subconjuntos que cumplen las mismas propiedades, conocidos como subgrupos. En este apartado, es fundamental la comprensión del Teorema de Lagrange, que relaciona el orden o tamaño de grupos finitos y sus subgrupos, de manera que el orden de los subgrupos es divisor del orden del grupo. Además a lo largo de esta sección veremos algunos tipos de grupos, grupos cocientes, homomorfismos de grupos y los teoremas de isomorfía.

Los **anillos** además de cumplir las propiedades de un grupo, tienen definidas dos operaciones sobre el conjunto de sus elementos, la de producto y la de suma. A lo largo de esta sección, también trataremos algunos tipos de anillos (como el anillo de polinomios), los dominios de integridad, que son anillos conmutativos que carecen de elementos divisores de cero, o los ideales de un anillo, que son conjuntos que operados por otros elementos del anillo, derivan de nuevo en un elemento del ideal.

Los **cuerpos** se diferencian de los anillos en que cumplen la propiedad de tener inversos para ambas operaciones (excepto para el elemento neutro de la suma). Esto significa que todos los elementos menos los neutros, tienen inverso para la multiplicación. Dentro de esta sección, se incluyen las extensiones de cuerpos, que son conjuntos más grandes que los cuerpos donde además se pueden resolver ecuaciones polinómicas. Por último, trataremos los cuerpos finitos y primos, los cuales son esenciales en el desarrollo de este trabajo, ya que serán utilizados para la criptografía de curvas elípticas.

Durante las últimas décadas, las curvas elípticas ha aumentado su importancia tanto en la teoría de números como en campos como la criptografía, hasta llegar a ser parte de algunos estándares industriales. Este tipo de criptografía utiliza diferentes algoritmos para cifrar o descifrar información a partir del problema del logaritmo discreto, llegando a garantizar una seguridad equivalente a métodos antiguos (como RSA), pero con la ventaja de trabajar con claves mucho menores que en otros casos. Esto implica una mejora de rendimiento significativa, y a su vez, el aumento de su uso en diferentes ámbitos, como por ejemplo, tarjetas inteligentes o en la tecnología blockchain.

A lo largo del capítulo 2, se desarrollan conceptos acerca de las **curvas elípticas**, sus propiedades y su uso en criptografía. Se profundiza en la explicación sobre el problema del logaritmo discreto y se pueden ver diferentes algoritmos de cifrado y su funcionamiento. La intención de este trabajo es probar la seguridad del problema del logaritmo discreto en curvas elípticas. Las curvas elípticas conforman un campo de las matemáticas relativamente nuevo. Hasta el año 1985, no se escucha hablar de estas curvas y su popularidad aumenta porque se empiezan a implementar sistemas criptográficos más difíciles de romper, cuyas claves son más pequeñas que con otros criptosistemas de clave pública convencionales.

Durante el capítulo 3 el tema principal es el **aprendizaje automático** (machine learning). Este campo trabaja con algoritmos que tratan de identificar patrones en conjuntos de datos, y a partir de estos patrones, se crean modelos de datos para hacer predicciones. El aprendizaje automático es utilizado en una gran cantidad de aplicaciones, como reconocimiento del lenguaje escrito, diagnósticos médicos, juegos, robótica o motores de búsqueda. Dentro del aprendizaje automático encontramos dos tipos según si es el aprendizaje es supervisado o no supervisado.

Concretamente, este capítulo expone las **redes neuronales artificiales**. Estos algoritmos aprenden a través de un elemento de retroalimentación, igual que los seres humanos siempre que tratamos de aprender algo, lo hacemos tal y como nos han dicho que es la manera correcta, por lo tanto, ajustamos nuestra conducta según los conocimientos adquiridos. La retroalimentación en una red neuronal es conocida como propagación hacia atrás y compara la salida de la red con lo que resultado esperado. Para entrenar una red de esta manera, se necesita un gran conjunto de ejemplos a partir de los cuáles, la red pueda aprender.

El objetivo de este proyecto es comprobar la seguridad de la criptografía de curvas elípticas a partir del aprendizaje automático. Esto se prueba en el capítulo 4. Al ser la primera vez que trabajo con redes neuronales y debido a la complejidad que presenta la criptografía de curvas elípticas, en este

capítulo también se incluye una prueba inicial frente al Cifrado César. Este consiste en desplazar todas las letras el mismo número de posiciones, y este cifrado es mucho más sencillo para que una red neuronal artificial aprenda a resolverlo. Finalmente, nuestra red neuronal contra la criptografía curvas elípticas se basa en el criptosistema de ElGamal, recibirá como parámetro de entrada la clave pública del cifrado y tratará de devolver la clave privada.

Tras realizar la diseño y el desarrollo de los modelos para ambas redes neuronales, los resultados obtenidos han sido disparejos para cada caso. Para el cifrado César, la red es capaz de encontrar un patrón para resolver el algoritmo y se ve claramente el aumento de la precisión y la disminución del error. Bien es cierto que este algoritmo es muy sencillo y no representa ningún problema para el aprendizaje. Por otro lado, en el caso de la criptografía de curvas elípticas, la red neuronal no es capaz de mejorar la precisión, por lo tanto, no distingue el ruido y no encuentra un patrón para alcanzar el resultado deseado.

Capítulo 1

Fundamentos matemáticos de la criptografía

1.1 Teoría de grupos

El estudio de algunos conceptos matemáticos es necesario para una correcta comprensión del texto y de los puntos clave de esta memoria. La criptografía es definida como el estudio de los algoritmos, protocolos y sistemas, para los que se necesita una base matemática. A lo largo de este capítulo, se estudiarán ciertos tipos de estructuras algebraicas y los conjuntos de elementos, operaciones y propiedades que las forman, como paso previo a la criptografía de curvas elípticas. En primer lugar vamos a conocer la teoría de grupos, que será la estructura más básica de la memoria y clave para entender los anillos y cuerpos. El contenido de esta sección se ha desarrollado en base a los capítulos 4 y 5 del libro *Números, grupos y anillos* de Eugenio Hernández y José Dorronsoro [JE07]. Además se recomienda una lectura previa del capítulo 3 sobre congruencias, que serán utilizadas a lo largo del capítulo.

1.1.1 Definición de Grupo

Un grupo es un conjunto no vacío de elementos G donde se encuentra definida la operación binaria $\cdot$ (adición o multiplicación) como una función $G \times G \to G$ y se satisfacen las siguientes propiedades:

- La operación binaria $\cdot$ es **cerrada** dentro del conjunto de elementos G. El resultado de la operación entre dos elementos del grupo G, será otro elemento del grupo.
- La operación binaria $\cdot$ es **asociativa** entre los elementos del grupo. Dados tres elementos del grupo G, se verifica que el orden de ejecución de las operaciones entre ellos, no afecta al resultado final.

$$g_1 \cdot (g_2 \cdot g_3) = (g_1 \cdot g_2) \cdot g_3$$

- Existe un elemento e que es **neutro**, de manera que la operación de cualquier elemento del grupo G con el neutro, equivale al elemento original. En caso de que la operación binaria sea la suma, también se conoce como **elemento cero** o **elemento nulo**.

$$g \cdot e = g$$

- Todo elemento g del grupo G tiene un elemento **inverso** g^{-1}. La operación binaria entre el elemento y su inverso, siempre equivale al elemento neutro.

$$g \cdot g^{-1} = e$$

Otra forma para definir al grupo G con la operación binaria $\cdot$ es $(G, \cdot)$.

Ejemplo 1.1.1. El conjunto de los números reales tiene estructura de grupo con la operación suma $(\mathbb{R}, +)$. Su elemento neutro es el 0 y el inverso de g es $-g$, ya que $g + (-g) = 0$.

Ejemplo 1.1.2. El conjunto de los números racionales no es grupo con la operación producto $(\mathbb{Q},\times)$, debido a que el elemento 0 no tiene inverso.

1.1.1.1 Grupo conmutativo o abeliano

Todos los grupos G cumplen las anteriores propiedades, pero algunos se caracterizan por tener otras. Aquellos en los que se cumpla la propiedad conmutativa, es decir, si el resultado de la operación entre dos elementos cualquiera, es el mismo, sea cual sea el orden de los elementos, es conocido como grupo **conmutativo** o **abeliano**.

$$g_1 \cdot g_2 = g_2 \cdot g_1$$

Ejemplo 1.1.3. El conjunto de los números enteros $(\mathbb{Z})$ junto con la operación de suma, forman un grupo abeliano, ya que se cumplen todas las propiedades anteriores.

1.1.1.2 Orden de un grupo

Se conoce como **orden de** G al número de elementos que forman un grupo G, es decir, el orden es su cardinalidad, lo denotaremos como $|G|$ y diremos que es un grupo finito. Si el orden del grupo G es 1, entonces el grupo se denomina grupo trivial.

Hay que diferenciar entre el orden de un grupo y el orden de un elemento. Posteriormente trataremos esto en profundidad, ya que es necesario para entender el Teorema de Lagrange (1.1.23).

Ejemplo 1.1.4. El orden del grupo formado por las clases de equivalencia módulo 2 $(\mathbb{Z}_2)$ es 2, ya que está formado por [0] y [1].

1.1.2 Ejemplos de grupos

Los siguientes ejemplos nos ayudarán a entender y profundizar en la estructura de un grupo.

1.1.2.1 Grupo de congruencias

Se recomienda la lectura del capítulo 3 del libro *Números, grupos y anillos* de Eugenio Hernández y José Dorronsoro [JE07], para una introducción completa a la teoría de congruencias.

Una congruencia equivale a que dos elementos a y b tengan el mismo resto al dividirlos por otro número m, conocido como módulo.

$$a \equiv b(m) \leftrightarrow a - b = k\dot{m}$$

La relación de congruencia definida en el conjunto $\mathbb{Z}$ de los números enteros es una relación de equivalencia, conocida como el conjunto de las clases de congruencias módulo m.

$$\mathbb{Z}_m = [0], [1], [2], ..., [m-1]$$

En la teoría de congruencias citada anteriormente, se demostró que las operaciones de suma y multiplicación están bien definidas en $\mathbb{Z}_m$ y son cerradas en el conjunto. Con la operación de suma, ($\mathbb{Z}_m$,+) es un grupo abeliano, pero con la multiplicación, no. Esto se debe a que al menos el elemento [0] no tiene elemento inverso y no se cumple la propiedad (4). En caso de trabajar con otros tamaños de modulo, nos encontramos con más elementos que no tienen inverso, como por ejemplo [2],[3],[4] para $\mathbb{Z}_6$.

Teorema 1.1.5. *Para todo número entero positivo primo p, ($\mathbb{Z}_p^*, \times$) es un grupo abeliano con $p-1$ elementos.*

Demostración 1.1.6. Para demostrar que la operación de adición es cerrada en el conjunto $\mathbb{Z}_m^*$, no puede darse el caso [a]×[b]=[0]. Si suponemos que se da ese caso, se deduce que $a \times b = k \times p$ para algún elemento k de $\mathbb{Z}$; entonces p divide a $a \times b$, o lo que es lo mismo $p|a$ o $p|b$. Esto implica que [a] o [b] equivale a [0], contradiciendo que [a],[b] pertenecen al conjunto $\mathbb{Z}_m^*$.

1.1.2.2 Grupo simétrico S_n

El grupo simétrico S_n sobre un conjunto C es el formado por las aplicaciones biyectivas del conjunto en sí mismo. Cuando el conjunto es finito, se denomina grupo de permutaciones de n elementos, es decir, las permutaciones son las diferentes formas de ordenar al conjunto. A partir del conjunto formado por n números naturales $C = 0, 1, 2, ..., n$, obtenemos el conjunto de todas sus biyecciones S_n.

La operación del grupo $\circ$ es la composición de funciones, que tiene que ser cerrada. Para ello, necesitamos comprobar que para las aplicaciones biyectivas f, g sobre C, $f \circ g$ es biyectiva también. A partir

de dos elementos del conjunto, tenemos que $g \circ f(x) = g \circ f(y)$, esto equivale a $g(f(x)) = g(f(y))$ y como ambas funciones son inyectivas, se demuestra que $x = y$ y que $g \circ f$ es inyectiva. Para demostrar que es sobreyectiva, basta con tomar $g(y) = z$ y como f es sobreyectica $f(x) = y$, $g(f(x)) = z$. Ha quedado demostrado que la composición de aplicaciones es cerrada en C y se demuestra fácilmente que es asociativa. Por último, hay que encontrar el elemento neutro y que todos tienen un inverso. La aplicación identidad de C es I_C, que es nuestro elemento neutro y para para cualquier f tenemos f^{-1}, donde $f(f^{-1}(x)) = f(f^{-1}(y))$, y también,$f^{-1}(x) = f^{-1}(f(y)) = y$, por lo que f^{-1} es biyectiva. Como esta definición implica que $f^{-1} \circ f(x) = f(x) \circ f^{-1}$, para cualquier elemento x de C, podemos asegurar que S_n es un grupo.

Los elementos de S_n se escriben de la siguiente manera:

$$\begin{pmatrix} 1 & 2 & \cdots & n \\ \alpha(1) & \alpha(2) & \cdots & \alpha(n) \end{pmatrix}$$

S_n no es un grupo abeliano porque no es conmutativo, ya que $f \circ g$ no es lo mismo que $g \circ f$.

Ejemplo 1.1.7. A partir de dos permutaciones $f, g \in S_3$:

$$f = \begin{pmatrix} 1 & 2 & 3 \\ 2 & 1 & 3 \end{pmatrix}, \quad g = \begin{pmatrix} 1 & 2 & 3 \\ 3 & 2 & 1 \end{pmatrix}$$

Y la operación cambiando el orden:

$$f \circ g = \begin{pmatrix} 1 & 2 & 3 \\ 2 & 1 & 3 \end{pmatrix} \circ \begin{pmatrix} 1 & 2 & 3 \\ 3 & 2 & 1 \end{pmatrix} = \begin{pmatrix} 1 & 2 & 3 \\ 3 & 1 & 2 \end{pmatrix}$$

$$g \circ f = \begin{pmatrix} 1 & 2 & 3 \\ 3 & 2 & 1 \end{pmatrix} \circ \begin{pmatrix} 1 & 2 & 3 \\ 2 & 1 & 3 \end{pmatrix} = \begin{pmatrix} 1 & 2 & 3 \\ 2 & 3 & 1 \end{pmatrix}$$

De esta manera confirmamos que el grupo D_3 no es conmutativo.

1.1.2.3 Grupo diédrico D_n

Otro ejemplo de grupo es el diédrico o diedral D_n, que se obtiene a partir de la simetría de un polígono regular de n lados. Su operación es la composición de los movimientos en el plano y se llaman simetrías (rotaciones y reflexiones).

Para entender mejor las simetrías nos vamos a apoyar en la figura del triángulo equilátero, también conocido como el grupo diedral D_3. La figura 1.1 muestra las simetrías de un triángulo equilátero. I sería la identidad, A (120^o) y A^2 (240^o) las rotaciones en contra del sentido de las agujas de un reloj y B_0, B_1, y B_2 son las reflexiones.

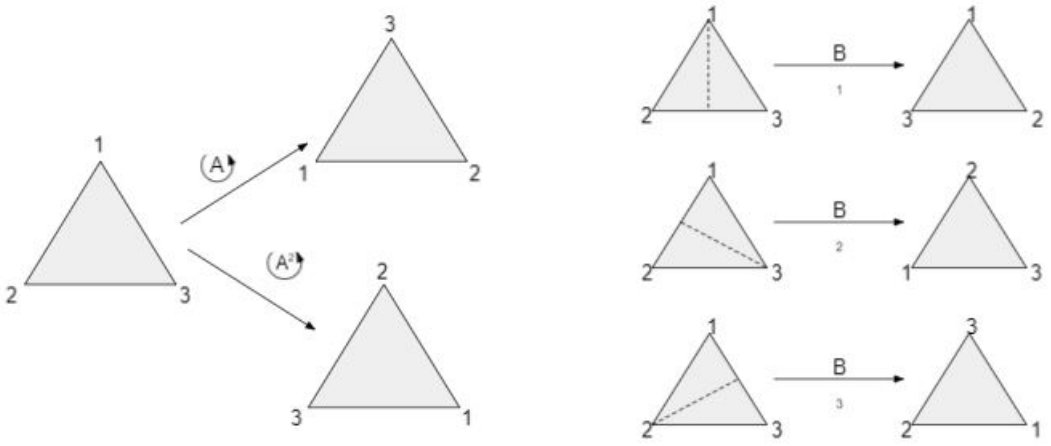

FIGURA 1.1: Rotaciones de un triángulo equilatero

Por lo que acabamos de ver, $S_3 = \{I, A, A^2, B_0, B_1, B_2\}$ y podemos encontrar los resultados de sus operaciones en la tabla de Cayley 1.2.

∘	I	A	A^2	B	AB	A^2B
I	I	A	A^2	B	AB	A^2B
A	A	A^2	I	AB	A^2B	B
A^2	A^2	I	A	A^2B	B	AB
B	B	A^2B	AB	I	A^2	A
AB	AB	B	A^2B	A	I	A^2
A^2B	A^2B	AB	B	A^2	A	I

FIGURA 1.2: Tabla de Cayley

1.1.3 Subgrupos

Dentro del conjunto de elementos que forman un grupo G, podemos encontrar subconjuntos. Se dice que un subconjunto H es subgrupo de $(G, \cdot)$ y se escribe $(H, \cdot) \leq (G, \cdot)$ cuando H es un grupo junto con la operación binaria $\cdot$ definida en G y cumple las siguientes propiedades:

- La operación binaria es **cerrada** en el subgrupo H.

$$\forall\, h_1, h_2 \in H \rightarrow h_1 \cdot h_2 \in H$$

- El elemento **neutro** del grupo G forma parte del subgrupo H.

$$e \in G \rightarrow e \in H$$

- El **inverso** de cualquier elemento del subgrupo H también pertenece al grupo.

$$\forall\, h^{-1} \in H \rightarrow h^{-1} \in G$$

Como la operación binaria $\cdot$ ya es asociativa en G, lo será también en H, y por lo tanto, no es necesario comprobarlo.

Ejemplo 1.1.8. El grupo de los números enteros $\mathbb{Z}$ y la operación suma es un subgrupo de los números racionales $\mathbb{Q}$, que a su vez es subgrupo de los números reales $\mathbb{R}$.

Otra manera de representar que un conjunto es subgrupo de otro aplicada al ejemplo 1.1.8:

$$(\mathbb{Z}, +) \leq (\mathbb{Q}, +) \leq (\mathbb{R}, +)$$

Ejemplo 1.1.9. El conjunto de los múltiplos enteros del número natural n, es decir $n\mathbb{Z} = nx : x \in \mathbb{Z}$, se tiene que (n$\mathbb{Z}$,+) es subgrupo de ($\mathbb{Z}$,+).

No es necesario demostrar las tres condiciones anteriores para verificar que H es un subgrupo de (G, $\cdot$) si para $h, g \in H$ se cumple que:

$$h \cdot g^{-1} \in H$$

Demostración 1.1.10. Suponiendo que H es un subgrupo de G. Dados los elementos $x, y, y^{-1} \in H$, por lo tanto $x \cdot y^{-1} \in H$. A partir de la segunda propiedad, $x \cdot x^{-1} = e \in H$, que nos lleva a $e \cdot x^{-1} = x^{-1} \in H$. Finalmente, $x \cdot y = x \cdot (y^{-1})^{-1} \in H$. Con esto, quedan probadas las tres propiedades y H es un subgrupo de G.

Todos los grupos tienen al menos dos subgrupos, que son conocidos como **subgrupos impropios**, el del elemento neutro y el de todos los elementos del grupo. Al resto de subgrupos se les denomina **subgrupos propios**.

Es posible obtener subgrupos mediante cualquier subconjunto S de G. $\langle S \rangle$ será el subgrupo más pequeño de (G, $\cdot$) que contiene a S.

$$\langle S \rangle = x_1^{\alpha 1}, x_2^{\alpha 2}, ..., x_n^{\alpha n} : x_1, x_2, ..., x_n \in S, \alpha_1, \alpha_2, ..., \alpha_n \in \mathbb{Z}$$

Ejemplo 1.1.11. Sea el grupo ($\mathbb{Z}_6$, $\cdot$) y el subconjunto [2], el grupo generado por dicho subconjunto es $\langle [2] \rangle$ = [0], [2], [4].

1.1.3.1 Subgrupo cíclico

Un grupo $(G, \cdot)$ se dice que es cíclico si al menos un elemento x del grupo G, puede llegar a generar el grupo G completo, es decir, $\langle s \rangle = G$. En este caso, el elemento x es conocido como generador de G.

Ejemplo 1.1.12. El grupo ($\mathbb{Z}_4$,+) es cíclico y los elementos [1] y [3] son generadores ya que con ellos se pueden obtener el grupo completo:

$$[1],\ [1]+[1]=[2] \qquad [1]+[1]+[1]=[3] \qquad [1]+[1]+[1]+[1]=[0]$$

$$[3],\ [3]+[3]=[2] \qquad [3]+[3]+[3]=[1] \qquad [3]+[3]+[3]+[3]=[0]$$

El orden de un elemento generador x es el número de elementos que posee el subgrupo H formado por dicho elemento, siempre que sea finito. Se representa $|\langle S \rangle|$.

Ejemplo 1.1.13. El orden del elemento [1] del grupo ($\mathbb{Z}_6$,+) es 6.

Teorema 1.1.14. *Para un grupo finito $(G, \cdot)$ y su elemento x, podemos encontrar un entero positivo n, de manera que $x^n = e$ y el subgrupo generado por x sea $\langle x \rangle = \{x, x^2, ..., x^{n-1}, x^n = e\}$.*

Demostración 1.1.15. Como el grupo G es finito, el conjunto de las potencias de x debe tener repeticiones, es decir, existen enteros i,j tal que $x^i = x^j$. De aquí se deduce que:

$$e = x^i \cdot x^{-i} = x^j \cdot x^{-i} = x^{j-i}$$

Podemos suponer que $j > i$ y tomar $n = j - i$ y el subconjunto generado por:

$$\langle x \rangle = \{x, x^2, ..., x^k, ..., x^{-1}, x^{-2}, ..., x^{-k}, ...\}$$

Si $k > n$, tenemos $k = cn + r$ donde $c, r \in \mathbb{N}$. Entonces $x^k = x^{cn} \cdot x^r = x^r$. Esto nos dice que los elementos x^k (k > n) se pueden eliminar en $\langle x \rangle$ porque están repetidos. Los elementos de la forma x^{-i} también aparecen repetidos ya que $-i = cn + r$ para $c, r \in \mathbb{Z}$.

Cuando $(G, \cdot)$ es un grupo finito y $x \in G$, tomando n como el menor entero positivo que cumple $x^n = e$, encontramos todos los elementos $x, x^2, ..., x^{n-1}, x^n = e$ distintos.

A partir de la explicación anterior, podemos demostrar que el orden de x coincide con el menor entero positivo k tal que $x^k = e$.

Ejemplo 1.1.16. A partir del grupo ($\mathbb{Z}_4$,+):

$$\text{orden de } [0] = 1 \qquad \text{orden de } [1] = 4 \qquad \text{orden de } [2] = 2 \qquad \text{orden de } [3] = 4$$

Teorema 1.1.17. *Cuando un elemento x del grupo finito G de orden k finito y m entero positivo, cumple que $x^m = e$, su orden k divide a m.*

Demostración 1.1.18. Si $x^m = e$, escribimos $m = ck + r$ donde $r = x^m = x^{ck} \cdot x^r = (x^k)^c \cdot x^r = (e)^e \cdot x^r = x^r$. Como $0 \leqslant r \leqslant k$ y k es el menor entero que cumple $x^k = e$, se deduce que $r = 0$ y $m = ck$, lo que prueba que k divide a m.

1.1.4 Teorema de Lagrange

En el ejemplo 1.1.16 hemos visto que el orden de los elementos de $(\mathbb{Z}_4,+)$ es 1, 2 y 4 y el del grupo es 4, por lo que todos los órdenes son divisores del orden del propio grupo. Se puede comprobar que en otros sucede lo mismo, como en $(\mathbb{Z}_6^*,+)$, cuyo orden es 5 y los de sus generadores son 1 y 5. Esta observación puede ser cierta para cualquier grupo finito.

Teorema 1.1.19. *Sea G un grupo finito y x un elemento del grupo, el orden de x es divisor del orden de G. Desde esta afirmación, se infiere que el número de elementos de un subgrupo H de G, divide al número de elementos de G.*

Demostración 1.1.20. Sea k el orden de x y n el número de elementos de G, por lo que pudimos ver en el apartado de subgrupos cíclicos, sabemos que

$$\langle x \rangle = x, x^2, ..., x^{n-1}, x^n = e$$

y que las potencias de x en el conjunto $\langle x \rangle$ no se repiten. Si partimos el grupo G tantas veces como sea necesario hasta obtener subconjuntos m que posean el mismo número de elementos, es decir, el orden de G. Si operamos cada subconjunto m con $\langle x \rangle$, obtenemos subconjuntos de tamaño igual al orden de $\langle x \rangle$. Entonces tenemos que $|\langle x \rangle| m = |G| = n$, por lo que demostramos que el orden del grupo es divisible por el del elemento.

La demostración anterior está ligada con el concepto de partición de un conjunto, que a su vez, está relacionado con relaciones de equivalencia. Por otro lado, esencialmente no está ligada a subgrupos de la forma de $\langle x \rangle$, sino que cualquier subgrupo H de G lo cumple. A partir de esto, podemos llegar a la afirmación **"el número de elementos de un subgrupo H de G divide al número de elementos de G"**. Esta afirmación es conocida como el Teorema de Lagrange (1.1.23) y lo demostraremos más adelante.

Para entender mejor el teorema, primero vamos a conocer la clases por la izquierda y por la derecha. Si H es un subgrupo de G, se definen como:

$$gH = \{gh : h \in H\} \text{ como clase por la izquierda}$$

$$Hg = \{hg : h \in H\} \text{ como clase por la derecha}$$

Ejemplo 1.1.21. Para el grupo $(\mathbb{Z},+)$ y su subconjunto de los múltiplos de 5 ($5\mathbb{Z}$), tenemos $5\mathbb{Z} = \{2 + 5z : z \in \mathbb{Z}\} = 7 + 5\mathbb{Z}$.

Dos elementos $x, y \in G$ se relacionan con H de manera que $x \equiv y(H)$, si $x^{-1}y \in H$. Esto es conocido como una relación de equivalencia y la clase de equivalencia del elemento x de G, coincide con la clase xH. Además, el número de elementos de xH, coincide con el de H.

Ejemplo 1.1.22. La relación de equivalencia que define al subgrupo $5\mathbb{Z}$ es $n \equiv m(5\mathbb{Z})$, siempre que $-m + n \in 5\mathbb{Z}$.

Teorema 1.1.23 (Teorema de Lagrange)**.** *Si G es un grupo finito y H un subgrupo de G, el número de elementos de H divide al número de elementos de G.*

Demostración 1.1.24. Las clases de equivalencia de esta relación establecen una partición de G, que coinciden con xH:

$$G = (x_1H) \cup (x_2H) \cup ... \cup (x_nH)$$

Cada conjunto x_iH tiene $|H|$ elementos, por lo tanto:

$$|G| = |x_1H| + |x_2H| + ... + |x_mH| = m|H|$$

El número entero m tal que $|G| = m|H|$, se denomina índice de H y se denota como $[G : H]$. Los xH que hemos visto anteriormente, son las clases de equivalencia por la izquierda módulo H que tiene el conjunto. De igual manera, pueden definirse Hx como las clases por la derecha.

Antes de avanzar a otro apartado, es importante conocer que un grupo G con orden primo p, siempre va a ser un grupo cíclico. Por el Teorema de Lagrange (1.1.23), el orden de los elementos del grupo tiene que dividir a p. Los únicos divisores posibles de un primo, son el 1 y el propio p. Como $|\langle x\rangle| > 1$, se tiene que $|\langle x\rangle| = p$ y x es generador de G. Por lo tanto, se trata de un grupo cíclico ya que tiene elemento generador.

1.1.4.1 Subgrupo normal

Un subgrupo H de un grupo G se dice normal y lo escribiremos como $H \lhd G$, cuando se cumpla:

$$(xH)(yH) = (xy)H \quad \forall x, y \in G$$

Ejemplo 1.1.25. En $(\mathbb{Z},+)$ el subgrupo $5\mathbb{Z}$ es normal ya que para todo m,n enteros:

$$(m+5\mathbb{Z})+(n+5\mathbb{Z}) = (m+n)5\mathbb{Z} = \{m+5z_1+n+5z_2 : z_1, z_2 \in \mathbb{Z}\} = \{m+n+5z : z \in \mathbb{Z}\} = m+n+5\mathbb{Z}.$$

Otra manera de saber si un subgrupo es normal en G es si se cumple $xN = Nx$ para todo $x \in G$, es decir, las coclases a ambos lados, coinciden. Si G es un grupo finito y H es un subgrupo de G, el orden de H divide al orden de G, es decir, $\dfrac{|H|}{|G|}$.

Al dividir G en clases de equivalencia $x_1H, x_2H, ..., x_kH$, se obtendrá un número finito k, de modo que el orden de G, es la suma de los órdenes de cada clase de equivalencia, pero todas las clases de equivalencia tienen orden $|H|$, por lo tanto, $|G| = k|H|$.

En general, no todo subgrupo es normal, pero todos los subgrupos de un grupo abeliano, son normales.

1.1.5 Grupo cociente

En el Teorema de Lagrange (1.1.23) hemos conocido las clases de equivalencia para ver su demostración. El grupo cociente G/H es el conjunto de clases de equivalencia que se obtienen al definir en G la relación de equivalencia modulo H.

$$x \equiv y(H) \text{ si y sólo si } x^{-1}y \in H$$

Una vez definido el conjunto cociente, habría que asegurar que cumple las condiciones de estructura de grupo con la operación binaria definida en G, pero esto no sucede con cualquier subgrupo, solamente con los normales.

Teorema 1.1.26. *Sea G un grupo y H un subgrupo normal de G, la operación*

$$(xH)(yH) = (xy)H$$

define en el conjunto cociente G/H la estructura de grupo. Recibe el nombre de ***grupo cociente de G sobre H.***

Demostración 1.1.27. La operación de clases de equivalencia tiene que ser verificada para asegurar que está bien definida en G/H, es decir, si x', y', $x'y'$ son representantes de $xH, yH, (xy)H$, respectivamente. Como $x' \in xH$, $y' \in yH$, $x'y' \in (xH)(yH)$ ya que H es un subgrupo normal de G. La asociatividad de la operación es una consecuencia de la asociatividad de G, puesto que:

$$((xH)(yH)(zH)) = ((xy)H)(zH) = (xy)zH = (xH)((yH)(zH))$$

El elemento neutro es $H = eH$, puesto que:

$$(xH)(eH) = (xe)H = xH \quad \text{y} \quad (eH)(xH) = (ex)H = xH$$

Por último, para $xH \in G/H$, tenemos que su inverso $x^{-1} \in G/H$, puesto que:

$$(xH)(x^{-1}H) = (xx^{-1})H = eH = H \quad y \quad (x^{-1}H)(xH) = (x^{-}1x)H = eH = H$$

G/N está formado por el conjunto de las coclases tal que:

$$G/N = (Ng \,|\, g \in G)$$

De modo que si N es subgrupo normal de G, G/N con la operación de multiplicación de subconjuntos de G, tiene estructura de grupo con las siguientes propiedades:

- $\forall x, y \in G, (Nx) \cdot (Ny) = N \cdot x \cdot y$
- N es el elemento neutro de G/N.
- $(Nx) - 1 = N \cdot x - 1 \ \ \forall x \in G$.

Los grupos cocientes serán muy útiles debido a que conseguiremos conjuntos mucho más pequeños que tengan estructura de grupo.

1.1.6 Homorfismos de grupos

La teoría de grupos contiene tres teoremas de isomorfía (1.1.30, 1.1.31, 1.1.32) que relacionan a los grupos con sus grupos cocientes y sirven para construir isomorfismos. Durante esta sección introduciremos los homomorfismos y estudiaremos algunas propiedades previas a los teoremas.

Un homomorfismo es una aplicación que conserva las operaciones de los grupos entre los que está definida.

$$f : G_1 \to G_2$$

Teorema 1.1.28. *Sean $(G_1, \cdot)$ y $(G_2, \circ)$ dos grupos y f una aplicación de G_1 en G_2, f es un **homomorfismo** si para todo $x, y \in G_1$:*

$$f(x \cdot y) = f(x) \circ f(y)$$

Ejemplo 1.1.29. Si G es un grupo abeliano y a un elemento de G, $f(a) = a^2$ define un homomorfismo de A en A:

$$f(ab) = (ab)^2 = abab = a^2b^2 = f(a)f(b)$$

Si f es una aplicación biyectiva, es decir, si todos los elementos del conjunto de salida tienen una imagen distinta en el conjunto de llegada, diremos que f es un **isomorfismo**.

Los isomorfismos nos sirven para definir igualdades entre grupos de diferentes elementos, tendrán el mismo comportamiento y para su estudio podemos tratarlos como equivalentes. También se pueden definir entre homomorfismos, pero como la aplicación no es biyectiva, no son tan fuertes.

Dado un homomorfismo f entre los grupos G_1 y G_2, se define el núcleo mediante:

$$ker(f) = x \in G_1 : f(x) = e_2$$

Es decir, el núcleo está formado por los elementos x de G_1 que tiene como imagen el elemento neutro de G_2. El núcleo del homomorfismo juega un papel importante en el desarrollo de los teoremas de isomorfía que estudiaremos a continuación.

1.1.7 Teoremas de isomorfía

Una de las técnicas fundamentales en la teoría de grupos, es la relación existente entre subgrupos normales y homomorfismos. Esta relación está recogida en el primer teorema de isomorfía, que es uno de los resultados principales de esta sección.

Teorema 1.1.30 (Primer teorema de isomorfía)**.** *Un homomorfismo suprayectivo $f : G_1 \to G_2$ y su núcleo es ker(f), cumple que:*

- *El grupo cociente de $G_1/ker(f)$ es isomorfo al otro grupo G_2.*
- *Entre los subgrupos de G_1 y G_2 que contienen a ker(f), existe una correspondencia biyectiva.*

Teorema 1.1.31 (Segundo teorema de isomorfía)**.** *Sea G un grupo, H y K subgrupos normales de G y H subgrupo de K. Se cumple que K/H es un subgrupo normal de G/H y:*

$$G/H \Big/ K/H \approx G/K$$

Teorema 1.1.32 (Tercer teorema de isomorfía)**.** *Sea G un grupo y H y K subgrupos de G con K normal en G. Entonces HK es un subgrupo de G, K es normal en HK y $H \cap K$ es normal en H. Además:*

$$HK \Big/ K = H \Big/ (H \cap K)$$

1.2 Teoría de anillos

Anteriormente hemos analizado el conjunto de elementos conocido como grupo y sus diferentes propiedades. A lo largo de esta sección, nos centraremos en conocer un nuevo conjunto que se conoce como anillo y cumple ciertas características similares a los grupos. Para consultar información más extensa y todas demostraciones de los teoremas que se exponen a lo largo de esta sección, mediante la lectura del capítulo 6 del libro *Un curso de álgebra* [Nav16].

1.2.1 Definición de anillo

Un anillo es un conjunto $(R, +, \cdot)$ no vacío con dos operaciones binarias:

$$r + s \text{ (adición)} \qquad r \cdot s \text{ (multiplicación)}$$

que satisface las siguientes propiedades:

- $(R, +)$ es un grupo abeliano. Al elemento neutro de $(R, +)$, que sabemos que es único, se le denominará elemento cero (o cero) y se denotará por 0.

- La operación binaria $\cdot$ (conocida como producto) es interna y asociativa en R

$$(r \cdot s) \cdot t = r \cdot (s \cdot t) \ \ \forall r, s, t \ \in R$$

- Se verifica la propiedad distributiva de $\cdot$ respecto de $+$, para $r, s, t \in R$:

$$(r + s) \cdot t = rt + s \cdot t, \qquad r \cdot (s + t) = r \cdot s + r \cdot t$$

Ejemplo 1.2.1. El conjunto de los números enteros con las operaciones adición y multiplicación $(\mathbb{Z}, +, \cdot)$ es un anillo.

Ejemplo 1.2.2. El conjunto de los números enteros módulo 6 con las operaciones adición y multiplicación $(\mathbb{Z}_6, +, \cdot)$ también es un anillo.

Al igual que ocurría con los grupos, hay conjuntos más pequeños dentro de los anillos que cumplen unas condiciones para considerarlos **subanillos**. Decimos que S es un subanillo de R si contiene a1 elemento neutro de la multiplicación, es decir, es unitario y la multiplicación y resta de cualquier par de elementos de S, también pertenece a S.

$$\forall s, t \in S \leftrightarrow s - t, s \cdot t \in S$$

Ejemplo 1.2.3. El conjuntos de los números enteros $\mathbb{Z}$ es un subanillo del conjunto de los racionales $\mathbb{Q}$.

1.2.2 Tipos de anillos

1.2.2.1 Anillo con identidad

Un anillo R se conoce como **anillo con identidad** cuando tiene un elemento 1 tal que $1 \cdot r = r \cdot 1 = r$ para todos elementos del anillo, es decir, algunos anillos tienen un elemento neutro para la multiplicación. Todos los anillos sobre los que trabajaremos a partir de ahora son anillos con identidad.

1.2.2.2 Anillo conmutativo

Al igual que ocurría con los grupos abelianos, que eran aquellos que cumplían la propiedad conmutativa, ocurre lo mismo en los **anillos conmutativos**. Un anillo R en el que:

$$rs = sr \ \forall \ \text{r,s} \in R$$

Ejemplo 1.2.4. Los números racionales, reales, y complejos forman anillos conmutativos con las operaciones habituales.

1.2.2.3 Anillo de polinomios

Un anillo bastante común es el **anillo de polinomios** $R[x]$, que está construido sobre un anillo R y se forma a partir de polinomios p:

$$p = \sum_{n \geq 0} a_n x^n = a_0 + a_1 x + ... + a_m x^m$$

Se conoce como grado del polinomio al valor de m, sus coeficientes son a_0, a_1, ..., a_m y su coeficiente director es a_m. En caso de que el coeficiente director sea 1, el polinomio es mónico.

El conjunto de todos los polinomios con coeficientes en R se define como $R[x]$ y las operaciones entre los polinomios $p(x) = \sum_{n \geq 0} a_n x^n$, $q(x) = \sum_{n \geq 0} b_n x^n$ son:

- Suma:
$$p(x) + q(x) = \sum_{n \geq 0} (a_n + b_n) x^n$$
- Producto:
$$p(x) \cdot q(x) = \sum_{n \geq 0} (\sum_{i+j=n}) a_i \cdot b_j) x^n$$

Diremos que los anteriores polinomios son iguales si y sólo si $a_n = b_n$.

Ejemplo 1.2.5. El anillo $\mathbb{Z}[x]$ está formado por los polinomios de coeficientes enteros como:

$$2 - x + 4x^5 \qquad\qquad -15 + x^8 - 2x^{21} + 7x^{33}$$

1.2.2.4 Dominios de integridad

Un anillo conmutativo no cero R es un **dominio de integridad** cuando $\forall r, s \in R, rs = 0$ implica que $r = 0$ ó $s = 0$, lo que significa que en el anillo no existe ningún elemento que sea divisor de 0.

Ejemplo 1.2.6. El conjunto de los números enteros $\mathbb{Z}$ es un dominio de integridad.

Si R un anillo, se dice que $u \in R$ es una unidad, si existe $u' \in R$, tal que $u \cdot u' = u' \cdot u = 1$, es decir, un elemento es una unidad si tiene un elemento inverso para la operación producto. Cuando todos los elementos tienen inverso respecto al producto, R es un **cuerpo**. En la siguiente sección estudiaremos en profundidad los cuerpos.

1.2.3 Ideales y teoremas de isomorfía

De alguna manera, es posible relacionar el concepto de los subgrupos normales en la teoría de grupos al de los ideales en la teoría de anillos. Si R es un anillo, un subgrupo I de R será ideal si el producto de un cualquier elemento de I con uno de R, es también un elemento de I. Es decir:

$$r \in R, i \in I \rightarrow xr, rx \in I$$

Ejemplo 1.2.7. 0 y R son ideales de cualquier anillo y se conocen como ideales triviales.

Ejemplo 1.2.8. El conjunto de los múltiplos de $\mathbb{Z}$ ($n\mathbb{Z}$) es un ideal.

En anillos conmutativos, los ideales no son más que un conjunto de múltiplos (r) de un elemento del anillo R:

$$(r) = Rr = \{sr | s \in R\}$$

Teorema 1.2.9. *A partir de un ideal I de R, se obtiene el grupo abeliano R/I que es un anillo con la multiplicación:*

$$(r + I)(s + I) = rs + I$$

Demostración 1.2.10. Si $r, r', s, s' \in R$ son tales que $r + I = r' + Iys + I = s' + I$, veamos que $rs + I - r's' + I$. Sabemos que $r - r'ys - s'$ están en I. Así:

$$rs - r's' = rs - r's + r's - r's' = (r - r')s + r'(s - s') \in I$$

El grupo abeliano R/I es conocido como anillo cociente y tiene estructura de dominio de integridad. En próximos apartados utilizaremos el anillo cociente $Z_n = \mathbb{Z}/n\mathbb{Z}$, es decir, el conjunto formado por el anillo $\mathbb{Z}$ y su ideal $n\mathbb{Z}$ (múltiplos de un entero positivo n). Es muy interesante porque forma un cuerpo siempre que n sea un número primo, pero sobre esto profundizaremos más adelante.

1.2.3.1 Ideales principales

Teorema 1.2.11. *Sea R un anillo conmutativo y el ideal I de $R[x]$, existe un polinomio mónico que pertenece al cuerpo, tal que $I = (p)$, es decir, un ideal principal es un ideal generado por un único elemento.*

Ejemplo 1.2.12. El conjunto de todos los múltiplos de 3 es el ideal principal generado por 3, debido a que un entero n es múltiplo de 3 cuando hay un número entero k tal que $n = k \cdot 3$.

Cuando todos los ideales de un anillo son principales, el anillo R es conocido como un **dominio de ideales principales (DIP)** y todos sus elementos se pueden factorizar como producto de los elementos primos del anillo. En estos dominios existe siempre el concepto de máximo común divisor y el mínimo común múltiplo, hecho que no ocurre en los dominios de integridad en general.

Ejemplo 1.2.13. El anillo $\mathbb{Z}$ de los números enteros.

1.2.3.2 Ideales primos

Diremos que un ideal $I \subset A$ es un ideal primo cuando I no sea lo mismo que R y se verifica la siguiente condición: el producto de dos elementos del anillo pertenece al ideal si y sólo si alguno de los elementos pertenece al ideal, es decir, si x, y $\in A, xy \in I \rightarrow x \in I$ y $\in I$.

Sea $I \subset R$ un ideal. Las propiedades siguientes son equivalentes:

- I es un ideal primo (demostración 1.2.14).
- El anillo cociente R/I es un dominio de integridad (demostración 1.2.15).

Demostración 1.2.14. R/I es un dominio de integridad donde I es el ideal primo de R:

Tomamos $a, b \in R$ tal que $a \cdot b \in I$, entonces $[a + I] \cdot [b + I] = [a \cdot b + I] = I$ (sabiendo que I es el elemento neutro del anillo R/I). Como R/I es dominio de integridad, se cumple que $[a + I] = I$, o bien, $[b + I] = I$. Por lo tanto, también se cumple que $a \in I$ o bien $b \in I$. Esto nos lleva a que I es un ideal primo.

Demostración 1.2.15. I es un ideal primo de $R \rightarrow R/I$ es dominio de integridad:

Sea $[a+I]\cdot[b+I] = I$ para un par de elementos $[a+I]$, $[b+I]$ de R/I, tenemos que $I = [a+I]\cdot[b+I] = [a \cdot b + I] \rightarrow a \cdot b \in I$. Como I es primo, tendremos que $a \in I$, o bien ,$b \in I$, y por tanto, $[a + I] = I$, o bien, $[b + I] = I$. Esto implica que R/I es dominio de integridad.

Ejemplo 1.2.16. Dado un anillo R formado por el conjunto de los números enteros $\mathbb{Z}$, los ideales primos de R son los ideales de la forma $\mathbb{Z}p$, donde p es un número primo.

1.2.3.3 Ideales maximales

Diremos que un ideal $I \subset R$ es un ideal maximal si $I = R$ y el único ideal de R que contiene a I es el propio R. Es importante saber que todo ideal maximal es primo y que todo ideal propio I de R, está contenido en algún ideal maximal.

Sea $I \subset R$ un ideal. Las propiedades siguientes son equivalentes:

- I es un ideal maximal.
- El anillo cociente R/I es un cuerpo.

Ejemplo 1.2.17. En $\mathbb{Z}$ todo ideal primo es maximal. Más generalmente, todo ideal primo de un Dominio Ideal Principal (DIP) es maximal.

Ejemplo 1.2.18. El conjunto $P = \{0, 2, 4, 6, 8, 10\}$ es un ideal primo y maximal en $\mathbb{Z}_{12}$.

1.2.3.4 Homomorfismo de anillos

Durante la teoría de grupos hemos comprendido que es un homomorfismo de grupos y el **homomorfismo de anillos** es prácticamente lo mismo. Concretamente, es una función entre objetos matemáticos con la misma estructura algebraica que mantienen las operaciones definidas. En este caso, un homomorfismo entre anillos preserva las operaciones de adición y multiplicación en el anillo. Dados dos anillos R y S, tendremos un homomorfismo de anillos si se cumple que $f : R \to S$, $f(xy) = f(x)f(y)$ y $f(1) = 1$.

Dentro de los diferentes tipos de homomofismos, cuando la función f es biyectiva (todos los elementos del conjunto de salida tienen una imagen distinta en el conjunto de llegada, y a cada elemento del conjunto de llegada le corresponde un elemento del conjunto de salida) se dice que f es un **isomorfismo de anillos**.

Es posible trabajar con el homomorfismo de un anillo R y su ideal I, a partir de la aplicación $f : R \to R/I y f(r) = r + I$.

Teorema 1.2.19 (Teorema de isomorfismo de anillos)**.** *El homomorfismo de anillos $f : R \to S$, donde $f(R)$ es un subanillo de S y $ker(f)$ es un ideal de R y existe la aplicación $\overline{f} = R/ker(f) \to f(R)$ dada por $\overline{f}(x + ker(f)) = f(x)$ es un isomorfismo de anillos.*

Demostración 1.2.20. Como f es un homomorfismo de grupos, sabemos que:

$$K = ker(f) = \{x \in R | f(x) = 0\}$$

es un subgrupo normal de R y la aplicación $\overline{f}$ está bien definida y es un isomorfismo de grupos. También, $f(R)$ es un subgrupo de S. Como $f(xy) = f(x)f(y)$ y $f(1) = 1$, está claro que $f(R)$ es un subanillo de S. Veamos que K es un ideal de R. Si $r \in R$ y $k \in K$, tendremos que:

$$f(rk) = f(r)f(k) = 0 = f(k)f(r) = f(kr)$$

Con lo que deducimos que $rk, kr \in K$. Finalmente:

$$\overline{f}((x + K)(y + K)) = \overline{f}(xy + K) = f(xy) = f(x)f(y) = \overline{f}(x + K)\overline{f}(y + K), \overline{f}(1 + K) = 1$$

y el teorema queda probado.

El elementos del homomorfismo de anillos que tiene como imagen el elemento neutro de la suma, juegan un papel fundamental en la teoría de anillos. Para cualquier homomorfismo $f : R \to S$ definimos el núcleo de un homomorfismo de anillos como el conjunto:

$$ker(f) = \{r \in R : f(r) = 0\}$$

El núcleo de un homomorfismo es importante porque siempre es un ideal.

Ejemplo 1.2.21. Un importante homomorfismo de anillos es el homomorfismo de evaluación. Si $p(x) = a_0 + a_1x + ... + a_mx^m \in R[x]$ y $a \in R$, definimos:

$$p(a) = a_0 + a_1a + ... + a_ma^m \in R$$

Teorema 1.2.22 (Teorema de evaluación)**.** *La aplicación $e_a : R[x] \to R$ dada por $e_a(p) = p(a)$ es un homomorfismo de anillos si R es un anillo conmutativo y $a \in R$. Gracias a esto, podemos decir que a será una raíz de p en R siempre que se cumpla $p(a) = 0$.*

Demostración 1.2.23. Observamos que $e_a(1) = 1$. Para dos polinomios $p(a)$ y $q(a)$, por definición tenemos que $(p + q)(a) = p(a) + q(a)$, y además, $pq(a) = p(a)q(a)$. Esto es inmediato.

En los anillos de polinomios también nos encontramos con un algoritmo para realizar la división, al igual que para el conjunto de los números enteros.

Teorema 1.2.24 (Teorema del algoritmo de división)**.** *Dados $f, g \in R[x]$ no cero, suponiendo que el coeficiente director de g es una unidad del anillo R, existen dos polinomios únicos $d, r \in R[x]$ tal que $f = dg + r$.*

Gracias al anterior teorema (1.2.24), podemos definir el máximo común divisor de dos polinomios. Además, dos polinomios son coprimos cuando su máximo común divisor es un polinomio constante. El objetivo final de todo esto, es conseguir polinomios irreducibles, es decir, lograr expresar un polinomio en varios de menor grado, ya que los irreducibles se comportan como los elementos primos, cuyo máximo común divisor con cualquier otro elemento es 1.

1.3 Teoría de cuerpos

El cuerpo es la última entidad matemática que vamos a estudiar a lo largo de este trabajo. Como hemos podido ver durante la teoría de anillos, un cuerpo es un anillo conmutativo con identidad donde todos los elementos tienen un inverso en el producto. A lo largo de esta sección vamos a profundizar en los teoremas más importantes hasta llegar a los cuerpos finitos que usaremos en la criptografía de curvas elípticas. La información sobre cuerpos descrita en este apartado, se ha desarrollado en base al contenido del capítulo 6 del libro *Un curso de algebra* [Nav16].

1.3.1 Definición de cuerpo

Un cuerpo es un sistema algebraico en el que las dos operaciones que lo forman ($+$ y $\cdot$ conocidas como adición y multiplicación respectivamente) se pueden realizar y se cumple que la multiplicación es asociativa, conmutativa y distributiva respecto de la adición. Además, posee inverso y elemento neutro para ambas operaciones, lo que permite efectuar las operaciones de sustracción y división

(excepto la división por cero). Esto lo diferencia de un anillo conmutativo en el cual no existe el elemento inverso de la multiplicación, y por lo tanto, no hay división.

Un cuerpo es un conjunto no vacío $(K, +, \cdot)$ que cumple las siguientes propiedades:

- $(K, +)$ es un grupo abeliano.
- $(K^*, \cdot)$ es un grupo abeliano, donde $K^* = K - \{0\}$.
- La operación suma será distributiva respecto al producto, es decir, para $r, s, t \in R$:

$$(r + s)t = rt + st$$

$$r(s + t) = rs + rt$$

Por lo tanto, un cuerpo siempre tendrá como mínimo dos elementos, el neutro para la suma y el neutro para el producto. Además de las anteriores propiedades, es necesario saber que todo cuerpo es tanto dominio de integridad y como anillo.

Ejemplo 1.3.1. $(\mathbb{Q}, +, \cdot)$ es un cuerpo, donde $\mathbb{Q}$ es el conjunto de los números racionales.

Ejemplo 1.3.2. $(\mathbb{R}, +, \cdot)$ es un cuerpo, donde $\mathbb{R}$ es el conjunto de los números reales.

Al igual que los anillos, los cuerpos también tienen elementos inversibles $rs = sr = 1$ conocidos como unidades.

Ejemplo 1.3.3. Las unidades de $K[x]$ son los polinomios constantes no cero. El conjunto de las unidades de un cuerpo K es K^x.

También en los cuerpos, encontramos conjuntos más pequeños que cumplen ciertas propiedades y son conocidos como subcuerpos. Si K es un cuerpo y S un subanillo S de K, S será un subcuerpo cuando todos sus elementos tienen inverso (excepto el cero).

Ejemplo 1.3.4. $\mathbb{Q}$ es un subcuerpo de $\mathbb{C}$.

1.3.2 Ideales y homomorfismos de cuerpos

El homomorfismo de cuerpos se produce cuando dos cuerpos K y S son un homomorfismo de grupos con la aplicación $f : K \to S$ tal que $f(xy) = f(x)f(y)$ y $f(1) = 1$. Si además f es biyectiva, es un isomorfismo de cuerpos.

Como todo cuerpo es un anillo, podríamos preguntarnos por la forma que tengan sus ideales. Para empezar, como todo cuerpo es anillo conmutativo, todo ideal por la izquierda es ideal y todo ideal por la derecha es también ideal. Así pues, sólo tenemos que estudiar los ideales del cuerpo.

Si I es ideal del cuerpo K, entonces todo elemento no nulo $a \in K$ ha de tener inverso, $a^{-1} \in K$, luego a es una unidad de K (esto es, $a \in U(K)$), y se tendrá que $I \cap U(K) \neq \varnothing$, es decir, $I = R$. De esta manera, los únicos ideales de un cuerpo son el propio cuerpo y el ideal nulo.

1.3.3 Extensiones de cuerpos

Un cuerpo E es una extensión de cuerpos de K si K es un subcuerpo de E. Se conoce al cuerpo K como cuerpo base y la extensión se presenta mediante E/K, donde E es un espacio vectorial sobre K.

Ejemplo 1.3.5. $\mathbb{C}$ es una extensión de $\mathbb{Q}$ y de $\mathbb{R}$.

Una extensión E/K es finita si la dimensión de E como K-espacio vectorial es finita. Como todo espacio vectorial tiene base, podemos calcularlo mediante:

$$|E : K| = dim_K(E)$$

y es conocido como el **grado de la extensión**. Por lo tanto, E/K será finita siempre que tenga un subconjunto finito $a_1, ..., a_n$

Teorema 1.3.6 (Teorema de transitividad de índices)**.** *Sea E/K una extensión y $K \subseteq L \subseteq E$ es un subcuerpo intermedio. E/K es finita siempre que E/L y L/K sean finitas. Por lo tanto:*

$$|E : K| = |E : L||L : K|$$

1.3.3.1 Elemento algebraico

Un elemento a de la extensión E/K es **algebraico** sobre K cuando existe un polinomio $p(x) \neq 0 \in K[x]$ tal que $p(a) = 0$. Cuando todos los elementos de E son algebraicos sobre K, diremos que E/K es una **extensión algebraica**.

Teorema 1.3.7. *Una extensión E/K es algebraica, siempre que es finita.*

Demostración 1.3.8. Sea $a \in E$ y supongamos que $|E : K| = n$. Queremos probar que a es raíz de algún polinomio no cero de $K[x]$. Podemos suponer que $a \neq 0$. Si existen enteros $i > j \geq 0$ tales que $a^i = a^j$, entonces a es raíz del polinomio $x^{i-j} - 1 \in K[x]$. En caso contrario, el conjunto $\{1, a, a^2, ..., a^n\}$ tiene $n + 1$ elementos, y por tanto, es K-ligado. Por tanto, existen $k_0, ..., k_n \in K$ no todos cero, tales que $k_0 + k_1 a + ... + k_n a^n = 0$. Entonces, el siguiente polinomio no cero se anula en a:

$$f(x) = k_0 + k_1 x + ... + k_n x^n \in K[x]$$

Teorema 1.3.9 (Teorema del elemento algebraico)**.** *Sea E/K una extensión y supongamos que $a \in E$ es algebraico sobre K:*

- *Existe un único polinomio mónico* $p \in K[x]$ *irreducible en* $K[x]$ *tal que* $p(a) = 0$.
- *Si* $q \in K[x]$, $q(a) = 0$ *si y sólo si* p *divide a* q.
- $K(a) = \{f(a) | f \in K[x]\}$
- *Si* $\delta(p) = n$, *entonces* $\{1, a, ..., a^{n-1}\}$ *es una* K*-base de* $K(a)$. *Por tanto,* $|K(a) : K| = \delta(p)$

El polinomio $p(x)$ es conocido como **polinomio mínimo o irreducible** de a sobre K.

Un cuerpo K es **algebraicamente cerrado** cuando no existen extensiones algebraicas propias de K, es decir, para la extensión E/K se tiene que $E = K$. Además, se conoce como clausura algebraica de K cuando se verifica que:

- E es algebraicamente cerrado.
- E/K es una extensión algebraica.

Todos los cuerpos tienen una clausura algebraica que es única salvo para K-isomorfismos.

1.3.3.2 Cuerpo de escisión

La clausura algebraica de un cuerpo K tiene como consecuencia que para cada polinomio $f(x) \in K[x]$ exista una determinada extensión de K en la que dicho polinomio, tiene todas sus raíces. El cuerpo generado por dichas raíces, es conocido como **cuerpo de escisión** de $f(x)$ sobre K es único salvo K-isomorfismos.

Teorema 1.3.10 (Teorema de existencia de cuerpos de escisión)**.** *Sean* K *un cuerpo y* $f \in K[x]$ *no constante, entonces existe un cuerpo de escisión de* f *sobre* K.

1.3.4 Cuerpos primos

El cuerpo primo F de K es la intersección de los subcuerpos de K. Todo cuerpo F contiene un cuerpo primo que es, o bien $\mathbb{Q}$, o bien $\mathbb{Z}_p = \mathbb{Z}/p\mathbb{Z}$ para algún primo p.

Sea K un cuerpo, llamamos característica de K al menor número natural n tal que la un número sumado n veces, es igual a 0. En caso de que no exista dicho número, $carK = 0$. Siempre la característica de K será igual a 0 ó a un número primo. Se llama subcuerpo primo de K a la intersección de los subcuerpos de K. Sea K un cuerpo y F su cuerpo primo. Cuando F sea isomorfo a $\mathbb{Q}$, diremos que la característica de K es 0. En cambio, cuando sea isomorfo a Z_p, diremos que la característica de K es p. Por lo tanto, un cuerpo K de característica p, se tiene que:

$$pk = k + ... + k = k(1 + ... + 1) = k(p1) = 0$$

Ejemplo 1.3.11. El cuerpo $\mathbb{Q}$ de los racionales es un cuerpo primo.

Sea K un cuerpo y sea F su cuerpo primo. Entonces F es isomorfo a $\mathbb{Q}$ ó F es isomorfo a $\mathbb{Z}_p$ para cierto primo p. En el primer caso $n1 \neq 0$ para todo $0 \neq n \in \mathbb{Z}$. En el segundo, $n1 = 0$ si y sólo si p divide a n.

1.3.5 Cuerpos finitos

Un **cuerpo finito** es un cuerpo definido sobre un conjunto finito de elemento. Tiene un amplio uso en teoría de números, geometría algebraica o criptografía y también son conocidos como **cuerpos de Galois**. Los cuerpos finitos están formados por un número de elementos $q = p^n$ (donde p es un número primo y n un entero positivo). La cardinalidad de un cuerpo finito se define de una única manera, por lo que todos los cuerpos finitos que comparten cardinalidad, son isomorfos entre sí.

Ejemplo 1.3.12. El conjunto $\mathbb{Z}_7$ formado por los enteros módulo 7 es un cuerpo finito.

Teorema 1.3.13 (Teorema de cuerpos finitos)**.** *Si K es un cuerpo finito, entonces $|K| = p^n$ para algún primo p y algún entero positivo n. Recíprocamente, dado un primo p y un entero positivo n, existe un único cuerpo de p^n elementos que denominamos como $\mathbb{F}_{p^n}$, excepto en caso de isomorfismo.*

Demostración 1.3.14. Un cuerpo primo de K es $F \cong \mathbb{Z}_p$ para algún primo p. Como K es finito, necesariamente K es un F-espacio vectorial de dimensión finita. Sea $\{a_1, ..., a_n\}$ una F-base de K. Como cada elemento de K se escribe de forma única como una F-combinación lineal de los elementos de la base, concluimos que $|K| = p^n$, donde $n = dim_F(K)$. Recíprocamente, supongamos que p es un primo y n un entero positivo. Sea $F = \mathbb{Z}_p$ y sea E un cuerpo de escisión de $f = x^{p^n} - x \in F[x]$. Sea U el conjunto de raíces de f en E. Notemos que $f' = -1$. Entonces $(f, f') = 1$ y $|U| = p^n$. Si $a, b \in U$, entonces $a - b \in U$. También, si $b \neq 0$, tenemos que $a/b \in U$. Por lo tanto, U es un subcuerpo de E. Como $E = F(U)$, necesariamente $E = U$ y $|E| = p^n$. Finalmente, supongamos que L es otro cuerpo de p^n elementos,y sea D su cuerpo primo. Por el primer párrafo de esta demostración, si $D \cong \mathbb{Z}_p$ es el cuerpo primo de L, entonces $|L|$ es una potencia de q. Por tanto, deducimos que $D \cong F$. Por consiguiente, L^x es un grupo cíclico de $p^n - 1$ elementos. De esta manera, si $l \in L^x$, tenemos que $l^{p^n-1} = 1$, que implica que $l^{p^n} = l$. Así L es cuerpo de escisión de $x^{p^n} - x$ sobre D. Por todo ello, podemos deducir que existe un isomorfismo de cuerpos entre D y F y otro entre E y L. Queda demostrado que existe el cuerpo de p^n elementos y si existen otros de la misma cardinalidad, serán isomorfos.

Construcción de cuerpos finitos

Una forma de obtener los cuerpos finitos es mediante la característica p de K, que es el orden del elemento neutro del grupo multiplicativo en el grupo aditivo, y siempre es un número primo o su potencia. Es decir, hay cuerpos de $2, 3, 4, 5, ...$ elementos, pero no los hay de $6, 10, 12, 14, ...$ elementos.

Ejemplo 1.3.15. La característica del cuerpo $\mathbb{F}_8$ es 2.

Por lo tanto, un cuerpo finito tiene p^n elementos. Además, excepto en isomorfismos, sólo existe un cuerpo de orden p^n. Para su construcción, partimos de un polinomio irreducible $p(x) \in \mathbb{Z}_p[x]$ de grado n. Esto se debe a que el conjunto de polinomios de grado menor que n, con la suma habitual de polinomios y el producto con módulo un polinomio irreducible $p(x)$ de grado n, tiene estructura de cuerpo.

El cuerpo K construido se denomina cuerpo cociente $\mathbb{Z}_p[x]/p(x)$. Como el polinomio $p(x)$ es irreducible, produce la existencia completa de los elementos inversos. Si no fuera irreducible, ciertos polinomios no serían inversibles y se trataría del anillo cociente $\mathbb{Z}_p[x]/p(x)$.

En el grupo $(K, +)$, todos los polinomios (excepto el nulo) tienen orden p, ya que los coeficientes están en $\mathbb{Z}_p$. Además, el orden de todos los elementos es p (excepto el 0). En cuanto al grupo $(K^*, \cdot)$ es un grupo conmutativo de orden $p^n - 1$ y resulta ser cíclico. Esto quiere decir que tiene generadores. Cuando el polinomio x es generador, es decir, si $K = \langle x \rangle$, entonces el polinomio $p(x)$, es un polinomio irreducible primitivo.

Ejemplo 1.3.16. En el cuerpo $\mathbb{F}_8$ el polinomio $x^3 + x^2 + 1$ es irreducible primitivo.

Capítulo 2

Criptografía de Curvas Elípticas

Durante las últimas décadas el uso de curvas elípticas ha aumentado su importancia tanto en la teoría de números como en campos relacionados directamente con la criptografía. Gracias a esta repercusión y todo lo que aportan, han llegado a ser parte de algunos estándares industriales.

Este tipo de criptografía utiliza diferentes algoritmos para cifrar o descifrar información a partir del problema del logaritmo discreto. Su rendimiento actual es capaz de garantizar una seguridad equivalente a métodos antiguos (como RSA), pero con la ventaja de trabajar con claves mucho menores que en otros casos. Esto implica una mejora significativa del rendimiento, sobretodo en para dispositivos o ámbitos donde minimizar el gasto de memoria es fundamental, como por ejemplo en tarjetas inteligentes o en la tecnología blockchain.

A lo largo de este capítulo, vamos a introducir la teoría de curvas elípticas, su uso en criptografía y diferentes algoritmos que trabajan ellas. Si desea profundizar en este tema, puede consultar el libro *Criptografía con curvas elípticas* [MEM18]. o la lectura del temario completo sobre la criptografía de curvas elípticas de Criptored [JMM]. Criptored es una Red Temática Iberoamericana de Criptografía y Seguridad, que colaboró con la Universidad Politécnica de Madrid a través del Dr. Jorge Ramió hasta su jubilación en 2021.

2.1 Curvas elípticas

Las curvas elípticas conforman un campo de las matemáticas relativamente nuevo. Hasta el año 1985 no se escucha hablar de ellas y su popularidad aumenta porque se empiezan a implementar sistemas criptográficos más difíciles de romper, cuyas claves son más pequeñas que con otros criptosistemas de clave pública convencionales.

Para la correcta comprensión de este tema, se van a explicar diferentes conceptos claves en las siguientes secciones.

2.1.1 Curva algebraica

Una curva algebraica está formada por el conjunto de puntos (x_1, y_1) que satisfacen que $p(x_1, y_1) = 0$. Una curva plana definida sobre un cuerpo K puede expresarse en el plano afín como $A^2(K)$ mediante la función $f(x, y) = 0$. El conjunto de puntos (x, y) de K^2 son las coordenadas afines o no homogéneas de un punto cualquiera de la curva. Por otro lado, la misma curva puede representarse en el plano proyectivo sobre el cuerpo K como $P^2(K)$, mediante la ecuación $f(x, y, z) = 0$. El conjunto de puntos $(x, y, z) \in K^3 - \{0, 0, 0\}$ son las coordenadas proyectivas y homogéneas de un punto de la curva.

Las coordenadas proyectivas están formadas por las relaciones de equivalencia $\sim$ tal que dos puntos (x_1, y_1, z_1) y (x_2, y_2, z_2) son equivalentes si existe un valor $\lambda \neq 0$ con el que $(x_1, y_1, z_1) = (\lambda x_2, \lambda y_2, \lambda z_2)$. Cada clase de equivalencia es un punto proyectivo y se denotan como $(x : y : z)$. + Existe solamente un punto de la forma $(x/z, y/z, 1)$. De modo que a partir de $z = 0$, es posible identificar puntos proyectivos con puntos afines.

En función del grado de los coeficientes de la curva, puede ser denominadas con un nombre particular. Las de grado 1 son conocidas como**lineales**, las de grado 2 como **cónicas** y las de grado 3, **cúbicas**. También, se diferencia las curvas según otra características de sus puntos. Un punto es singular en el caso de que las derivadas parciales se anulen en dicho punto, es decir, el punto puede ser definido por varias tangentes. Si son dos tangentes distintas, se trata de un nodo (figura 2.1), pero si es una tangente doble, se conoce como cúspide (figura 2.2).

Los puntos de la curva que se definen con una única tangente, son denominados lisos. Si todos los puntos de una curva son lisos, se tratará de una curva no singular, pero con un solo punto que no sea liso, se tratará de una curva singular.

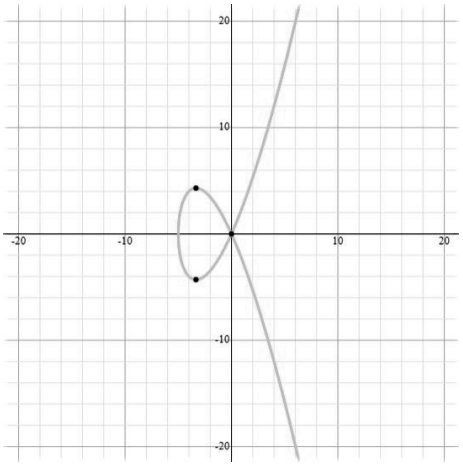

FIGURA 2.1: Curva $y^2 = x^3 + 5x^2$ sobre $\mathbb{R}$ con un nodo en (0,0).

Sea K un cuerpo, decimos que $p = (x, y)$ es un punto racional de la curva si $f(p) = 0$ y $p \in K^2$.

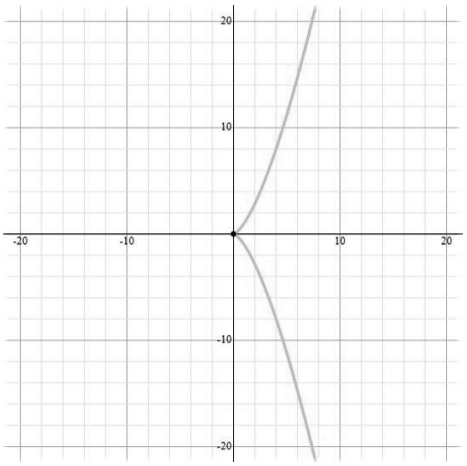

FIGURA 2.2: Curva $y^2 = 0.25x^3$ sobre $\mathbb{R}$ con una cúspide en (0,0).

2.1.2 Definición de curva elíptica

Una **curva elíptica** es una curva plana no singular de grado 3 junto con un punto racional prefijado, que denominaremos punto base. Se denomina orden o cardinalidad de una curva al número de elementos de esa curva.

2.1.2.1 Ecuación de Weierstrass

Una curva elíptica sobre un cuerpo K es una curva algebraica que se representa a partir de la ecuación general de Weierstrass:

$$y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6 \qquad a_i \in K$$

Cuando la característica del cuerpo es distinta de 2 y 3, es denominada ecuación reducida de Weierstrass y se representa:

$$y^2 = x^3 + ax + b \qquad a, b \in K$$

En la figura 2.3 podemos ver una curva elíptica representada con la ecuación reducida de Weierstrass.

Los elementos que tomamos para los valores de a, b pueden pertenecer a diferentes conjuntos, pero siempre con la estructura de cuerpo. Por ejemplo, los números reales $\mathbb{R}$, los números complejos $\mathbb{C}$, los números racionales $\mathbb{Q}$, uno de los campos finitos $F_p(= \mathbb{Z}_p)$ para un primo p, o uno de los campos finitos F_q, donde $q = pk$ con $k \geq 1$. En uno de los siguientes apartados, vamos a profundizar en cuerpos finitos, que serán en los que se va a apoyar este trabajo.

Para obtener los valores $a, b \in K$ y que la curva no tenga singularidades, se tiene que cumplir que:

$$4a^3 + 27b^2 \neq 0 \,(mod\, p)$$

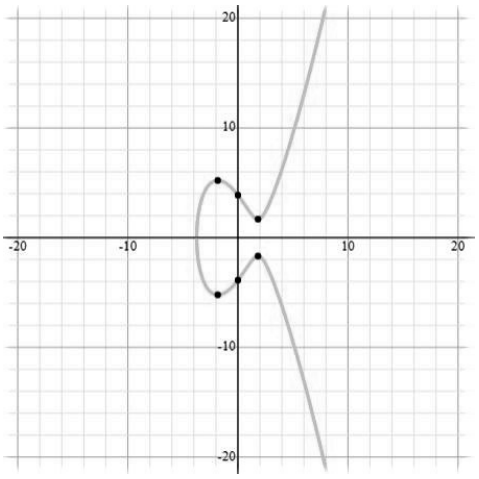

FIGURA 2.3: Curva elíptica $y^2 = x^3 - 10x + 15$ definida sobre $\mathbb{R}$.

La ecuación homogénea de Weierstrass define una curva proyectiva plana con el punto especial denominado **punto en el infinito**, $O = [0 : 1 : 0]$, que no tiene correspondencia en el plano afín.

2.1.3 Estructura de grupo

El conjunto de puntos G que forman la curva (soluciones de la ecuación y punto en el infinito O) junto con la operación aditiva $+$, forman un grupo abeliano. Una vez más es importante destacar que el problema del logaritmo discreto sobre un conjunto de puntos de un grupo abeliano finito (PLDCE) presenta un nivel de seguridad igual o superior al de los cuerpos finitos (PLD), con la ventaja extra de un menor tamaño de claves en la criptografía de curvas elípticas.

Si E/K es una curva elíptica sobre un cuerpo K, denotaremos por $E(K)$ el conjunto de puntos $P = (x, y) \in K \times K$ que satisfacen la ecuación de la curva junto con el punto del infinito de la curva O. Sobre ese conjunto de puntos se define una operación interna a partir del método de la cuerda y la tangente (figura 2.4), que consiste en trazar una recta r por los dos puntos P y Q y se calcula el tercer punto intersección R de la recta r con la curva. Lo llamamos suma elíptica y le proporciona al conjunto la estructura de grupo abeliano, cuyo elemento neutro es el punto del infinito de la curva.

Con lo que acabamos de ver, para todo punto P de la curva, podemos definir la operación suma de puntos de la siguiente manera:

1. $P + O = O + P = P$, es decir, O es el elemento neutro de la suma.
2. Existe $-P = (x_P - y_P - a_1 x_P - a_3)$, de manera que $P + (-P) = O$.
3. Dados dos puntos $P \neq Q$, se define $R = P + Q = (x_R, y_R)$ de manera que:

$$x_R = \lambda^2 + a_1\lambda - a_2 - x_P - x_Q,$$

$$y_R = \lambda(x_P - x_R) - y_P - a_1 x_R - a_3,$$

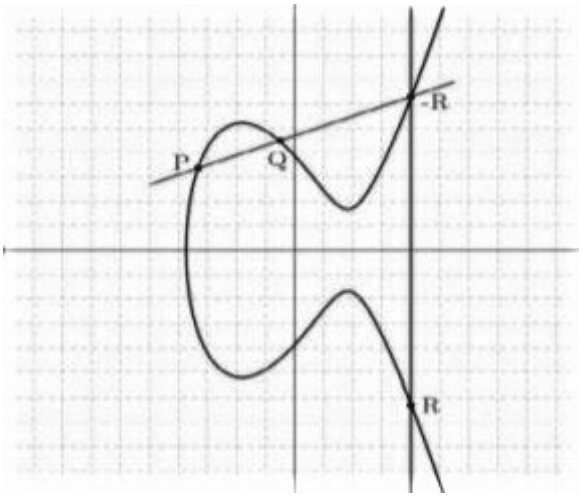

FIGURA 2.4: Ejemplo de suma de puntos $R = P + Q$

$$\lambda = \frac{y_Q - y_P}{x_Q - x_P}$$

4. La duplicación de un punto P es el punto $R = P + P = 2P$ de manera que:

$$x_R = \lambda^2 + a_1\lambda - a_2 - x_P - x_Q,$$

$$y_R = \lambda(x_P - x_R) - y_P - a_1 x_R - a_3,$$

$$\lambda = \frac{3x_P^2 + 2a_2 x_P + a_4 - a_1 y_P}{2y_P + a_1 x_P + a_3}$$

Como hemos visto en el primer punto, al sumar dos puntos que tengan el mismo valor de en la primera coordenada, la recta que pasa por ambos conjuntos no corta a la curva en ningún otro punto. Esto se debe a que la recta corta a la curva en el punto del infinito.

Finalmente, vamos a asegurar que la operación suma descrita anteriormente cumple las propiedades de grupo y tiene estructura de grupo abeliano:

1. Asociatividad: $(P + Q) = R = P + (Q + R)$ para todo $P, Q, R \in E(K)$.
2. Existencia de elemento neutro: $O : P + O$ para todo punto $P \in E(K)$.
3. Existencia de elemento inverso: $P + P' = O$ donde $P' = -P$.
4. Conmutatividad: $P + Q = Q + P$ para todo $P, Q \in E(K)$.

De manera adicional, existe otra operación en los cálculos con puntos de una curva. Consiste en el producto de un punto P por un entero positivo k, que es el equivalente de sumar el punto P una cantidad de k veces consigo mismo.

$$kP = P + ...^{(k)}... + P$$

Esta operación, que es el equivalente aditivo de la exponenciación en grupos abelianos multiplicativos, es la base de la criptografía con curvas elípticas. Además, en criptografía se trabajará con grupos finitos grandes, cuyos elementos tengan una buena representación. Así, en lugar de definir las curvas elípticas sobre cuerpos de característica 0, serán definidas sobre cuerpos finitos.

2.1.4 Curvas elípticas sobre cuerpos finitos

Como hemos visto anteriormente, el orden de un cuerpo finito $\mathbb{F}$ es el número de elementos de dicho cuerpo. Es posible demostrar que la existencia de un cuerpo finito de orden q, denominado $\mathbb{F}_q$, está relacionada con el valor de $q = p^m$, donde p es la característica del cuerpo que siempre es un número primo y m es cualquier entero positivo. Además, existe un único cuerpo, salvo isomorfismos, con cardinal p^m. Normalmente en criptosistemas de curvas elípticas se utilizan dos tipos de cuerpos finitos: el cuerpo finito primo $\mathbb{F}_q$ y el cuerpo finito binario $\mathbb{F}_{2^m}$.

Ejemplos:

$$\mathbb{F}_7 = \{0, 1, 2, 3, 4, 5, 6\}$$

$$\mathbb{F}_{2^3} = \{0, 1, x, x+1, x^2, x^2+1, x^2+x, x^2+x+1\}$$

En la figura 2.5 se puede ver la representación de los puntos con coordenadas afines y la suma de puntos en una curva sobre un cuerpo finito.

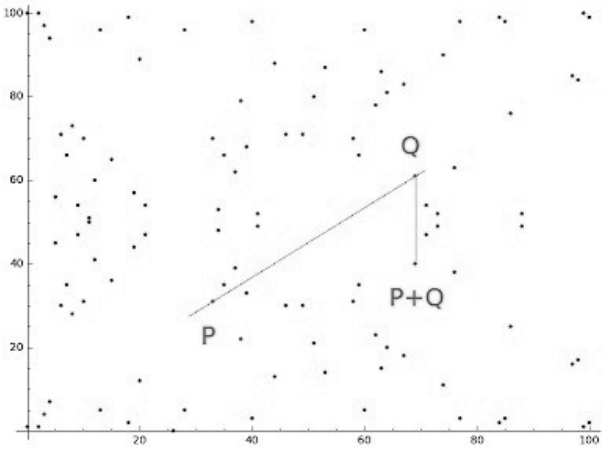

FIGURA 2.5: Ejemplo de suma de puntos $R = P + Q$ en un cuerpo finito

2.2 Uso en criptografía

En esta sección, analizaremos algunos de los criptosistemas que utilizan curvas elípticas. En su mayoría trataremos aquellos que se basan en el problema del logaritmo discreto, por lo que se incluye una explicación más profunda sobre PLD.

La criptografía de curvas elípticas proporciona una seguridad equivalente a los sistemas clásicos utilizando menos bits. Se estima que una clave tamaño de 4096 bits para RSA (algoritmo asimétrico) proporciona el mismo nivel de seguridad que 313 bits en un sistema criptografíco de curva elíptica.

Seguridad (bits)	RSA	ECC
80	1024	160-223
112	2048	224-255
128	3072	256-383
192	7680	384-511
256	15360	512+

FIGURA 2.6: Comparación entre RSA y ECC

En la tabla 2.6 se puede ver la comparación de las longitudes de las claves entre RSA y ECC., mientras que en la figura 2.7, se puede ver la comparación de rendimiento entre RSA y ECC. Con claves mucho menores, las curvas elípticas son capaces de conseguir el mismo nivel de seguridad.

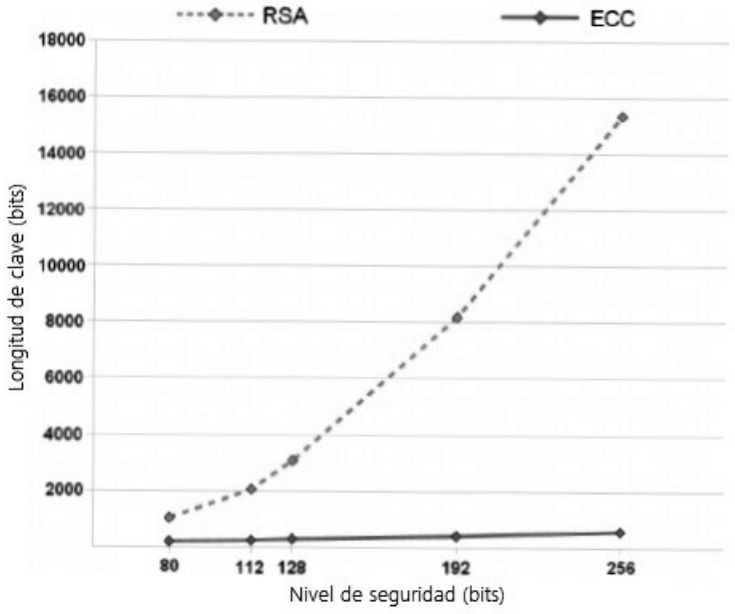

FIGURA 2.7: Comparación entre RSA y ECC

Estos datos demuestran que el uso de curvas elípticas ofrece mejor rendimiento que otras propuestas, por lo que consideramos que sus ventajas provocarán un incremento de uso en un futuro próximo.

La información descrita a lo largo de esta sección ha sido obtenida a partir de libros como *Elliptic curves: number theory and cryptography* [Was08] o *Criptografía con curvas elípticas* [MEM18].

2.2.1 Introducción

El auge de las comunicaciones digitales durante los últimos años ha provocado el aumento de manera considerable de problemas de relacionados con el ámbito de la seguridad. La información enviada a través de la red puede llegar a tener un alto valor, por lo su intercepción supone un alto nivel riesgo. La seguridad de dicha información debe garantizarse para un uso adecuado de la comunicación.

La criptografía es la parte de la criptología que se encarga del estudio de diferentes métodos (algoritmos, protocolos y sistemas) que tratan de garantizar la confidencialidad y la integridad de la información. La criptografía utiliza diferentes técnicas matemáticas que sirven como herramientas para garantizar objetivos.

Hay dos tipos básicos de cifrado. El cifrado simétrico es aquel en el que la clave de cifrado y la clave de descifrado son las mismas (o una puede deducirse fácilmente una de la otra), y el cifrado de clave pública o asimétrico, en el que tanto emisor como receptor tienen una clave pública los emisores la usen para cifrar la información y una privada para que solamente ellos pueden descifrarla.

2.2.2 El Problema del Logaritmo Discreto

La criptofrafía de curvas elípticas está basada en el Problema del Logaritmo Discreto. El hecho de que el problema no se pueda resolver en un tiempo razonable si se utiliza aritmética modular, conlleva a su amplio uso en criptografía. El logaritmo discreto de y en base g donde g, y son elementos de un grupo cíclico finito G, es la solución x de la ecuación $g^x = y$, o lo que es lo mismo:

$$x = log_g(y) \rightarrow g^x = y$$

El valor de x es difícil de averiguar cuando trabajamos con números primos de gran tamaño. El cálculo de su inversa es una tarea muy sencilla en términos computacionales, pero el cálculo del logaritmo discreto es complejo cuando se trabaja con ciertos grupos. El problema real es que no se puede resolver en un tiempo razonable si se utiliza aritmética modular.

Como se acaba de mencionar, la dificultad a la hora de resolver el problema, varía según el grupo en el que se trabaja. Por ejemplo, cuando trabajamos con el conjunto de los números enteros $\mathbb{Z}$ con la operación suma o el conjunto de los números reales sin el cero $(\mathbb{R} - \{0\})$ con la multiplicación, es demasiado fácil de romper.

Existen diferentes tipos de algoritmos que se encargan de la computación para resolver el problema del logaritmo discreto, como elevar sucesivas potencias hasta encontrar la deseada o la factorización de enteros. Estos algoritmos funcionan más rápido que el algoritmo anterior, pero ninguno corre en un tiempo polinómico. Algunos de los algoritmos funcionan para cualquier grupo, mientras otros sólo trabajan para grupos concretos. Como el objetivo de este trabajo no es el estudio de estos algoritmos,

para obtener más información se recomienda leer el capítulo 6 del libro *Cryptography: Theory and Practice* [Sti06].

La criptografía clásica trabaja con el conjunto de enteros módulo n $(\mathbb{Z}_n^*, \cdot)$, donde la operación aumenta considerablemente su dificultad. El problema con este último grupo es que no está demostrado que sea o no resoluble en un tiempo polinómico. En el apartado anterior 2.1, hemos estudiado las curvas elípticas, que a partir de los puntos de un grupo G que forman dicha curva, obtenemos un conjunto finito donde se cree que es más difícil de resolver el Problema del Logaritmo Discreto. Este caso, se conoce como Problema del Logaritmo Discreto en Curvas Elípticas (PLDCE). Para un análisis más profundo de este tema, se recomienda la lectura del capítulo *The elliptic curve discrete logarithm problem* del libro *Guide to Elliptic Curve Cryptography* [HMV04].

Ejemplo 2.2.1. Sea $\mathbb{F}_{2131}^*$ un grupo multiplicativo del conjunto de enteros módulo 2131, se tiene que $\mathbb{F}_{2131}^* \equiv \langle 37 \rangle$. Como $1217 \equiv 37^5$ (mód 2131), el logaritmo discreto de 1217 en base 37 es 5.

Hemos podido comprobar que la exponenciación es una función trampa, ya que aumenta la complejidad. Por este motivo, algunos criptosistemas basan su seguridad en el PLD. Durante los siguientes apartados se estudian varios algoritmos cuya seguridad reside en la extrema dificultad de calcular logaritmos discretos en un cuerpo finito. Como la idea fundamental del trabajo es estudiar la criptografía del curvas elípticas, todos los algoritmos trabajarán con curvas elípticas establecidas por los interlocutores sobre un cuerpo finito. La información descrita ha sido obtenida en el capítulo 6 (*Elliptic Curve Cryptography*) del libro *Elliptic curves: number theory and cryptography* [Was08]

2.2.3 Algoritmo de Diffie-Hellman

El protocolo criptográfico Diffie-Hellman (figura 2.8) consiste en utilizar una clave compartida entre dos interlocutores sin permitir que alguien que acceda a las comunicaciones, pueda llegar a obtenerla. Para ello, cada uno tiene una clave pública propia y otra secreta.

Los interlocutores utilizan una fórmula matemática que incluye exponenciación y necesita dos números públicos y un número secreto. Los resultados tras la operación, se intercambian de forma pública. Revertir esta función es tan difícil como calcular un logaritmo discreto. Como hemos visto en la introducción, esta operación es muchísimo más costosa que la exponenciación usada para transformar los números inicialmente.

Por último, cada interlocutor utiliza una fórmula matemática que combina los resultados de la primera operación con su número secreto y ambos obtienen resultado final, que será la clave compartida. La descripción paso a paso del protocolo criptográfico de Diffie-Hellman entre los interlocutores A y B sería:

1. A y B acuerdan trabajar con la curva elíptica E sobre un campo finito $\mathbb{F}_q$ y el punto $P \in E(\mathbb{F}_q)$, cuyo subgrupo generado tiene un orden grande. Normalmente la curva elíptica y el punto son elegidos para que el orden sea un número primo de gran tamaño.

2. A elige un número entero secreto a, calcula $P_a = aP$ y envía P_a a B.

3. B elige un número entero secreto b, calcula $P_b = bP$ y envía P_b a A.

4. A computa $aP_b = abP$.

5. B computa $bP_a = baP$.

6. A y B utilizan un método acordado para extraer una clave del resultado de abP. Por ejemplo, podrían usar los últimos bits de la coordenada x de abP o utilizar una función hash en la coordenada.

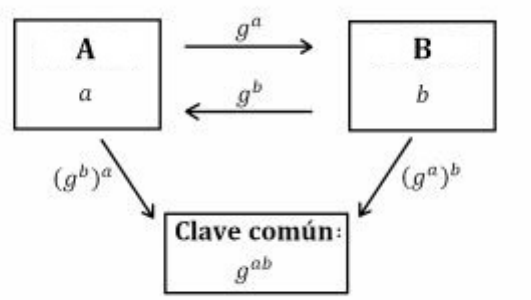

FIGURA 2.8: Intercambio de claves Diffie-Hellman

Por lo tanto, la única información pública es la curva E, el campo finito $\mathbb{F}_q$ y los puntos P, P_a y P_b. La única manera de calcular abP sería mediante el Problema del Logaritmo Discreto.

2.2.4 Algoritmo de Massey-Omura

Este criptosistema (figura 2.9) utiliza la conmutatividad de ciertas funciones para conseguir que dos personas intercambien un mensaje de forma segura, sin llegar a compartir ninguna clave públicamente. El procedimiento entre A y B sería el siguiente:

1. A y B eligen una curva elíptica E sobre un campo finito $\mathbb{F}_q$ de manera que el Problema del Logaritmo Discreto es difícil de resolver en $E(\mathbb{F}_q)$.

2. A representa su mensaje como un punto $M \in E(\mathbb{F}_q)$. Elige un número entero como su clave m_A que cumpla $MCD(m_A, E(\mathbb{F}_q)) = 1$. Por último, calcula $M_1 = m_A M$ y se lo envía a B.

3. B elige un número entero como su clave m_B que cumpla $MCD(m_B, E(\mathbb{F}_q)) = 1$. Por último, calcula $M_2 = m_B M$ y se lo envía a A.

4. A calcula $m_A^{-1} \in \mathbb{Z}_{E(\mathbb{F}_q)}$. Calcula $M_3 = m_A^{-1} M_2$ y envía M_3 a B.

5. B calcula $m_B^{-1} \in \mathbb{Z}_{E(\mathbb{F}_q)}$. Calcula $M_4 = m_B^{-1} M_3$ y obtiene el mensaje inicial, porque M_4 equivale al M inicial del paso 2.

La fórmula completa del proceso es:

$$M4 = m_B^{-1} m_A^{-1} m_B m_A M = M$$

Donde m_A^{-1} y m_B^{-1} representan al inverso de m_A y m_B mod N, respectivamente.

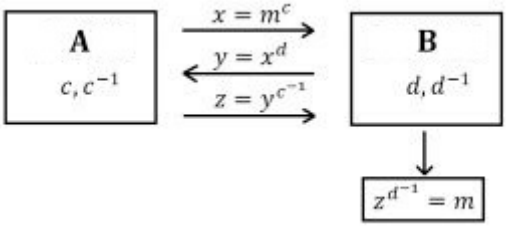

FIGURA 2.9: Criptosistema de Massey-Omura

2.2.5 Algoritmo de ElGamal para Clave Pública

El procedimiento de cifrado/descifrado ElGamal (figura 2.10) es un algoritmo de criptografía asimétrica basado en la idea de Diffie-Hellman y que funciona de una forma parecida, pero con algunas mejoras.

El algoritmo consiste en que el receptor elige una curva elíptica E sobre un campo finito F_q de manera que el problema del logaritmo discreto sea difícil de resolver para $E(F_q)$. También elige un punto P en la curva elíptica E, cuyo orden es un número primo grande. Además, elige un número entero secreto s y calcula $B = sP$. La curva elíptica E, el campo finito $\mathbb{F}_q$ y los puntos P y B son la clave pública para el emisor y cualquiera puede conocer estos datos. La clave privada es el entero s que solamente conoce el receptor del mensaje.

Para enviar un mensaje al receptor anterior, hay que seguir los siguientes pasos:

1. Descargar su clave pública (curva elíptica E, el campo finito $\mathbb{F}_q$ y los puntos P y B).
2. Expresar nuestro mensaje como un punto $M \in E(\mathbb{F}_q)$.
3. Elegir un entero aleatorio secreto k y calcular $M_1 = kP$.
4. Calcular $M_2 = M + kB$.
5. Enviar M_1, M_2 al receptor.

El receptor obtendrá el mensaje inicial, después de descifrar mediante la siguiente operación:

$$M = M2 - sM1$$

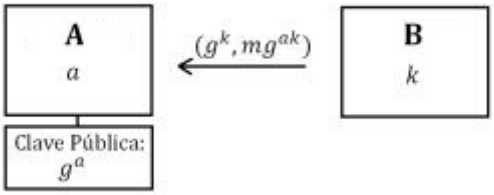

FIGURA 2.10: Algoritmo de ElGamal para Clave Pública

2.2.6 Algoritmo de ElGamal para Firma Digital

La firma clásica se puede integrar de modo muy sencillo a cualquier documento gracias a la digitalización, pero tiene la desventaja de que cualquiera puede utilizar esa firma. Por lo tanto, es necesario tomar medidas para asociar la firma al documento de tal manera que no se pueda volver a utilizar. Además, debe ser posible verificar que la firma es válida y demostrar que pertenece a esa persona. La solución a estos problemas, se resuelve mediante el algoritmo de ElGamal para la firma digital (figura 2.11). Inicialmente el algoritmo se desarrolló para el grupo multiplicativo de un campo finito y se aplica a cualquier grupo finito. En este caso, lo vamos a presentar para curvas elípticas.

Para firmar un documento con este algoritmo, primero se necesita establecer una clave pública y elegir la curva elíptica E sobre un campo finito $\mathbb{F}_q$ donde se va a trabajar, de manera que el Problema del Logaritmo Discreto sea difícil de resolver. Además hay que elegir un punto de $A \in E(\mathbb{F}_q)$, cuyo orden sea un número primo grande N. Por último, a partir de un número entero privado a, calcular $B = aA$ y elegir la función $f : E(\mathbb{F}_q) \to \mathbb{Z}$, necesaria para operar.

Por lo tanto, La información pública es $E, \mathbb{F}_q, f, A$ **y** B, a es la clave privada y N no necesita hacerse público, porque ni afecta a la seguridad, ni sirve para firmar.

Los pasos para firmar un documento son los siguientes:

1. Representar el documento como un entero m. En caso de que m sea más grande orden del punto A, habría que utilizar una curva más grande o usar una función hash.
2. Elegir un número entero aleatorio k con $\text{mcd}(k, N) = 1$ y calcula $R = kA$.
3. Calcular $s \equiv k - 1(m - af(R))(modN)$.

Después de estos pasos, el mensaje firmado estaría formado por m, R, s donde m, s son números enteros y $R \in E(F_q)$. Para poder verificar que el documento está firmado, se tienen que realizar los siguientes pasos:

1. Descargar la información pública del firmante.
2. Calcular $V_1 = f(R)B + sR$ y $V_2 = mA$.
3. Si $V_1 = V_2$, la firma es válida y el documento está firmado por la persona adecuada.

FIGURA 2.11: Algoritmo de ElGamal para Firma DIgital

2.2.7 Algoritmo de Firma Digital

La firma digital es básicamente un conjunto de datos asociados a un mensaje que permiten asegurar la identidad del firmante y la integridad del mensaje, igual que hemos podido ver en el apartado anterior. La firma digital debe depender tanto del mensaje como del autor. Si esto no fuese así, el receptor podría modificar el mensaje y mantener la firma, produciendo así un fraude. Los criptosistemas de clave pública pueden ser fácilmente utilizados para generar firmas digitales. El Algoritmo de Firma Digital (DSA por sus siglas en inglés) es una variante del esquema de firmas de ElGamal, con algunas modificaciones.

Para firmar un documento m, que es un número entero (normalmente se firma el hash del documento), se elige una curva elíptica sobre un campo finito $\mathbb{F}_q$ tal que $E(\mathbb{F}_q) = fr$, donde r es un número primo grande y f es un número entero pequeño, generalmente 1, 2 o 4 (f debería ser pequeño para mantener el algoritmo eficiente). Además, se elige un punto G en $E(\mathbb{F}q)$ de orden r. Finalmente, se escoge un entero secreto a y hay que calcular $Q = aG$.

La información pública es $\mathbb{F}_q, E, r, G, Q$ y para firmar el mensaje m, se siguen los siguientes pasos:

1. Elegir un número entero aleatorio k con $1 \leq k < r$ y calcula $R = kG = (x, y)$.
2. Calcular $s = k^{-1}(m + ax)(\text{mod } r)$.

El documento firmado es (m, R, s). Para verificar la firma, se hace lo siguiente:

1. Calcular $u_1 = s^{-1}m(modr)$ y $u_2 = s^{-1}x(modr)$.
2. Calcular $V = u_1G + u_2Q$.
3. En caso de que $V = R$, la firma es válida y el documento está firmado por la persona adecuada.

Capítulo 3

Redes Neuronales

3.1 Introducción

La computación cada día es más potente y el ser humano continua trabajando en el desafío de automatizar tareas relativamente simples para las personas, como reconocer una letra o diferenciar un animal. Gracias a nuestro aprendizaje, somos capaces de lograr un mejor desempeño en tareas más complejas, como maximizar recursos o alcanzar un objetivo común. Algunas de estas actividades pueden ser agrupadas en patrones similares o ser resueltas mediante soluciones informáticas.

El desarrollo de las redes neuronales artificiales comenzó hace aproximadamente 50 años, motivado por el deseo de intentar comprender el cerebro e imitar algunas de sus fortalezas. Las ideas iniciales sobre el aprendizaje en máquinas estuvieron en un segundo plano debido al gran progreso de la computación digital. Además, la ideas generadas sobre los primeros modelos de redes neuronales fueron negativas y llevaron a numerosas dudas. Un renovado reciente interés en el campo ha surgido gracias a nuevas arquitecturas y técnicas de entrenamiento, ordenadores de alto rendimiento y hardware mucho más específico para realizar simulaciones de redes neuronales en condiciones mucho más óptimas. A su vez, el progreso en la computación tradicional ha hecho que el estudio de este campo sea más sencillo y se han encontrado nuevas limitaciones, que han motivado a tomar otras direcciones en la investigación.

Las redes neuronales son interesantes en diferentes áreas por muchas razones. En electrónica se usan en aplicaciones de procesamiento de señales y teoría de control, en computación para la mejora de rendimiento de hardware, o en robótica, diferentes áreas como inteligencia artificial o el reconocimiento de patrones, las utilizan como herramienta para modelar problemas.

Una red neuronal artificial es un sistema de procesamiento de información que comparte características con las redes neuronales biológicas. Concretamente, son modelos matemáticos que cumplen que:

- La información se procesa en elementos simples llamados neuronas.
- Las neuronas se encuentran conectadas entre sí y se envían señales de datos.

- Cada conexión tiene un peso o valor que multiplica la señal que llega a la neurona.
- Cada neurona utiliza una función de activación para determinar la señal de salida.

La red neuronal está caracterizada por su patrón de conexiones entre las neuronas, que se conoce como arquitectura. El método que se usa para determinar los pesos de las conexiones es llamado entrenamiento o aprendizaje.

Se recomienda la lectura de *Fundamentals of neural networks: architectures, algorithms, and applications* [Fau08] y *Pattern Recognition and Machine Learning* [Bis16] para obtener información más profunda y completa sobre de las redes neuronales.

3.1.1 Aprendizaje automático

El aprendizaje automático forma parte de una rama de la inteligencia artificial y es un proceso mediante el cual se usan modelos matemáticos de datos con el objetivo de que las computadoras aprendan sin instrucciones directas. El aprendizaje automático trabaja con algoritmos que tratan de identificar patrones en conjuntos de datos, y a partir de los patrones, se crean modelos de datos para hacer predicciones. Los resultados del aprendizaje son más precisos con la experiencia y conjuntos de datos más amplios, es decir, las computadoras aprenden de manera similar a los humanos, ya que mejoran con la práctica.

El aprendizaje automático es una opción ideal en casos donde los datos o la tarea a realizar se cambian constantemente, ya que la codificación de una solución para estos problemas, sería prácticamente imposible de construir manualmente o tendría un alto coste en caso de ser posible. Es utilizado en una gran cantidad de aplicaciones, como por ejemplo en diagnósticos médicos, robótica o motores de búsqueda.

Dentro del aprendizaje automático encontramos dos tipos según si es el aprendizaje es supervisado o no. En la figura 3.1 podemos observar ambos conjuntos y otros subtipos dentro de cada una. A lo largo de este capítulo estudiaremos los más importantes.

3.1.2 Redes neuronales biológicas

La semejanza entra la estructura de las redes neuronales artificiales y biológicas es bastante clara. Un neurona biológica (figura 3.2) tiene tres tipos de componentes que son interesantes para entender la artificial (figura 3.4): dentritas, soma y axón.

- Las **dentritas** reciben señales desde otras neuronas. Las señales son impulsos eléctricos, transmitidos a través de una brecha sináptica, es decir, mediante procesos químicos. El transmisor químico modifica la señal entrante de manera similar a los pesos en la neurona artificial.

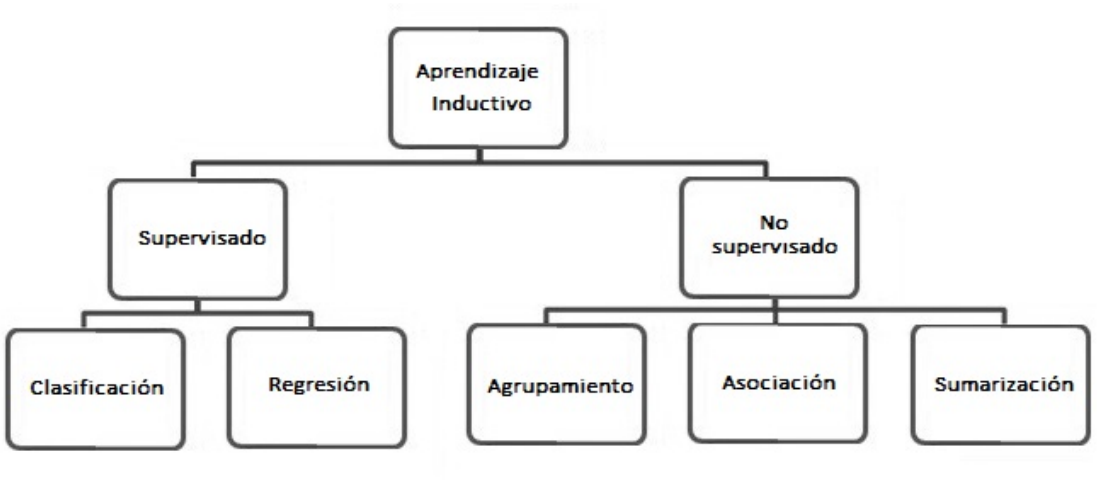

FIGURA 3.1: Clasificación aprendizaje automático

- El **soma** es el cuerpo de la neurona y suma los impulsos recibidos. Cuando recibe la cantidad suficiente, se produce el incendio celular que transmite la señal a otras células. Esta forma de transmitir puede ser tratada como binaria, ya que o la neurona manda la señal o no.
- El **axón** es la conexión entre las neuronas, por donde cada una envía sus impulsos.

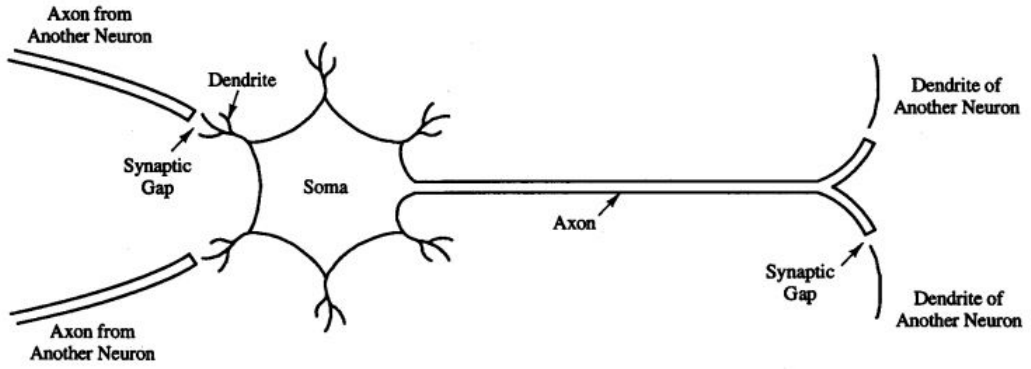

FIGURA 3.2: Neurona biológica

3.2 Funcionamiento

Una red neuronal artificial puede tener entre un par de neuronas hasta millones que se encuentran separadas en diferentes capas. En la figura 3.3 se representa una red neuronal de varias capas.

Cada capa está conectada con la anterior y con la siguiente. Las neuronas también son conocidas como unidades y sus conexiones se representan con un número llamado peso, el cuál puede ser positivo o negativo, y su valor está relacionado con la influencia que tenga la neurona previa. Cuanto mayor es el peso, más influencia tiene una neurona sobre otra. Como hemos podido ver antes, esto es lo que hacen las neuronas biológicas, que disparan impulsos a través de sus conexiones.

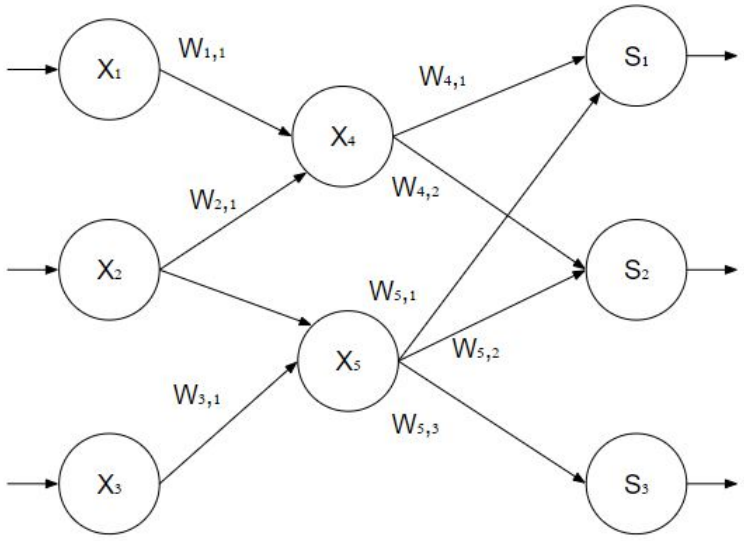

FIGURA 3.3: Estructura de una red neuronal

La figura 3.4 es una neurona con tres entradas (x_1, x_2, x_3), los pesos de cada entrada (w_{1j}, w_{2j}, w_{3j}), el bias (b_j), el cálculo final de entrada con los valores anteriores (z_j), la función de activación $(f(z_j))$ y el resultado de salida de la nuerona (y_j).

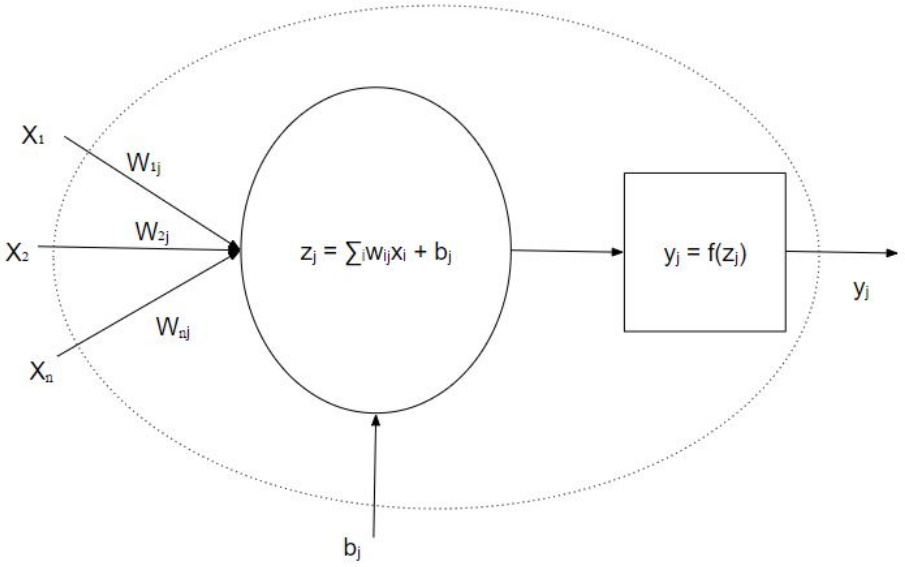

FIGURA 3.4: Neurona artificial y parámetros

Así considerando su estructura podemos hablar de redes monocapa, compuestas por una única capa de neuronas, o redes multicapa (figura 3.3), donde las neuronas se organizan en diferentes capas. Normalmente, las neuronas de una misma capa, se comportan de la misma manera.

Varios factores clave a la hora de determinar el comportamiento de una neurona, son la función de activación y el patrón de ponderación de conexiones con el resto de neuronas, pero esto lo trataremos

un poco más adelante. Por lo general, las neuronas de la misma capa, comparten ambas cosas.

3.2.1 Capas

En general las neuronas se suelen agrupar en unidades estructurales que denominaremos capas. El conjunto de una o más capas constituye la red neuronal. Se distinguen tres tipos de capas: de entrada, salida y ocultas (figura 3.5).

- La **capa de entrada** está compuesta por neuronas que reciben datos o señales procedentes del entorno.
- La **capa oculta** no tiene una conexión directa con el entorno, es decir, no se conecta directamente ni a órganos sensores ni a efectores. Este tipo de capa oculta proporciona grados de libertad a la red neuronal gracias a los cuales es capaz de representar determinadas características del entorno que trata de modelar.
- La **capa de salida** se compone de neuronas que proporcionan la respuesta de la red neuronal.

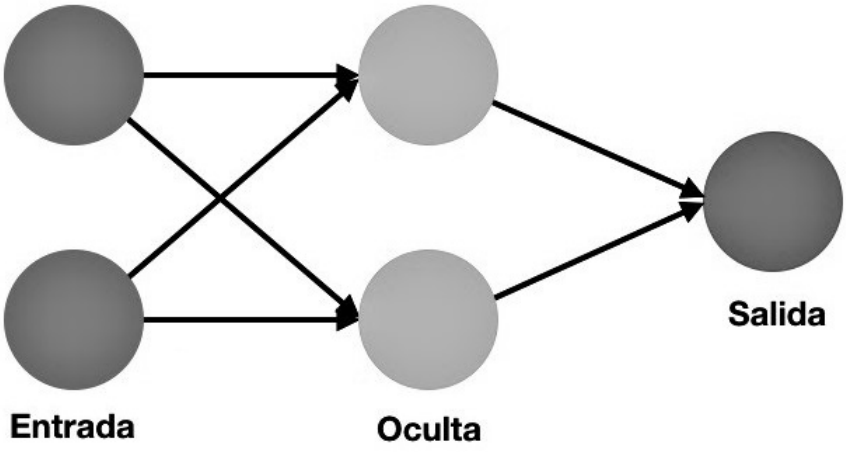

FIGURA 3.5: Capas de una red neuronal

Los patrones de información alimentan a la red a través de las entradas, las cuales disparan valores en la capa oculta y estas llegan eventualmente a las neuronas de salida. Esto es denominado red de alimentación hacia adelante. No necesariamente una neuorana se activa siempre. Esta activación depende de los valores de las entradas y de si alcanza un valor umbral. Esto es algo similar a las neuronas humanas y es la señal que se conoce como estímulo nervioso.

La disposición de las neuronas en capas y los patrones de conexión se denomina arquitectura de red.

3.2.2 Arquitectura de red

La **red monocapa** tiene una única capa de pesos de conexión. Las neuronas reciben señales de entrada y generan la respuesta de la red. Las unidades de entradas están directamente conectadas a las unidades de salida.

La **red de Hopfield** es una red monocapa. Aunque también se puede mostrar como una red bicapa, donde la primera capa es de sensores y la segunda es la encarga de realizar el procesamiento. En la versión bicapa, la manera de conectar las neuronas es mediante la unión de la primeras con las de la segunda linealmente, es decircada neurona con su respectiva.

La **red multicapa** es aquella con una o más capas ocultas entre las unidades de entrada y salida. Normalmente hay una capa de pesos entre dos capas adyacentes (entrada, oculta y salida). Las redes multicapa pueden resolver problemas más complejos que las monocapa, pero el entrenamiento es más complejo y conlleva una cantidad superior de tiempo. En algunas ocasiones los entrenamientos más grandes pueden llevar al éxito que una red de una capa no puede alcanzar porque su entrenamiento sea incompleto.

La **red competitiva** se diferencia de otras redes neuronales porque las neuronas compiten para representar los patrones, es decir, cuando ejecutamos un patrón en una red competitiva, se activa únicamente la neurona que representa mejor dicho patrón. Eso la diferencia del resto de redes, donde las neuronas colaboran entre ellas para la representación de los patrones. Las redes competitivas son usualmente bicapas. La función de la primera capa es hacer de sensor y por ella entran los patrones a la red. Por lo tanto, debe tener el mismo tamaño que la longitud del patrón. La segunda capa tiene tantas neuronas como categorías deseemos. Sin embargo, algunas redes competitivas, crean neuronas dinámicamente, con el objetivo de ajustar el número de categorías automáticamente.

3.2.3 Forward Step. Función de activación

El funcionamiento básico de un neurona equivale a sumar la señal de entrada ponderada y aplicar la **función de activación** que define la salida. Esta salida determina si la neurona se va a activar o no. Normalmente, se trabaja con el mismo tipo de función para todas las neuronas de una capa, aunque no es obligatorio. En redes multicapa, la función suele ser no lineal. Dentro de las diferentes funciones con las que podemos trabajar, destacamos las siguientes:

La función **de paso binario** (figura 3.6) es utilizada por redes monocapa para convertir una entrada de valor continuo, en una señal de salida binaria (0 ó 1) o bipolar (1 ó -1).

$$f(x) = max(0, x) = \begin{cases} 0 & si \quad x < 0 \\ 1 & si \quad x \geq 0 \end{cases}$$

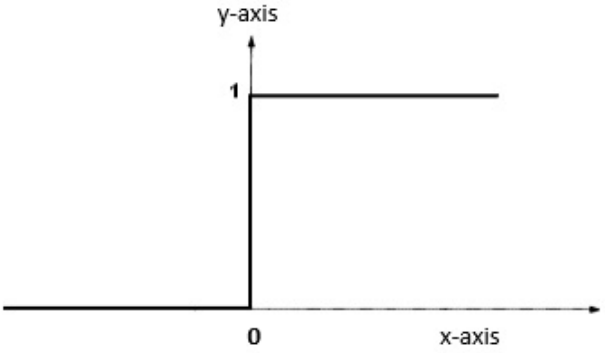

FIGURA 3.6: Función de paso binario

La función **lineal** (figura 3.7) es la más básica de esta lista y normalmente se utiliza en el nodo de salida, cuando el objetivo es obtener un valor real. Su mayor inconveniente es que no puede manejar la complejidad de los datos.

$$f(x) = x$$

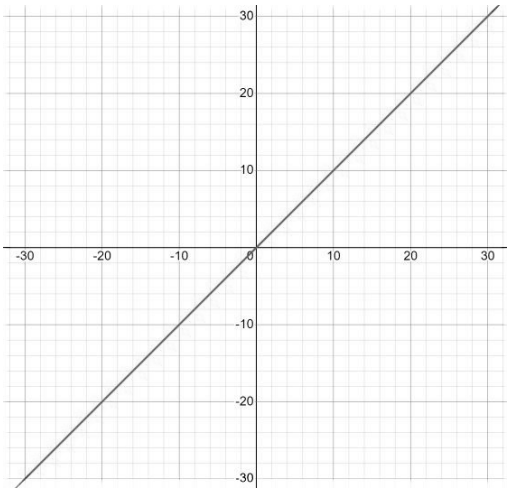

FIGURA 3.7: Función lineal

La función **sigmoide o logística** (figura 3.8) convierte los valores introducidos a una escala (0,1), de modo que los valores más altos tienden a 1 y los más bajos a 0. Es útil para cálculos probabilísticos, aunque su optimización es más compleja porque no está centrada en cero.

$$f(x) = \frac{1}{1 + e^{-x}}$$

La función **tangente hiperbólica** (figura 3.9) transforma los valores introducidos a una escala (-1,1), de modo que los valores altos tienden a 1 y los más bajos a -1. A diferencia que la función sigmoide, las salidas pueden ser valores negativas, por lo que se entrena más fácilmente, ya que se puede centrar a cero. Computacionalmente es pesada y su convergencia es mucho más lenta.

$$f(x) = \frac{2}{1 + e^{-2x}} - 1$$

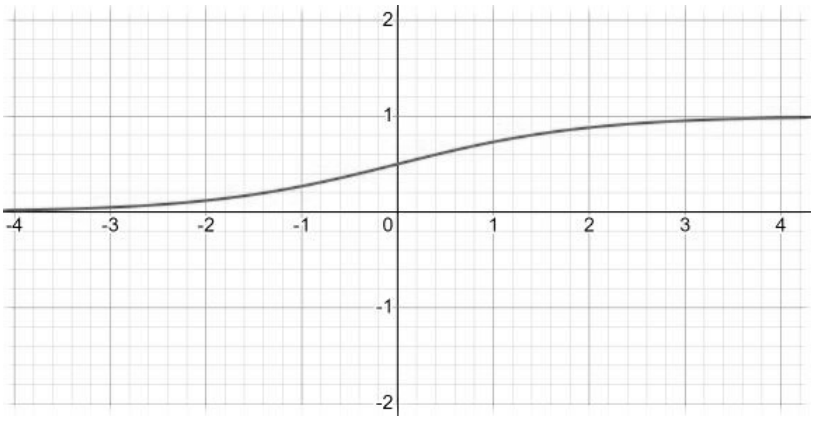

FIGURA 3.8: Función sigmoide

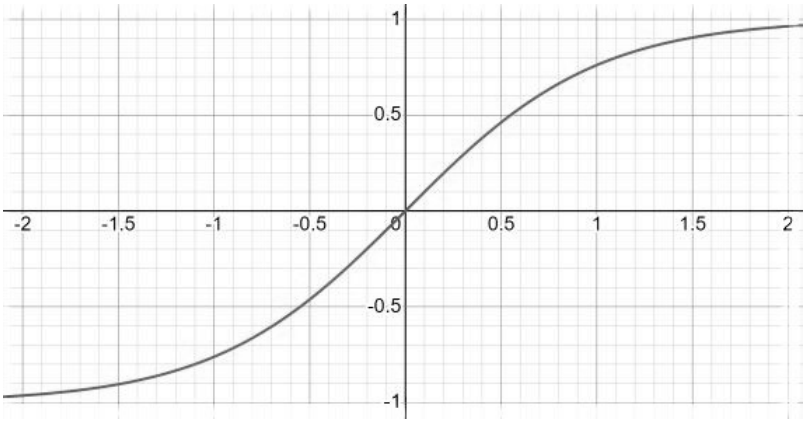

FIGURA 3.9: Función tangente hiperbólica

La función **ReLU**(*Rectified Linear Unit*) (figura 3.10) anula los valores negativos y deja los positivos tal y como entran. Es una función extremadamente rápida de calcular y tiene gran efectividad frente a varios dominios de aplicación. Además, ayuda a minorizar el problema de la saturación de gradiente (véase 3.2.5.1) cuando los valores de la combinación lineal de entrada son mayores que cero. Esta función logra que la propagación hacia atrás sea efectiva incluso en redes profundas. Es importante tener en cuenta que algunas neuronas pueden llegar a colapsar durante el entrenamiento, debido a su derivada para ciertos valores.

$$f(x) = max(0, x) = \begin{cases} 0 & si \quad x < 0 \\ x & si \quad x \geq 0 \end{cases}$$

La función **Leaky ReLU** (figura 3.11) multiplica valores de entrada negativos por un coeficiente rectificativo y deja los positivos tal y como entran. Soluciona el problema de colapso de las neuronas que hemos visto en la función ReLU.

$$f(x) = \begin{cases} 0 & si \quad x < 0 \\ a \cdot x & si \quad x \geq 0 \end{cases}$$

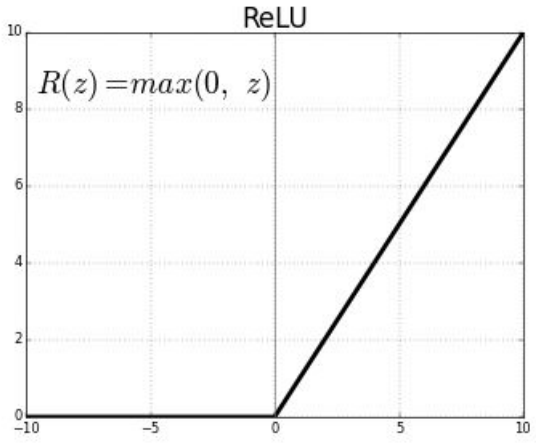

FIGURA 3.10: Función ReLU

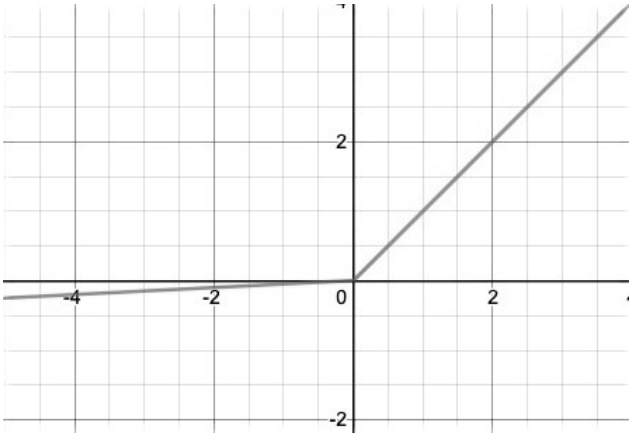

FIGURA 3.11: Función Leaky ReLU

La función **Softmax** (figura 3.12) no es realmente una función de activación, ya que no recibe un único elemento como entrada. Se encarga de calcular la exponencial de cada elemento y la divide entre la suma de las exponenciales del vector. El resultado es un vector con las probabilidades de cada elemento que suman en total 1. La función Softmax debe usarse en aquellos casos donde las clases son excluyentes, en otro caso, se recomiendo usar la función sigmoide.

$$f(x) = \frac{e^{x_j}}{\sum_{k=1}^{K} e^{x_k}}$$

El crecimiento de las capas utilizadas en las redes neuronales y sus capacidades provoca el uso y la creación de nuevas arquitecturas, modelos de neuronas y formas de optimización. En los próximos apartados, exploraremos algunas de las estrategias para enfrentar estos desafíos, así como nuevas arquitecturas y su aplicación a problemas más complejos.

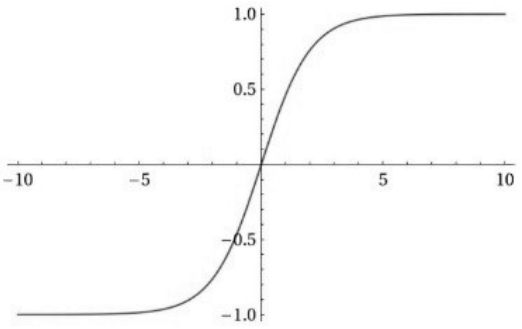

FIGURA 3.12: Función Softmax

3.2.4 Entrenamiento

Durante muchos años, el perceptrón fue el algoritmo utilizado en todos los estudios de redes neuronales, pero solamente era capaz de resolver problemas lineales y era incapaz de resolver problemas más complejos. Con la publicación del libro *Perceptrons* [MP69] donde se mostraban sus limitaciones, tuvo lugar un parón en la financiación y el estudio de la inteligencia artificial. Más de 15 años después, se publicó el artículo *Learning representations by back-propagating errors* [DERW86] sobre un algoritmo que autoajustaba los parámetros de la red neuronal, conocido como backpropagation.

Las redes neuronales aprenden a través de un elemento de retroalimentación, igual que el ser humano usa la retroalimentación todo el tiempo. Siempre que tratamos de aprender algo, lo hacemos tal y como nos han dicho que es la manera correcta. Esto quiere decir que ajustamos nuestra conducta a los conocimientos adquiridos. La retroalimentación en una red neuronal es conocida como **propagación hacia atrás** y su principal objetivo es el de comparar la salida de la red con el resultado esperado. Para entrenar una red de esta manera, se necesita un gran conjunto de ejemplos a partir de los cuáles la red pueda avanzar en el aprendizaje. Todo este entrenamiento tiene que trabajar sobre datos en binario, para que las neuronas hagan su trabajo y se aproximen lo máximo posible al resultado esperado.

Las redes neuronales se clasifican según la disposición de las neuronas en cada capa, los diferentes patrones para la conexión y los algoritmos o métodos que se utilizan para entrenar y lograr el aprendizaje. Es importante tener en cuenta que las redes neuronales no siempre se entrenan, como en el caso de las redes de pesos fijos, que no utilizan ningún tipo de entrenamiento. El método de ajustar los valores de los **pesos** es otra característica distintiva y se utiliza tanto en el aprendizaje supervisado y como en el no supervisado.

Los modelos de **aprendizaje supervisado** son aquellos en los que se especifica el conjunto de datos con la relación de entrada y su salida deseada, para llegar al objetivo del aprendizaje. Según el tipo de salida, hay una subcategoría que diferencia entre **modelos de clasificación** cuando la salida es un valor categórico (por ejemplo, un conjunto finito de clases), y **modelos de regresión** cuando la salida es un valor de un espacio continuo. Las redes neuronales de entrenamiento supervisado son las más populares, el hecho de conocer la salida beneficia al entrenamiento como si tuviera la supervisión de un maestro. En estos casos, el peso de una neurona en la etapa $(m+1)$ se calcula con la siguiente

fórmula:

$$w_{ij}^{m+1} = w_{ij}^{m} + \triangle w_{ij}^{m}$$

Por otro lado, los modelos de **aprendizaje no supervisado** no cuentan con datos que definan qué información es satisfactoria o no, su objetivo no es ajustar los pares de entrada y salida, sino trabajar en el aumento del conocimiento estructurado de los datos disponibles, es decir, clasificar los datos en grupos en función de sus atributos. Algunos ejemplos son la Regla de Aprendizaje de Hebb y el aprendizaje competitivo.

Los modelos de **peso fijo** se utilizan para tratar de resolver problemas de optimización con restricciones. Los pesos son establecidos para representar las restricciones y la cantidad a maximizar o minimizar. Normalmente, una solución casi óptima es satisfactoria. Ejemplos de este modelos son la máquina de Boltzmann (sin aprendizaje) y la red de Hopfield.

3.2.5 Función de pérdida

El propósito de entrenar un modelo es predecir un resultado lo más cercano posible a la respuesta correcta, es decir, minimizar al máximo el error. Este error se mide a partir de la función de pérdida y para obtener sus mínimos, se necesita encontrar los valores óptimos del gradiente.

Una función de pérdida $J(x)$ mide el nivel de insatisfacción de las predicciones de nuestro modelo respecto a la respuesta correcta. Existen varias funciones de pérdida como el error cuadrático medio o la entropía cruzada. La selección de uno de ellos depende de varios factores como el algoritmo seleccionado o el nivel de confianza deseado, pero principalmente depende del objetivo del modelo.

Una **función de pérdida** es la que evalúa la desviación entre las predicciones realizadas por la red neuronal y los valores reales de las observaciones utilizadas durante el aprendizaje. La eficiencia de la red neuronal depende del resultado de esta función. El objetivo principal de una red neuronal es reducir al mínimo la desviación entre ambos valores y se consigue ajustando los pesos de las neuronas.

El **error de aprendizaje** se calcula obteniendo la diferencia entre el valor real que hay que predecir y el valor retornado por la neurona artificial. Este es el error que buscaremos minimizar durante los aprendizajes.

$$Error = Prediccion\,real - Prediccion\,realizada$$

El **error cuadrático medio** es una función de error global del aprendizaje. Es decir, que esta función nos permitirá conocer de manera global el porcentaje de error cometido por nuestra neurona.

Esto se conoce como optimización y es la base de los algoritmos usados en redes neuronales. Al igual que con la función de pérdida, existen varios métodos de optimización que impactan en el rendimiento de nuestro modelo y el tiempo que toma entrenarlo.

3.2.5.1 Descenso de gradiente

El descenso de gradiente (figura 3.13) es un algoritmo que se utiliza en las redes neuronales para modificar ciertos parámetros y reducir el error, es decir, mejorar el rendimiento y por lo tanto, lograr el aprendizaje. Para lograr el mayor aprendizaje, necesitamos conocer los mínimos locales de la función de coste, es decir, aquellos puntos donde el coste sea menor. Nos apoyaremos en la derivada, que será igual a cero en esos puntos y nos ayuda a conocer la pendiente de la función en cada punto. El descenso de gradiente es un algoritmo que estima numéricamente dónde una función genera sus valores más bajos. Por lo tanto, evalúa la pendiente en cada punto en busca de los mínimos (figura 3.14).

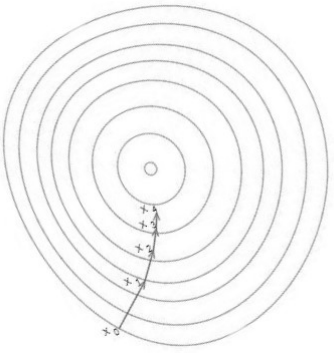

FIGURA 3.13: Descenso del gradiente

El gradiente es la generalización vectorial de la derivada, es un vector de tantas dimensiones como la función y cada dimensión contiene la derivada parcial en dicha dimensión:

$$\nabla f = \left(\frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial x_2}, ..., \frac{\partial f}{\partial x_n} \right)$$

La fórmula para recalcular los parámetros tiene el siguiente formato:

$$\widehat{\theta} = \theta - \alpha \nabla J(\theta)$$

Donde θ es el valor anterior del parámetro $\widehat{\theta}$ el nuevo, α la tasa de aprendizaje y $\nabla J(\theta)$ el gradiente de la función de coste para el parámetro θ.

Es importante no utilizar una tasa de aprendizaje grande, porque determina el tamaño del paso que hace el algoritmo por la función y puede impedir que se encuentren los puntos mínimos. Una estrategia sería si la función crece mucho, descender la tasa de aprendizaje mucho, y si crece poco, descender la tasa un poco. En otras palabras, dar el paso en forma proporcional al gradiente. El ratio de aprendizaje también se conoce como parámetro de *Tunning*.

A pesar de que avancemos en dirección contraria al gradiente, no se puede garantizar que lleguemos al punto más mínimo. En algunas ocasiones llegaremos a mínimos locales. Estos son regiones que parecen mínimos por la configuración alrededor a ellos, pero no son el mínimo global.

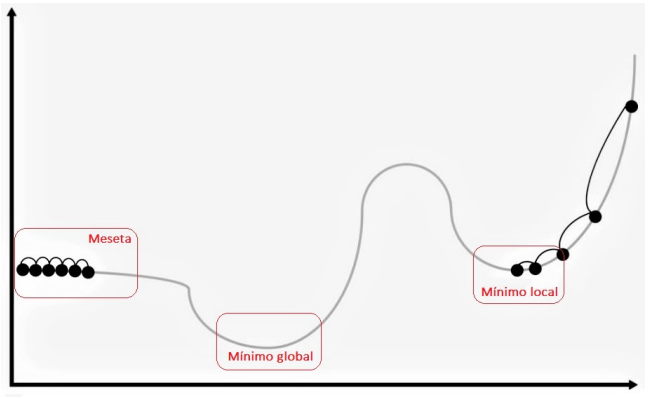

FIGURA 3.14: Mínimo global y local

Cuando una red neuronal es profunda, el proceso de backpropagation puede provocar que gradientes menores a uno se hagan cada vez más pequeños al pasar por múltiples capas. Si el gradiente se vuelve demasiado pequeño, puede redondearse a cero (colapso numérico). Este suceso se conoce como **desvanecimiento de gradiente**. De manera similar, cuando tenemos muchas capas y los gradientes son mayores que uno, puede comenzar a crecer rápidamente ocasionando que el cálculo tenga un **overflow**, conocido como explosión de gradiente. Estos problemas se agravan según la forma de la derivada de la función de activación de cada una de las neuronas en las capas ocultas (saturación de gradiente). Por estos motivos, es fundamental la selección de la función de activación para evitar problemas con el gradiente.

El concepto de parámetros, función de pérdida y optimización son los principales componentes en muchos algoritmos usados en el aprendizaje automático.

3.2.5.2 Parámetros de la red

Para la optimización de la red neuronal, se emplean diferentes métodos de ajuste de parámetros de la red (pesos de las conexiones y sesgo de las neuronas), a partir de unos valores o bien aleatorios, o bien predefinido (inicialización de la red).

Al construir un sistema neuronal, se parte de un cierto modelo de neurona y de una determinada arquitectura de red, estableciéndose los pesos iniciales como nulos o aleatorios. En la optimización del entrenamiento, se ajustan los **pesos**.

El **sesgo o bias** permite indicar cuándo debe reaccionar la neurona, es decir, es un valor que controla qué tan predispuesta está la neurona a devolver un 1 o un 0, independientemente de los pesos. Un sesgo alto hace que la neurona necesite una entrada más alta para generar 1 de salida, justo al contrario sucede con un sesgo bajo.

Ajustar el sesgo nos ayudará a mejorar el proceso de ajuste de datos y por consecuencia, modelos más precisos. Adicionalmente, evitará el error de sobreajuste o la falta de ajuste.

3.2.5.3 Optimización a partir del dataset

Entrenar al modelo con datos sesgados hacia una determinada dirección, puede empeorar el desempeño, condenando los resultados obtenidos. Es la diferencia entre la predicción esperada de nuestro modelo y los valores verdaderos. Aunque al final nuestro objetivo es siempre construir modelos que puedan predecir datos muy cercanos a los valores verdaderos, no siempre es tan fácil porque algunos algoritmos son demasiado rígidos para aprender señales complejas del conjunto de datos. Es necesario tener esto en cuenta a la hora de preparar el dataset de entrenamiento.

3.2.6 Pruebas

La información fluye a través de una red neuronal de dos maneras. Cuando la red se entrena durante el proceso de aprendizaje, o bien, tras el entrenamiento, cuando se opera con ella para obtener los resultados. Es decir, una vez entrenado y optimizado el modelo, es el momento de probar la red neuronal y comprobar los resultados que genera.

3.3 Construir una red neuronal com Keras

Keras es una biblioteca de código abierto escrita en Python, que se basa principalmente en el trabajo del desarrollador de Google François Chollet. La primera versión se lanzó en marzo de 2015 con el objetivo de acelerar la creación de redes neuronales. Keras no funciona como un framework independiente, sino como una interfaz que permite acceder a varios frameworks de aprendizaje automático y desarrollarlos de manera mucho mas sencilla. Entre los frameworks compatibles con Keras, se incluyen TensorFlow[1], Theano[2] o Microsoft Cognitive Toolkit[3]. Para más información sobre Keras, véase [Cho]. También puede acceder a la documentación de Python en [Fou].

[1] https://www.tensorflow.org/
[2] https://theano-pymc.readthedocs.io/en/latest/
[3] https://docs.microsoft.com/en-us/cognitive-toolkit/

Capítulo 4

Implementación práctica de una red neuronal

4.1 Introducción

El objetivo inicial de este proyecto es el de construir una red neuronal para ponerla a prueba y comprobar si es capaz de romper un algoritmo criptográfico. A lo largo de esta sección vamos a tratar de implementar dos redes neuronales con un diseño y una meta diferente. La red inicial tiene el objetivo de romper el algoritmo de cifrado César y el destino de la segunda es el de probar la inteligencia artificial contra la criptografía de curvas elípticas que vimos en el capítulo 2.

La necesidad de probar el funcionamiento de las redes neuronales contra dos algoritmos diferentes de cifrado, se debe a la complejidad que presenta la criptografía de curvas elípticas. Este nivel de dificultad debería impedir a la red neuronal aprender hasta encontrar un patrón que rompa este problema matemático. Ese es el motivo principal por el que en primer lugar trabajaremos con el cifrado César. Gracias a su sencillez, la red neuronal no debería encontrar dificultad para encontrar el patrón y aprender, y por lo tanto, partiremos de una red neuronal inicial que funcione, y podremos comparar los resultados de ambos casos.

Antes de empezar en profundidad con los casos prácticos, es importante comentar que el propósito de este trabajo es el estudio de la aplicación de redes neuronales a la criptografía, y no el desarrollo desde cero de una red neuronal. Por esta razón, el desarrollo de código se va a apoyar en una biblioteca de código abierto para redes neuronales, llamada Keras [Cho]. Esta biblioteca facilita la implementación de redes neuronales desde una capa muy simple y sencilla, sin tener que programar las partes más complejas y profundas del código.

Este desarrollo ha sido realizado en el entorno Colab de Google[1] y se puede encontrar en mi GitHub personal [Mun].

[1] https://colab.research.google.com

4.2 Red neuronal cifrado César

Esta sección trata sobre la implementación de la red neuronal para tratar de romper el cifrado César. Tras una breve explicación de este tipo de cifrado, mostraremos el análisis y diseño de la red y el código utilizado. La idea es aumentar la complejidad de la red en el avance de las pruebas, es decir, al comienzo la red es más básica y en las siguientes pruebas aumentaremos el tamaño de las palabras para ver el comportamiento y como responde la red en cada caso.

4.2.1 Descripción

El cifrado César o cifrado por desplazamiento es un tipo de cifrado por sustitución, en el que cada letra es reemplazada por la que se encuentra un número fijo de posiciones más adelante en el alfabeto.

La figura 4.1 muestra el ROT3, que significa remplazar la letra por la que se encuentra 3 posiciones delante.

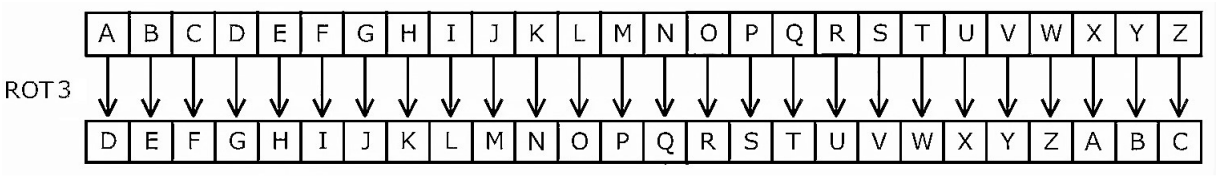

Figura 4.1: Tabla ROT3

El cifrado ROT3 o la rotación de 3 posiciones de la palabra *HOLA* sería *KROD* (figura 4.2).

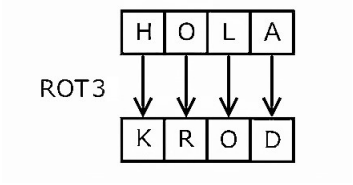

Figura 4.2: Ejemplo en ROT3

Las redes neuronales trabajan mejor directamente con valores numéricos y para obtener mejores resultados, pasaremos las letras a números (figura 4.3).

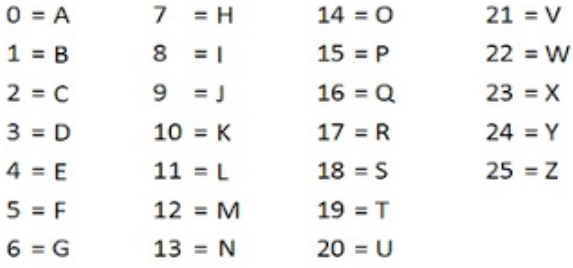

Figura 4.3: Abecedario numérico

Los números ahora se pasan a formato bits, que es cómo trabajan las redes neuronales, para ello usamos *One-hot* (figura 4.4), que consiste en una cadena de bits del tamaño del número de elementos (en este caso 26 por el abecedario) con un único bit a 1 para cada letra. Por ejemplo, para el número 1, la cadena solamente activa el último bit y para el número 25, el primero.

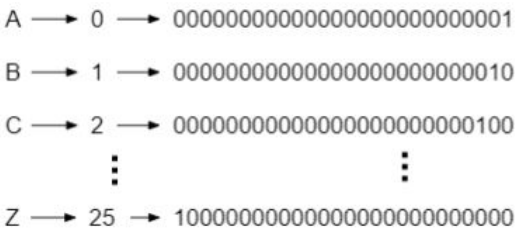

FIGURA 4.4: Codificación One-hot

El diseño de la red neuronal será como el de la figura 4.5. Una única letra en la entrada y salida de la red, dividida en 26 neuronas debido a la codificación *One-hot*.

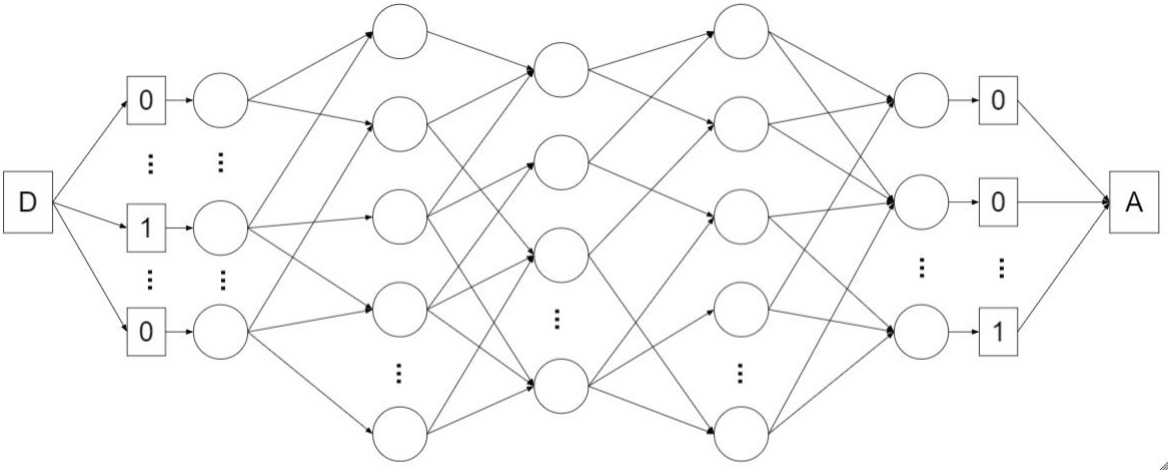

FIGURA 4.5: Diseño para la red neuronal de cifrado César

Progresivamente vamos a aumentar la complejidad del cifrado. Inicialmente, queremos ver como la red neuronal se comporta con cadenas de tres caracteres. Más adelante, probaremos cadenas de mayor tamaño y habrá que optimizar la red y ajustar los hiperparámetros en busca de un mejor resultado.

4.2.2 Implementación del código

Para facilitar la comprensión del trabajo y no cargar el contenido de este capítulo, se ha incluido el código implementado para el desarrollo del ejercicio en el apéndice A. El código está desarrollado en python y para su ejecución es necesario el uso de ficheros de texto plano con extensión ".py" (o ".pyw"). Además, para su ejecución es necesario el uso de un terminal, un editor IDLE o un entorno interactivo.

4.2.3 Resultados obtenidos

Una vez terminado el desarrollo de la red y comprobado su funcionamiento, nos hemos ayudado de la función GridSearchCV[2] para tratar de encontrar los mejores resultados para diferentes hiperparámetros de entrada. Un ejemplo de las pruebas realizadas es siguiente código.

```
batch_size = [50, 100, 200, 500]
epochs = [50, 60, 100, 150, 300]
optimizer = ['SGD', 'RMSprop', 'Adagrad', 'Adadelta', 'Adam', 'Adamax', 'Nadam']
loss = ['binary_crossentropy','categorical_crossentropy']
activation = ['relu','sigmoid','softmax']
unit1 = [50,  80, 120, 200]
unit2 = [50, 80, 120, 200]
```

Esta función evalúa y selecciona los valores que le proporcionamos en una lista para cada hiperparámetro mencionado anteriormente y nos muestra el rendimiento en cada caso, y de esta manera podemos realizar el modelo con la mejor selección. El modelo modelo finalmente usado es el siguiente:

```
def crearModelo(loss='poisson',activation='relu',u1=120,u2=80,optimizer='RMSprop'):
  # Definir modelo
  model = Sequential()
  # Capas
  model.add(Dense(nn[4], input_dim=(26*word_size)))
  model.add(Dense(u1, activation))
  model.add(Dense(u2, activation))
  model.add(Dense(u1, activation))
  model.add(Dense(nn[4], activation))
  # Compilar modelo
  model.compile(loss='binary_crossentropy', optimizer=optimizer,metrics=['accuracy'])
  return model
```

Durante las pruebas, se ha modificado también algunas características de la red neuronal, como el tamaño de palabra en cifrado César, para tratar de aumentar la complejidad, o el uso de ROT3 (visto en la figura 4.1). Las pruebas se han separado en función de varios hiperparámetros, como la longitud de la cadena de caracteres (3, 5 ó 10 letras) o la cantidad de combinaciones del dataset de entrenamiento (5.000, 10.000 ó 50.000).

En primer lugar, la figura 4.6 muestra los resultados de la primera prueba con un entrenamiento de 5.000 combinaciones de entrada. La que mejor resultados presenta es la red neuronal con 3 caracteres de entrada/salida. Como es de esperar, al ser más pequeña, las combinaciones posibles son muchas menos que la de 10 letras.

En segundo lugar, la figura 4.7 contiene los resultados de las prueba con tras el entrenamiento con 10.000 combinaciones de entrada. La red de tamaño 3 sigue siendo la que mejor resultados presenta y

[2] https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html

	3 letras	5 letras	10 letras
% COMBINACIONES ENTRADA	28,45%	0,04%	0,00%
PALABRAS CORRECTAS	64,10%	15,20%	0,20%
LETRAS CORRECTAS	86,50%	76,97%	42,92%

FIGURA 4.6: Pruebas con 5.000 combinaciones de entrada

mejora los resultados de la prueba anterior, pero en los otros 2 casos, se han empeorado los porcentajes de acierto, a pesar de aumentar el entrenamiento con más datos de entrada.

	3 letras	5 letras	10 letras
% COMBINACIONES ENTRADA	56,90%	0,08%	0,00%
PALABRAS CORRECTAS	74,00%	10,50%	0,00%
LETRAS CORRECTAS	90,63%	70,58%	23,42%

FIGURA 4.7: Pruebas con 10.000 combinaciones de entrada

Por último, en la figura 4.8 muestra los resultados de la prueba con un entrenamiento de 50.000 combinaciones de entrada. El modelo de 3 letras prácticamente acierta todas las letras, pero es cierto que el entrenamiento es completo a nivel de combinaciones.

	3 letras	5 letras	10 letras
% COMBINACIONES ENTRADA	100,00%	0,42%	0,00%
PALABRAS CORRECTAS	74,00%	44,70%	0,00%
LETRAS CORRECTAS	93,12%	87,65%	5,32%

FIGURA 4.8: Pruebas con 50.000 combinaciones de entrada

En las figuras 4.9 y 4.10, se muestra como varía el porcentaje de acierto de palabras y letras respectivamente, con el aumento de las combinaciones del dataset de entrada.

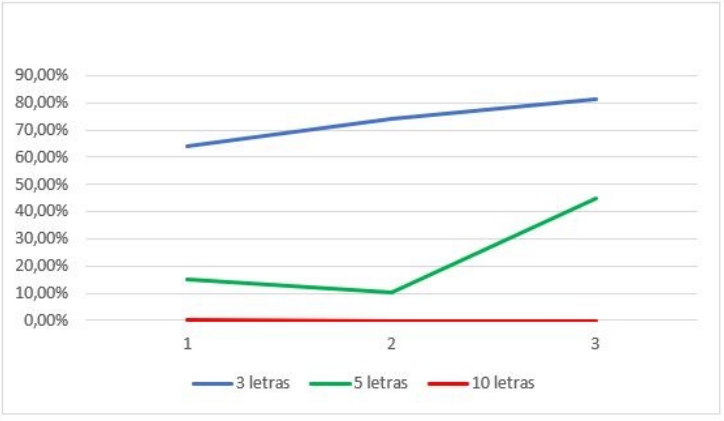

FIGURA 4.9: Evolución del acierto de palabras

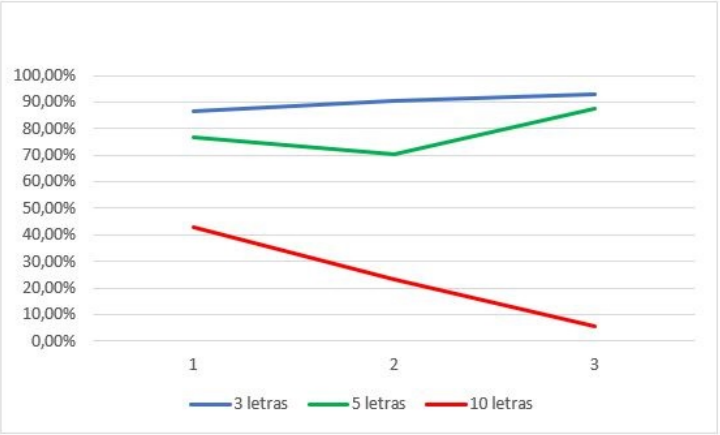

FIGURA 4.10: Evolución del acierto de letras

Las cadenas de 3 caracteres tienen un porcentaje alto de acierto, tanto de letras como de palabras completas. Es normal que esto suceda debido a la baja complejidad, el número de combinaciones (17.576) y la alta cantidad de muestras usadas para el entrenamiento. Además de esto, los hiperparámetros usados para configurar la red, se han elegido en base a pruebas sobre este caso, es decir, la función GridSearchCV trabajó contra palabras de 3 caracteres.

Los resultados obtenidos para cadenas de 5 letras son bastante buenos, casi al nivel del anterior caso. Al contrario sucede con palabras de 10 letras. En este caso, la red se rompe y el porcentaje de aciertos es casi nulo. Seguramente se deba a la configuración de la red, que como hemos mencionado no se había optimizado para este caso.

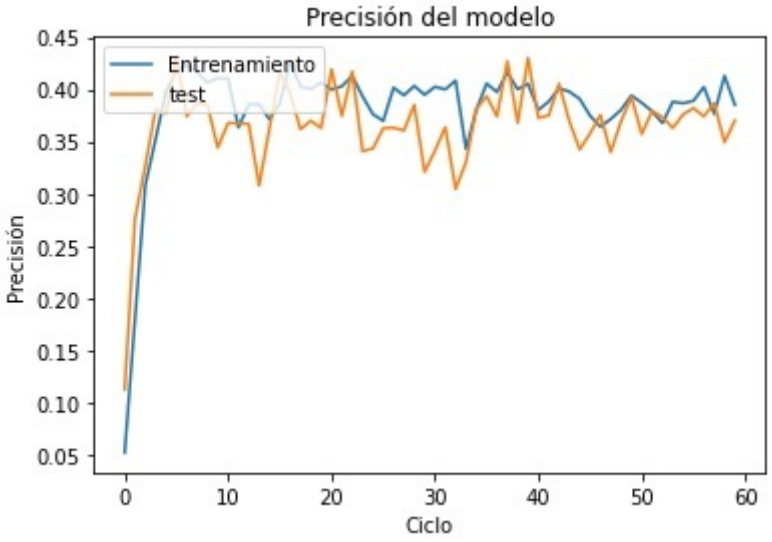

FIGURA 4.11: Precisión Red Neuronal Cifrado César

Las figuras 4.11 y 4.12 muestran la precisión y pérdida de nuestro modelo con cadenas de cinco caracteres y un dataset de diez mil combinaciones. A partir del gráfico de precisión (figura 4.11) podemos ver que el modelo tiene un rendimiento similar tanto en el entrenamiento como en la validación, por

lo tanto, el modelo no ha sobre-aprendido el conjunto de datos utilizado. También en ambas gráficas podemos ver el ruido que tiene el modelo, por un sobreajuste u overfitting, es decir, nuestro modelo en algunos casos no es capaz de reconocer nuevos datos de entrada. Esto puede ser debido al tamaño del dataset o al ajuste de hiperparámetros realizado inicialmente.

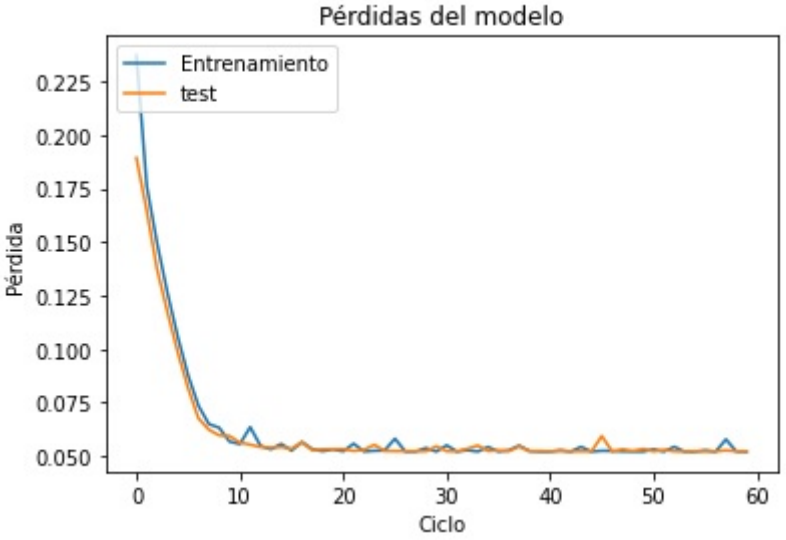

FIGURA 4.12: Pérdida Red Neuronal Cifrado César

A partir de este punto, se podría tratar de optimizar aún más nuestro modelo, pero el objetivo principal de este trabajo no era optimizar la red neuronal contra el cifrado César, sino el de probar el rendimiento de un modelo contra la criptografía de curvas elípticas, que veremos en la siguiente sección.

Otra visión de los resultados, es el mostrar un caso real de prueba. Podemos ver la salida esperada de las cadenas generadas aleatoriamente y la salida que devuelve la red neuronal. Se utiliza el carácter '-' cuando no se devuelve el correcto.

Salida esperada: 'efkzrn', 'evymta', 'eufork', 'jhwgml', 'pupwvy', 'anttwe', 'slagap', 'ttbfkn', 'otelpe'

Salida devuelta: 'efkz-n', 'ivymta', 'eufo-k', 'jhwgml', 'pupwv-', 'anttwe', 'slaiap', 'ttbokn', 'otelpe'

Otra prueba más visual ha sido la de introducir mi nombre en ROT3 (figura 4.1) como se puede ver en la figura 4.13.

```
Palabra encriptada =  ['mdylhu']
1/1 [==============================] - 0s 22ms/step
Palabra devuelta por la red ['javier']
```

FIGURA 4.13: Prueba con mi nombre encriptado

Lo más importante y el objetivo de este apartado era lograr un buen porcentaje de acierto para nuestra red, para poder confirmar el uso correcto de las herramientas usadas a lo largo de su desarrollo. Ahora podemos afrontar el objetivo real del trabajo y tratar de analizar el comportamiento de la red frente a la criptografía de curvas elípticas.

4.3 Red neuronal curvas elípticas

Esta sección está dedicada a la implentación de una red neuronal que va a poner a prueba la seguridad de la criptografía de curvas elípticas. Para ello, al igual que en la sección de la red neuronal contra el cifrado César (4.2), nos ayudaremos en el desarrollo con la biblioteca Keras [Cho].

Antes de comenzar con la descripción de la red neuronal, la implementación y los resultados, es importante señalar que la complejidad que presenta este tipo la criptografía de curvas elípticas es bastante alta, tal y como se puedo ver a lo largo del capítulo 2 donde se explica en profundidad su propiedades y como trabajan los diferentes algoritmos de cifrado con las curvas elípticas.

4.3.1 Descripción

Nuestra red neuronal contra la criptografía de curvas elípticas va a trabajar en concreto contra el criptosistema de cifrado/descifrado ElGamal. Por recordar brevemente, este criptosistema es un algoritmo matemático de criptografía asimétrica que se basa en el problema del logaritmo discreto. El algoritmo trabaja con un par de claves, una pública para el cifrado del mensaje y otra privada para el descifrado. Para ver el funcionamiento del algoritmo en profundidad, se puede volver al apartado donde se explica (2.2.5).

Como el propósito de esta prueba es el de analizar el comportamiento de este algoritmo de cifrado en curvas elípticas, nuestro diseño se basa en la propia idea del algoritmo y vamos a tratar que la red a partir la clave pública del cifrado como entrada, trate de devolver la clave privada que se usa para el descifrado.

Para la implementación de este diseño vamos a contar con la ayuda de una librería pública de GitHub [Cha] que genera los pares de claves públicas y privadas que necesitamos para entrenar a la red neuronal.

Tanto en el caso de la libería pública, como en el de la implementación de nuestra red neuronal, se trabaja sobre la curva Curve25519[3], que está diseñada para su uso con el esquema de intercambio de clave Diffie-Hellman (2.2.3). El criptosistema de ElGamal a su vez se basa en el intercambio Diffie-Hellman, por lo que la curva es perfectamente válida para nuestras pruebas.

En el siguiente código se puede ver la función de generar el par de claves que usaremos como datos de entrada y salida para la red neuronal. Para más información sobre el código de las funciones de generación de las claves privadas y públicas ver el anexo C o en el repositorio de GitHub [Cha].

```
def gen_keypair(curve: Curve, randfunc: Callable = None) -> Tuple[int, Point]:
  randfunc = randfunc or urandom
  private_key = gen_private_key(curve, randfunc)
  public_key = get_public_key(private_key, curve)
  return private_key, public_key
```

[3] https://hmong.es/wiki/Curve25519/

Una las complicaciones encontradas para que la implementación de este diseño es la conversión del par de claves para que la red neuronal no tenga problema a la hora de trabajar con los valores de entrada. Es necesario recordar que las redes neuronales artificiales tienen un mejor rendimiento al trabajar directamente con bits, pero la entrada para este diseño es la clave pública que es un punto de coordenadas (X, Y). La decisión tomada (ver figura 4.14) consta de formar un número con la concatenación de los valores X, Y de las coordenadas en binario. También es necesario tener en cuenta que en algunas ocasiones el tamaño de las coordenadas varía, por lo que en estos casos se rellena con ceros por la izquierda hasta obtener el total de dos números de 256 bits. Por consiguiente, la entrada de la red neuronal en este caso tendrá de 512 neuronas. En el caso de la salida, la clave privada es un número entero y estará formado por una salida de 256 neuronas en formato binario.

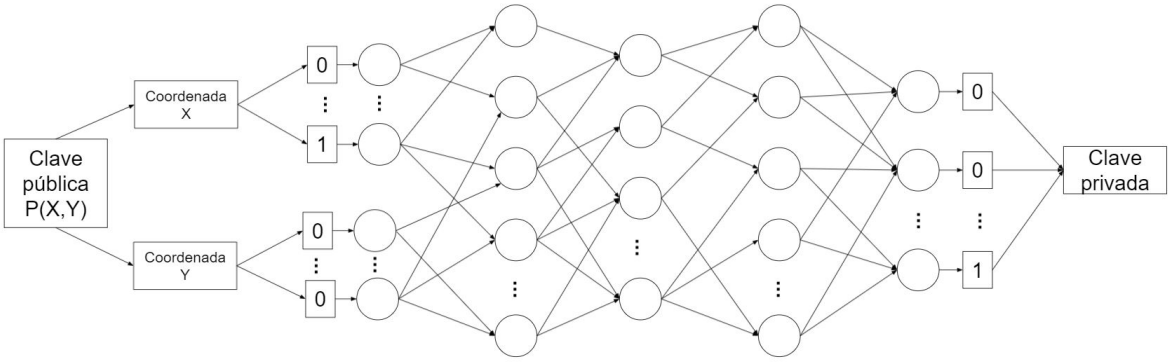

FIGURA 4.14: Diseño para la red neuronal de curvas elípticas

4.3.2 Implementación del código

Para facilitar la comprensión del trabajo y no cargar el contenido de este capítulo, se ha incluido el código utilizado para el desarrollo del ejercicio en el apéndice B. El código está desarrollado en python y para su ejecución es necesario el uso de ficheros de texto plano con extensión ".py" (o ".pyw"). Además, para su ejecución es necesario el uso de un terminal, un editor IDLE o un entorno interactivo.

4.3.3 Resultados obtenidos

Antes de comenzar a exponer los resultados obtenidos al poner a prueba el modelo de la red neuronal contra la criptografía de curvas elípticas, es necesario recordar que la complejidad que presenta este tipo de criptografía es significativamente superior a la del cifrado César del apartado anterior. Esta complejidad es el principal condicionante a la hora de analizar el aprendizaje de esta red neuronal.

Durante la búsqueda del mejor diseño y la configuración de los hiperparámetros de la red neuronal, al igual que en el apartado anterior, se ha utilizado la función GridSearchCV[4].

[4] https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html

Tras la elección de los hiperparámetros, se ha entrenado el modelo y se ha testeado con los resultados de precisión y pérdida que se muestran en las figuras 4.15 y 4.16, respectivamente. La curva de aprendizaje en el caso de la precisión (fig. 4.15), no consigue aumentar, por lo que nuestro modelo no está siendo capaz de distinguir el ruido y encontrar un patrón que logra el resultado deseado. También podemos ver que en el caso de la pérdida (fig. 4.16), la curva de entrenamiento, ha empezado a aumentar tras el ciclo 100 y está aumentando el error, al contrario de lo que debería hacer un modelo que esté aprendiendo correctamente. Para obtener más información sobre las curvas de aprendizaje y la mejora del modelo, se recomienda la lectura del libro *Learning curve models and applications: Literature review and research directions* [AF11]

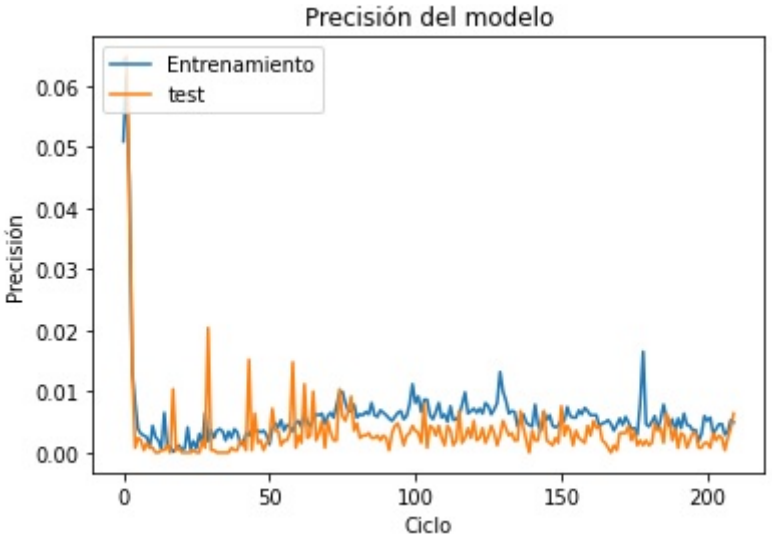

FIGURA 4.15: Precisión Red Neuronal ECC

Como hemos mencionado anteriormente, la complejidad a la que nos enfrentamos en este problema es el principal punto a tener en cuenta. ¿Existe función con potencia y base para resolver el Problema del Logaritmo Discreto? Una solución posible es la fuerza bruta, pero se trata de una solución demasiado lenta. Una explicación más a fondo podemos encontrar en el apartado dedicado al Problema del Logartimo Discreto2.2.2, donde ya mencionamos la complejidad que supone el cálculo de la inversa es muy sencilla en términos computacionales en comparación con el cálculo del logaritmo discreto para ciertos grupos y en esta caso concreto, en trabajando con curvas elípticas. El problema no tiene solución en un tiempo razonable utilizando aritmética modular, pero en nuestro caso, la idea de probar a resolver este problema a través de las redes neuronales, era precisamente por la velocidad y el rendimiento que nos proporciona esta tecnología, aunque tampoco hemos logrado los resultados deseados.

Como era de esperar, nuestra red no ha sido capaz de romper el cifrado y devolver la clave privada esperada, y mucho menos, de encontrar patrones para lograr el aprendizaje. Después de numerosas pruebas y ajuste de los hiperparámetros, la precisión se mantenía cercana a cero.

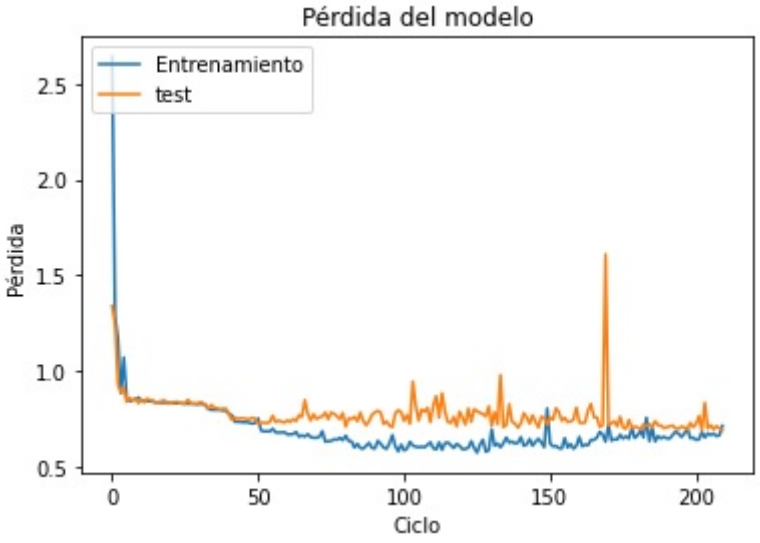

FIGURA 4.16: Pérdida Red Neuronal ECC

La optimización de la red no ha sido posible, a pesar de usar diferentes niveles de capas o modificar los valores de los hiperparámetros. Tampoco ha servido la ayuda de la función GridSearchCV. Uno de los motivos que nos lleva al fracaso del aprendizaje de nuestro modelo puede ser el uso de keras. Esta biblioteca nos ha facilitado el desarrollo y la implementación de la red neuronal, pero también limita las pruebas y las modificaciones de código a niveles más bajos, ya que lo trabajamos muy superficialmente. Podría ser interesante en un futuro o en otro trabajo continuar por este camino y desarrollar el modelo sin la ayuda de keras, pero el tiempo limitado impide hacerlo en este proyecto.

Impacto social, ambiental, ético y legal

Este trabajo se ha realizado con el objetivo de poner a prueba el rendimiento de diferentes tecnologías, pero sin lograr unos resultados que puedan tener interés social más allá del propio ámbito educativo. La posibilidad de encontrar alguna vulnerabilidad en la seguridad de la criptografía de curvas elípticas era mínima, por lo que el desarrollo de este trabajo no suponía ningún problema a nivel social. El avance de las tecnologías es evidente y la seguridad de la información nunca estará completamente garantizada. De esta manera, el uso de mejores herramientas para mejorar la seguridad, implica claramente múltiples beneficios para la sociedad.

Por la naturaleza de este proyecto, no tiene ninguna implicación ambiental, ya que únicamente se trataba de analizar el comportamiento de las redes neuronales y la criptografía de curvas elípticas. Aunque uno de los puntos comentados a lo largo de los apartados, ha sido la mejora que supone el uso de la criptografía de curvas elípticas frente a la criptografía tradicional. Este punto puede tener un enfoque ambiental, ya que esta mejora de rendimiento, puede implicar una mejora ambiental.

A nivel ético y legal, el trabajo estaba enfocado en poner a prueba tecnologías que se usan ya en nuestra sociedad, pero sin salir del terreno legal y con el objetivo de ayudar siempre a mejorar la sociedad.

Trabajo futuro

A lo largo de este proyecto hemos trabajado sobre dos ramas, las criptografía de curvas elípticas y las redes neuronales. Se podría seguir ampliando conocimiento en el futuro en ambas direcciones, por ello, se proponen las siguientes ideas o expansiones:

- **Cambiar los datos de entrada y salida de la red:** En este trabajo, hemos trabajo a través de la clave privada como datos de entrada y la clave pública como salida. Otra opción viable podría ser repetir el modelo utilizado para la red neuronal contra el cifrado César 4.2, es decir, usar palabras cifradas como entrada y descifradas como salida para las pruebas con criptografía de curvas elípticas 4.3.

- **Probar contra otros algoritmos criptográficos:** Nuestra red neuronal contra la criptografía curvas elípticas se basa en el criptosistema de ElGamal (2.2.5). En el futuro, se podría implementar la red contra otros algoritmos como Diffie-Hellman o DSA.

- **Crear nuestro propio código de la red neuronal:** La red neuronal ha sido implementada gracias a la biblioteca de código abierto Keras. Su uso es sencillo y modular, de forma que es muy amigable para los usuarios. Podría ser interesante profundizar en la implementación del código para obtener mejor rendimiento.

- **Estudiar otros tipos de ataques contra curvas elípticas**: Comprobar la seguridad de la criptografía de curvas elípticas frente a otros tipos de ataques, como por ejemplo, ataques mediante computación cuántica[5] y compararlo con los resultados de las redes neuronales.

[5] https://www.welivesecurity.com/la-es/2016/06/14/computacion-cuantica-armagedon-criptografico/

Conclusiones

A modo de cierre de este trabajo, podemos señalar que se ha tratado de aprender los fundamentos matemáticos (capítulo 1) necesarios para una correcta comprensión de la criptografía de curvas elípticas (capítulo 2) y el funcionamiento de una red neuronal artificial (capítulo 3). Tras una parte teórica, la parte práctica de este trabajo ha tratado de comprobar la seguridad que presentan dos diferentes sistemas de cifrado frente al aprendizaje automático de una red neuronal (capítulo 4). En primer lugar, la red neuronal ha sido capaz de aprender a resolver el cifrado César de una manera sencilla. Esto se debe a lo sencillo que es este algoritmo de cifrado y a la poca complejidad que presenta. En segundo lugar, la alta complejidad que presenta la criptografía de curvas elípticas ha dificultado el trabajo de la red neuronal, que no ha conseguido romper su seguridad.

La potencia de nuestro equipo y lo sencilla que era nuestra red neuronal, han sido otros factores que frente a la alta complejidad de este tipo de criptografía, han dado como resultado un sistema no vulnerable a nuestro estudio.

A pesar de estos resultados, desde el inicio del trabajo se conocía la dificultad de conseguir resultados positivos, por lo que el objetivo del trabajo era corroborar esta idea.

Apéndice A

Red neuronal cifrado César

```
def generate_words (number_words,test,X_train):
  words = []
  if(test == 0):
    for x in range(number_words):
      words.append(''.join(random.SystemRandom().choice(string.ascii_letters).lower()
                   for _ in range(word_size)))
  else:
    for x in range(number_words):
      repeated = 1
      while(repeated == 1):
        word = (''.join(random.SystemRandom().choice(string.ascii_letters).lower()
                  for _ in range(word_size)))
        if(word not in X_train):
          words.append(word)
          repeated = 0
  encrypted_words = []
  for word in words:
    cadena = ""
    for x in range(0,word_size):
      cadena += chr((((ord(word[x]) - 97) + shift) % 26) + 97)
    encrypted_words.append(cadena)

  return encrypted_words, words
```

```
def wordToNumber(words):
  word_id = {'a': 0, 'b': 1, 'c': 2, 'd': 3, 'e': 4, 'f': 5, 'g': 6, 'h': 7,
    'i': 8, 'j': 9,'k': 10, 'l': 11, 'm': 12, 'n': 13, 'o': 14,'p': 15, 'q': 16,
    'r': 17, 's': 18, 't': 19,'u': 20, 'v': 21, 'w': 22, 'x': 23, 'y': 24, 'z': 25}
  lista = []
  x = 0
  one_hot_encoding = np.zeros((len(words),len(word_id)*word_size))
  for word in words:
    for i in range(0,word_size):
```

```
      one_hot_encoding[x][word_id[word[i]]+(26*i)] = 1
    x+=1
  return one_hot_encoding
```

```
def numberToWord(numbers):
  word_id = {0:'a', 1:'b', 2:'c', 3:'d', 4:'e', 5:'f', 6:'g', 7:'h', 8:'i',
    9:'j', 10: 'k', 11:'l', 12:'m', 13:'n', 14:'o', 15:'p', 16:'q', 17:'r',
    18:'s', 19:'t', 20:'u', 21:'v', 22:'w', 23:'x', 24:'y', 25:'z'}
  words = []
  for number in numbers:
    cadena = ""
    for i in range(0,word_size):
      car = '-'
      for x in range(0,len(word_id)):
      if(number[x+(i*26)] > 0):
        car =  word_id[x]
      cadena += car
    words.append(cadena)
  return words
```

```
def crearModelo(loss='poisson', activation='relu', unit1=120, unit2=80, optimizer='
                                          RMSprop'):
  # Definir modelo
  model = Sequential()
  # Capas
  model.add(Dense(nn[4], input_dim=(26*word_size)))
  model.add(Dense(unit1, activation='relu'))
  model.add(Dense(unit2, activation='relu'))
  model.add(Dense(unit1, activation='relu'))
  model.add(Dense(nn[4], activation=activation))
  # Compilar modelo
  model.compile(loss='binary_crossentropy', optimizer=optimizer,metrics=['accuracy'])
  return model
```

```
batch_size = [200,300,500]
epochs = [50, 100, 150]
optimizer = ['SGD', 'RMSprop', 'Adagrad','Adam', 'Adamax', 'Nadam']
loss = ['binary_crossentropy','categorical_crossentropy','poisson']
activation = ['softmax','relu','sigmoid','softplus']
unit1 = [80,100,120]
unit2 = [100]
```

```
model = KerasClassifier(build_fn=crearModelo, epochs=100, batch_size=10, verbose=0)
param_grid = dict(batch_size=batch_size, epochs=epochs, optimizer=optimizer, unit1=
                                          unit1, unit2=unit2, loss=loss, activation=
                                          activation)
grid = GridSearchCV(estimator=model, param_grid=param_grid, n_jobs=-1, cv=3)
grid_result = grid.fit(X_train, Y_train)
```

```
# summarize results
print("Best: %f using %s" % (grid_result.best_score_, grid_result.best_params_))
means = grid_result.cv_results_['mean_test_score']
stds = grid_result.cv_results_['std_test_score']
params = grid_result.cv_results_['params']
for mean, stdev, param in zip(means, stds, params):
print("%f (%f) with: %r" % (mean, stdev, param))
```

Apéndice B

Red neuronal curva elíptica

```
from keras.models import Sequential,Model
from keras.layers import SimpleRNN, Input, Dense
import numpy as np
import matplotlib.pyplot as plt
from keras.losses import sparse_categorical_crossentropy
from sklearn.model_selection import GridSearchCV
from keras.wrappers.scikit_learn import KerasClassifier
import string
import random
import hashlib

from ecc.curve import Curve25519
from ecc.key import gen_keypair
from ecc.cipher import ElGamal

# Parametros configuracion inicial
word_size = 6
hash = "sha1" # "sha256"

# Declaracion de parametros
nn = [256*2,300,500,300,256]

pub_train, pri_train = generate_keys(10000)
pub_test, pri_test = generate_keys(1000)
```

```
def crearModelo(loss='poisson',activation='relu', unit1=500, unit2=750,optimizer='
                                                RMSprop'):
  # Definir modelo
  model = Sequential()
  # Capas
  model.add(Dense(nn[4], input_dim=512))
  model.add(Dense(unit1, activation='relu'))
  model.add(Dense(unit2, activation='relu'))
```

```
  model.add(Dense(unit1, activation='relu'))
  model.add(Dense(nn[4], activation=activation))
  # Compilar modelo
  model.compile(loss='binary_crossentropy', optimizer=optimizer, metrics=['accuracy']
                                            )
  return model
```

```
model = crearModelo()

snn = model.fit(pub_train, pri_train, epochs=120, batch_size=400)

scores = model.evaluate(pub_test, pri_test)
print("%s: %.2f%%" % (model.metrics_names[1], scores[1]*100))

pred = model.predict(pub_test, batch_size=400, verbose=1)
print(pred)
```

```
def generate_keys (number_keys):
  pub_keys = np.zeros((number_keys,512))
  pri_keys = np.zeros((number_keys,256))
  for x in range(number_keys):
    pri, pub = gen_keypair(Curve25519)
    y=255
    while pub.x != 0:
      pub_keys[x][y]= pub.x % 2
      pub.x //= 2
      y-=1
    y=511
    while pub.y != 0:
      pub_keys[x][y]= pub.y % 2
      pub.y //= 2
      y-=1
    y=255
    while pri != 0:
      pri_keys[x][y]= pri % 2
      pri //= 2
    y-=1
  return pub_keys, pri_keys
```

```
batch_size = [300,500]
epochs = [300, 500]
optimizer = ['SGD', 'RMSprop', 'Adagrad', 'Adadelta', 'Adam', 'Adamax', 'Nadam']
loss = ['binary_crossentropy','categorical_crossentropy']
activation = ['relu','sigmoid','softmax']
dense = [1,2]
unit1 = [500,750]
unit2 = [750]
```

```
model = KerasClassifier(build_fn=crearModelo, epochs=100, batch_size=10, verbose=0)

param_grid = dict(batch_size=batch_size, epochs=epochs, optimizer=optimizer, unit1=
                                        unit1, unit2=unit2, loss=loss, activation=
                                        activation)
grid = GridSearchCV(estimator=model, param_grid=param_grid, n_jobs=-1, cv=3)

grid_result = grid.fit(pub_train, pri_train)
# summarize results
print("Best: %f using %s" % (grid_result.best_score_, grid_result.best_params_))
means = grid_result.cv_results_['mean_test_score']
stds = grid_result.cv_results_['std_test_score']
params = grid_result.cv_results_['params']
for mean, stdev, param in zip(means, stds, params):
  print("%f (%f) with: %r" % (mean, stdev, param))
```

```
# summarize results
print("Best: %f using %s" % (grid_result.best_score_, grid_result.best_params_))
means = grid_result.cv_results_['mean_test_score']
stds = grid_result.cv_results_['std_test_score']
params = grid_result.cv_results_['params']
for mean, stdev, param in zip(means, stds, params):
  print("%f (%f) with: %r" % (mean, stdev, param))
```

Apéndice C

Generador de claves

```
def gen_keypair(curve: Curve, randfunc: Callable = None) -> Tuple[int, Point]:
  randfunc = randfunc or urandom
  private_key = gen_private_key(curve, randfunc)
  public_key = get_public_key(private_key, curve)
  return private_key, public_key
```

```
def gen_private_key(curve: Curve, randfunc: Callable = None) -> int:
  order_bits = 0
  order = curve.n

  while order > 0:
    order >>= 1
    order_bits += 1

  order_bytes = (order_bits + 7) // 8
  extra_bits = order_bytes * 8 - order_bits

  rand = int(hexlify(randfunc(order_bytes)), 16)
  rand >>= extra_bits

  while rand >= curve.n:
    rand = int(hexlify(randfunc(order_bytes)), 16)
    rand >>= extra_bits

  return rand
```

```
def get_public_key(d: int, curve: Curve) -> Point:
  return d * curve.G
```

Bibliografía

[AF11] Anzanello, M. J. and Fogliatto, F. S. *Learning curve models and applications: Literature review and research directions. International Journal of Industrial Ergonomics*, 41(5):573–583, 2011.

[Bis16] Bishop, C. M. *Pattern Recognition and Machine Learning*. Springer New York, 2016.

[Cha] ChangLiu. *Python package for ecc and elgamal elliptic curve encryption*. https://github.com/lc6chang/ecc-pycrypto/. Último acceso: 18-05-2022.

[Cho] Chollet, F. *Keras documentation*. https://keras.io/. Último acceso: 03-07-2022.

[DERW86] David E. Rumelhart, G. E. H. and Williams, R. J. *Learning representations by back-propagating errors. Nature*, 1986.

[Fau08] Fausett, L. V. *Fundamentals of neural networks: architectures, algorithms, and applications*. Pearson Education, 2008.

[Fou] Foundation, P. S. *Python documentation*. https://docs.python.org/3/. Último acceso: 03-07-2022.

[HMV04] Hankerson, D., Menezes, A. and Vanstone, S. *Guide to Elliptic Curve Cryptography*. Springer Professional Computing, 2004.

[JE07] José, D. and Eugenio, H. *Números, grupos y anillos*. Addison-Wesley Iberoamericana, 2007.

[JMM] Josep M. Miret, U. d. L., Javier Valera & Magda Valls Grupo de Investigación Cryptography & Graphs: www.cig.udl.cat. *Criptografía con curvas elípticas*. http://www.criptored.upm.es/crypt4you/temas/ECC/leccion1/leccion1.html. Último acceso: 20-06-2022.

[MEM18] Martínez, V. G., Encinas, L. H. and Muñoz, A. M. *Criptografía con curvas elípticas*. Consejo Superior de Investigaciones Cientificas, 2018.

[MP69] Minsky, M. and Papert, S. *Perceptrons: An Introduction to Computational Geometry*. MIT Press, Cambridge, MA, USA, 1969.

[Mun] Munoz, J. G. *Código redes neuronales.* https://github.com/JGM10/Proyecto-Final-Grado-ETSISI/. Último acceso: 18-05-2022.

[Nav16] Navarro, G. *Un curso de álgebra.* Universitat de València, 2016.

[Sti06] Stinson, D. *Cryptography: Theory and Practice, Third Edition*, pp. 233–280. Chapman & Hall, CRC, London, UK, 2006.

[Was08] Washington, L. C. *Elliptic curves: number theory and cryptography.* Chapman & Hall/CRC, 2008.