



Redes neuronales aplicadas al criptoanálisis del Advanced Encryption Standard.

Pedro Novas Otero
Grado en Ingeniería Informática
Inteligencia Artificial

David Isern Alarcón
Carles Ventura Royo

05/06/2018



Esta obra está sujeta a una licencia de Reconocimiento-NoComercial-SinObraDerivada [3.0 España de Creative Commons](https://creativecommons.org/licenses/by-nc-nd/3.0/es/)

FICHA DEL TRABAJO FINAL

Título del trabajo:	<i>Redes neuronales aplicadas al criptoanálisis de Advanced Encryption Standard.</i>
Nombre del autor:	<i>Pedro Novas Otero</i>
Nombre del consultor/la:	<i>David Isern Alarcón</i>
Nombre del PRA:	<i>Carles Ventura Royo</i>
Fecha de entrega:	06/2018
Titulación:	<i>Grado en Ingeniería Informática</i>
Área del Trabajo Final:	<i>Inteligencia Artificial</i>
Idioma del trabajo:	<i>Español</i>
Palabras clave	<i>Redes neuronales, criptoanálisis , AES</i>
Resumen del Trabajo:	
<p>En el presente trabajo se realiza un estudio sobre la aplicación de redes neuronales al criptoanálisis del algoritmo Advanced Encryption Standard.</p> <p>El criptoanálisis propuesto consiste en entrenar una red neuronal con ejemplos formados por pares de textos en plano y sus correspondientes textos cifrados para, posteriormente, a partir de un texto cifrado predecir el texto en plano correspondiente.</p> <p>El estudio muestra capacidad de aprendizaje, aunque reducida, de las redes neuronales testadas en versiones reducidas del algoritmo. Sin embargo, en las versiones completas del algoritmo las pruebas muestran aleatoriedad en las predicciones.</p> <p>Por lo tanto, en base a los resultados obtenidos, el criptoanálisis desarrollado no se puede considerar un ataque eficaz contra el algoritmo objeto de estudio.</p>	

Abstract:

This study will explore the application of neural networks to the cryptanalysis of the Advanced Encryption Standard algorithm.

The proposed cryptanalysis consists of training a neural network with examples comprised of pairs of plaintexts and their respective cyphertexts, so that it can subsequently obtain the correct cyphertext having access only to the plaintext.

The study shows learning capacity, albeit reduced, when the neural networks are tested against a reduced-round version of the algorithm. However, when the full-round version of the algorithm is used, the tests show randomness.

Therefore, according to the results obtained, the proposed cryptanalysis cannot be considered an effective attack against the algorithm studied.

Índice

1. Introducción.....	1
1.1 Contexto y justificación del trabajo.....	1
1.2 Objetivos del Trabajo.....	2
1.3 Planificación del Trabajo.....	3
1.4 Breve resumen de productos obtenidos.....	4
1.5 Breve descripción de los otros capítulos de la memoria.....	4
2. Redes Neuronales.....	6
2.1 Introducción a las redes neuronales.....	6
2.2 Representación de la información en una red neuronal.....	6
2.4 Evolución y tipos de redes neuronales.....	10
2.5 Aplicaciones de las redes neuronales a la criptografía.....	11
3. Algoritmo de cifrado en bloque Advanced Encryption Standard.....	12
3.1. Cifrado en bloque.....	12
3.2 Modos de operación del cifrado en bloque.....	13
3.3 Surgimiento del Advanced Encryption Standard.....	13
3.4 Funcionamiento del Advanced Encryption Standard.....	13
3.5 Criptoanálisis de Advanced Encryption Standard.....	15
4. Implementación práctica de una red neuronal.....	17
4.1 Justificación de las tecnologías escogidas para la implementación.....	17
4.2 Diseño y programación de la red neuronal.....	17
5. Aplicación de la red neuronal al criptoanálisis de AES.....	20
5.1 Generación de los datos de entrenamiento y testado de la red neuronal.....	20
5.2 Entrenamiento y testado de la red neuronal.....	21
5.3 Resultados del estudio.....	24
5.4 Conclusiones sobre los resultados.....	27
5.5 Comparación de los resultados obtenidos con otros métodos de criptoanálisis aplicados a AES.....	27
6. Conclusiones.....	28
7. Glosario.....	29
8. Bibliografía.....	30
9. Anexos.....	32
9.1 Código correspondiente a la generación de datos.....	32
9.2 Código correspondiente a la red neuronal perceptrón multicapa.....	33
9.3 Código correspondiente a la red neuronal basada en capas de tres dimensiones.....	34
9.4 Código correspondiente a las pruebas de test realizadas.....	35

Lista de figuras

- *Figura 1: Diagrama de Gantt*
- *Figura 2: Esquema conceptual de una neurona*
- *Figura 3: Función de activación escalonada*
- *Figura 4: Función de activación sigmoideal*
- *Figura 5: Función de activación a tramos*
- *Figura 6: Ejemplo de arquitectura de una red neuronal*
- *Figura 7: Esquema conceptual de una arquitectura de perceptrón multicapa*
- *Figura 8: Esquema del funcionamiento general de AES*
- *Figura 9: Arquitectura basada en el perceptrón multicapa*
- *Figura 10: Arquitectura basada en capas de 3 dimensiones*
- *Figura 11: Función de activación aplicada a los bits de salida*
- *Figura 12: Evolución función de coste perceptrón multicapa*
- *Figura 13: Evolución función de coste red neuronal basada en capas 3D*

1. Introducción

1.1 Contexto y justificación del trabajo

En los últimos años el aumento de la capacidad de computación y el aumento de capacidad de almacenaje de los sistemas de la información con el consecuente aumento de datos disponibles sobre muchos procesos han traído consigo un especial interés por la Inteligencia Artificial y sus posibles aplicaciones a la hora de sacar partido a estos datos. En este trabajo se estudiará una de estas aplicaciones de la Inteligencia Artificial, en concreto de las redes neuronales, a una disciplina muy relevante en la seguridad de los sistemas de comunicación: la criptografía. En este caso se estudiará la eficacia de las redes neuronales en un ataque de texto en claro conocido.

Los ataques de texto en claro conocido son un método de criptoanálisis donde el atacante tiene acceso tanto a el texto en plano como al texto cifrado. Este tipo de ataques han sido muy utilizados en muchos contextos teniendo una gran relevancia, por ejemplo, en la Segunda Guerra Mundial donde Inglaterra utilizó este tipo de ataques para descifrar textos de carácter militar enviados por los soldados alemanes.

Actualmente se han realizado investigaciones de la aplicación de las Redes Neuronales Artificiales al criptoanálisis de algoritmos de cifrado en bloque como DES y Triple-Des[7] utilizando ataques de texto en claro conocido demostrando su eficacia a la hora de reducir el número de ejemplos y de tiempo de computación necesarios para realizar estos ataques. Esta aplicación se basa en la capacidad de una Red Neuronal Artificial de determinar, a partir un conjunto de ejemplos de muestra, la función que está implícitamente determinada por dicho conjunto de ejemplos. En el caso concreto del ataque de texto en claro conocido, la Red Neuronal Artificial aproxima la función de descryptación a partir de ejemplos de duplas de textos en claro y textos cifrados.

En este trabajo se implementará una red neuronal en el lenguaje de programación Python y se estudiará la eficacia de la metodología descrita anteriormente en el algoritmo de cifrado AES (Advanced Encryption Standard) que substituyó a DES y es ampliamente utilizado en la actualidad. Este algoritmo ha mejorado la resistencia a los ataques de texto en claro conocido respecto sus predecesores, lo cual, hace interesante la investigación sobre una nueva vía para atacar este cifrado.

1.2 Objetivos del Trabajo

Generales:

- Estudiar la eficacia de un ataque de texto en claro conocido al algoritmo AES utilizando redes neuronales.

Específicos:

- Diseñar e implementar una red neuronal en el lenguaje de programación Python.
- Entrenar y testar la red neuronal mediante datos generados por el algoritmo AES.
- Realizar pruebas de ataques de texto en claro conocido utilizando la red neuronal implementada.
- De ser posible el ataque, cuantificar los recursos necesarios para llevar a cabo el ataque en términos de ejemplos de duplas de texto en claro y texto cifrado y de recursos computacionales.
- Comparar la efectividad de las redes neuronales aplicadas al criptoanálisis de AES con otros métodos de criptoanálisis.
- Extraer conclusiones sobre las posibilidades de la aplicación de las redes neuronales al criptoanálisis de AES.

1.3 Planificación del Trabajo

Planificación temporal:

Periodo	Tarea
Desde 20/03/18 Hasta 01/04/18	Investigación y redacción del marco teórico del trabajo: <ul style="list-style-type: none"> • Redes Neuronales Artificiales • AES
Desde 01/04/18 Hasta 20/04/18	Implementación de una red neuronal. <ul style="list-style-type: none"> • Diseño y programación en Python de una red neuronal Entrenamiento y testado de la red neuronal con los datos generados por el algoritmo AES
Desde 20/04/18 Hasta 23/04/18	Redacción del primer informe de seguimiento correspondiente a la PEC 2.
Desde 23/04/18 Hasta 18/05/18	Implementación de una red neuronal. <ul style="list-style-type: none"> • Diseño y programación en Python de una red neuronal • Entrenamiento y testado de la red neuronal con los datos generados por el algoritmo AES Estudio de la aplicación de la Red Neuronal al criptoanálisis de AES. <ul style="list-style-type: none"> • Pruebas de ataque de texto en claro conocido sobre AES y posibles modificaciones en la red neuronal • Comparación de los resultado con otros métodos de criptoanálisis • Extracción de conclusiones del estudio
Desde 18/05/18 Hasta 21/05/18	Redacción del segundo informe de seguimiento correspondiente a la PEC 3.
Desde 20/03/18 Hasta 05/06/18	Redacción de la memoria.
Desde 05/06/18 Hasta 13/06/18	Elaboración de la presentación.
Desde 13/06/18 Hasta 25/06/18	Defensa pública.

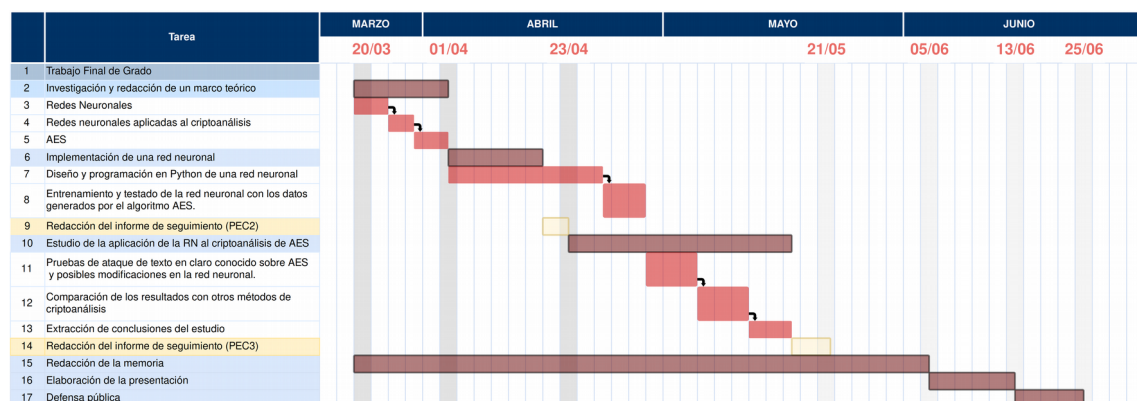


Figura 1: Diagrama de Gantt

Hitos:

Hito	Fecha aproximada de consecución
Conclusión de la redacción del marco teórico	01/04/18
Implementación de una red neuronal en Python	20/04/18
Obtención de los resultados de las pruebas del ataque de texto en claro conocido al algoritmo AES mediante redes neuronales.	07/05/18
Finalización de la comparación de los resultados de las pruebas con otros métodos de criptoanálisis.	14/05/18
Finalización de la extracción de conclusiones.	18/05/18
Finalización de la redacción de la memoria.	05/06/18
Finalización de la elaboración de la presentación.	13/06/18
Finalización de la defensa pública.	25/06/18

Priorización de los objetivos:

El objetivo principal de este trabajo es estudiar la eficacia de un ataque de texto en claro conocido al algoritmo AES utilizando redes neuronales pero esto no implica que la implementación práctica del ataque mediante redes neuronales en este algoritmo en concreto sea efectiva por lo que se establecerá un periodo máximo de pruebas. De esta forma, aunque los resultados de las pruebas no sean favorables esto no interferirá en los objetivos de comparación de los resultados con otros métodos de criptoanálisis y con la extracción de conclusiones del estudio.

Este periodo máximo de pruebas será de dos semanas: desde el 23/04/18 hasta el 07/05/18.

1.4 Breve resumen de productos obtenidos

Se ha obtenido un marco teórico que contextualiza el trabajo a través de una introducción a las redes neuronales y una explicación breve del algoritmo Advanced Encryption Standard.

Además, se ha obtenido una implementación práctica de las arquitecturas planteadas, unos casos de prueba y sus resultados.

1.5 Breve descripción de los otros capítulos de la memoria

Redes Neuronales: En este capítulo se introducen los conceptos básicos relacionados con las redes neuronales.

Algoritmo de cifrado en bloque Advanced Encryption Standard: En este capítulo contextualiza y se describe brevemente el funcionamiento del algoritmo Advanced Encryption Standard.

Implementación práctica de una red neuronal: En este capítulo se describen las tecnologías utilizadas para la implementación práctica del estudio y las

arquitecturas de las redes neuronales propuestas para realizar el ataque al algoritmo de cifrado.

Aplicación de la red neuronal al criptoanálisis de AES: En este capítulo se expone como se entrenan las diferentes arquitecturas de las redes neuronales y los resultados obtenidos por estas.

Conclusiones: En este capítulo se establecen las conclusiones globales del trabajo.

2. Redes Neuronales

2.1 Introducción a las redes neuronales

Las redes neuronales artificiales son un modelo computacional enmarcado en el aprendizaje automático, el cual, es una rama de la inteligencia artificial que estudia el desarrollo de mecanismos que permitan a las máquinas *aprender*.

El aprendizaje en un ser humano es un fenómeno complejo y abarca diferentes facetas como la adquisición de conocimiento declarativo, el desarrollo de habilidades cognitivas a través de la práctica, la organización del conocimiento o el descubrimiento de nuevos hechos a través de la experimentación. En el campo del aprendizaje automático el término aprender hace referencia a la capacidad de los sistemas de mejorar su rendimiento e inferir conocimiento a partir de la experiencia. Para ello, se utilizan diferentes modelos como las reglas difusas, la lógica de primer orden, las redes bayesianas o, el caso en el que se centrará este trabajo, las redes neuronales.

2.2 Representación de la información en una red neuronal

Las redes neuronales son un modelo computacional inspirado en la forma en la que los sistemas nerviosos biológicos, como el cerebro, procesan la información. Este modelo se representa mediante un grafo dirigido donde cada nodo representa una neurona y cada arista representa la conexión entre las dos neuronas.

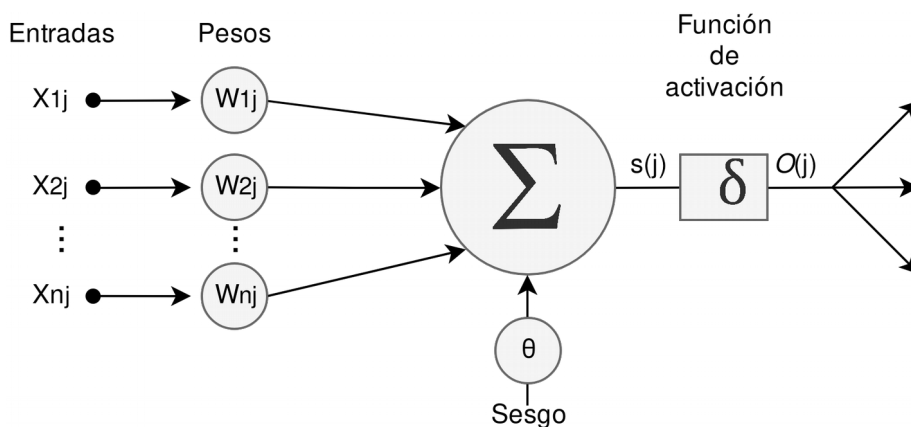


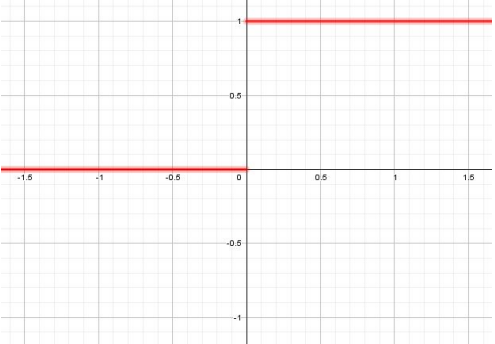
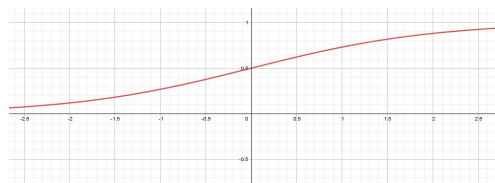
Figura 2: Esquema conceptual de una neurona

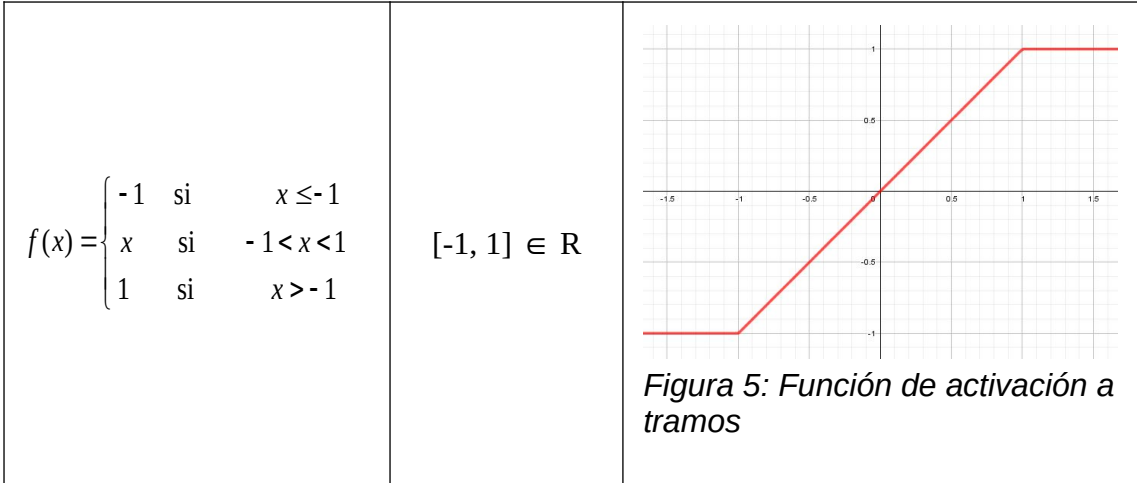
Una neurona recibe varias entradas que representan las conexiones con otras neuronas. Cada una de estas entradas tiene asociado un peso que determina la importancia de cada conexión para la salida de esa neurona. Para la neurona j -ésima estas entradas se representarán formalmente :

$X_{1j}, X_{2j}, \dots, X_{nj}$ y los pesos asociados a estas entradas se representan formalmente: $W_{1j}, W_{2j}, \dots, W_{nj}$. Además, se puede utilizar un valor constante para sesgar la salida de la neurona, este valor se representará mediante el símbolo θ .

De esta forma, cada neurona j -ésima produce una salida $s(j)$. Esta salida vendrá determinada por la siguiente fórmula $s(j) = \sum_{i=0}^n (X_{ij} \cdot W_{ij}) + \theta$. El resultado de esta fórmula puede ser cualquier número real, sin embargo, es posible que cuando se diseña la red neuronal se quiera que las neuronas se activen dentro de un rango determinado por lo que se utilizará una función de activación tipo umbral para determinar la salida final de la neurona. Existen diferentes funciones de activación. Por ejemplo, se podría utilizar una función tipo escalón para que la salida fuese 0 o 1 o la función sigmoide para que la salida sea un valor real entre 0 y 1.

Ejemplos de funciones de activación tipo umbral:

Expresión	Imagen de la función (rango de salida de la neurona)	Representación gráfica
$f(x) = \begin{cases} 0 & \text{si } x \leq 0 \\ 1 & \text{si } x > 0 \end{cases}$	$[0,1] \in \mathbb{N}$	 <p><i>Figura 3: Función de activación escalonada</i></p>
$f(x) = \frac{1}{1 + e^{-x}}$	$[0,1] \in \mathbb{R}$	 <p><i>Figura 4: Función de activación sigmoide</i></p>



La salida de esta función de activación se conectará a otras neuronas. Las conexiones ente estas neuronas determinará el tipo arquitectura, los cuales analizaremos en un apartado posterior. El objetivo de este conjunto de neuronas conectadas es aproximar, a partir de un conjunto numérico de ejemplos de muestra, la función que está implícitamente determinada por dicho conjunto de ejemplos. Una red neuronal es la representación del conjunto de pesos, conexiones y sesgos que caracterizan una función. Por lo tanto, la información de los ejemplos de entrada y de salida estará representada en un formato donde se le pueda hacer corresponder un número, en el rango apropiado, a unas neuronas de entrada y habrá unas neuronas que su salida represente el resultado de la función.

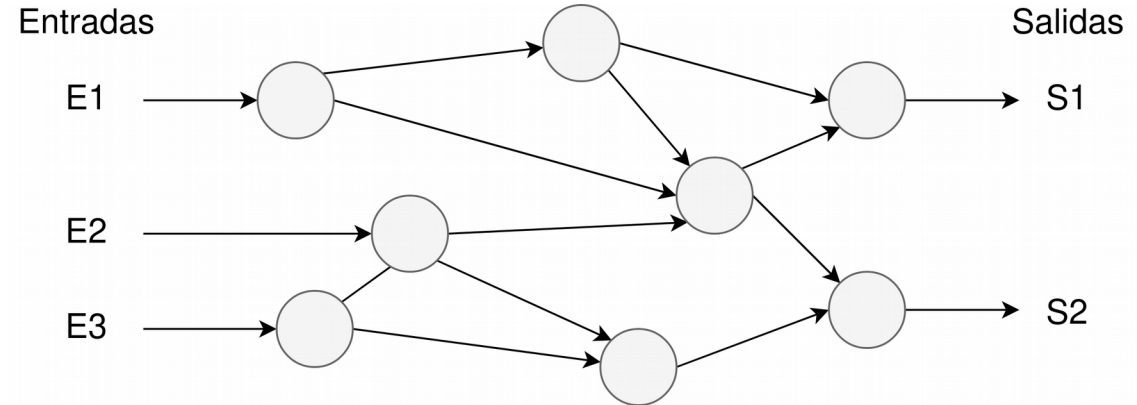


Figura 6: Ejemplo de arquitectura de una red neuronal

2.3 Aprendizaje en una Red Neuronal

En el apartado anterior se ha definido formalmente una red neuronal como un grafo dirigido (donde los nodos equivalen a neuronas y las aristas equivalen a las conexiones entre estas neuronas) que permite aproximar la función que está implícitamente determinada por un conjunto de ejemplos. El objetivo de aproximar está función implícita es, una vez entrenada la red neuronal, realizar predicciones sobre la salida que producirán ejemplos de entrada

independientes de los ejemplos de entrenamiento. En este apartado se expondrá el proceso mediante el cual la red neuronal varía el conjunto de parámetros que la definen para adaptarse a la función que caracteriza estos ejemplos. Es decir, se expondrá en que consiste el proceso de aprendizaje de una red neuronal.

Existen diferentes formas de inicializar los parámetros de una red neuronal. Una de ellas es la inicialización pseudo-aleatoria que se utilizará a continuación por su simplicidad. Suponiendo que la red neuronal está inicializada de forma aleatoria, un ejemplo de entrada producirá una salida que es muy probable que diste mucho del ejemplo de salida esperado. Podemos cuantificar esa diferencia calculando la resta cuadrática entre el valor esperado y el valor obtenido de cada neurona. La suma de esta diferencia en cada neurona tendrá como resultado lo que se definirá como el “coste” de ese ejemplo. Cuanto menor sea el coste, el resultado más se acercará al resultado esperado. A partir de este concepto podemos definir la **función de coste** como la función que a un ejemplo dado le hace corresponder su coste. Por lo tanto, el proceso de aprendizaje consistirá en minimizar esta función de coste haciendo que cada vez las salidas obtenidas se aproximen más a las salidas esperadas.

Para exponer el proceso de minimización de esta función se introducirá un nuevo concepto: el **algoritmo de descenso del gradiente**. Este algoritmo nos permite localizar un mínimo de una función de forma iterativa. En cada iteración el algoritmo parte de un punto de la función y calcula el gradiente: vector que indica la dirección en el cual la función crece más rápidamente y su módulo indica el ritmo. El algoritmo se desplazará hacia la dirección opuesta a este gradiente tanto como indique el ratio de aprendizaje (existen diferentes criterios para definir este parámetro y su elección es determinante en el éxito del proceso de aprendizaje). En este punto se pasa a una siguiente iteración donde se repetirá el proceso hasta que el modulo del vector sea muy próximo a cero, momento en el cual se asumirá que se ha localizado un mínimo de la función.

Aplicado a las redes neuronales este algoritmo indica en cada ejemplo cuanto ha de variar el valor de cada una de las neuronas de salida para minimizar el coste de ese ejemplo. Para variar su salida se varían los pesos asociados a estas neuronas. De esta forma los pesos que definen la red neuronal se irán ajustando en función del coste de cada ejemplo hasta que el coste sea tan pequeño que nos permita realizar una predicción lo suficientemente fiable para la tarea que la red neuronal fue diseñada.

Sin embargo, las redes neuronales actuales son complejas y se dividen en capas: una capa de entrada, unas capas intermedias también llamadas capas ocultas, y una capa de salida. En este tipo de arquitectura las salidas de una neurona solo pueden conectarse con neuronas de la siguiente capa. Esta arquitectura se muestra en la Figura 6. Por lo tanto, necesitamos un mecanismo que nos permita determinar como, además de variar los pesos de la capa de salida, variar los pesos de las capas ocultas con el objetivo de minimizar la función de coste.

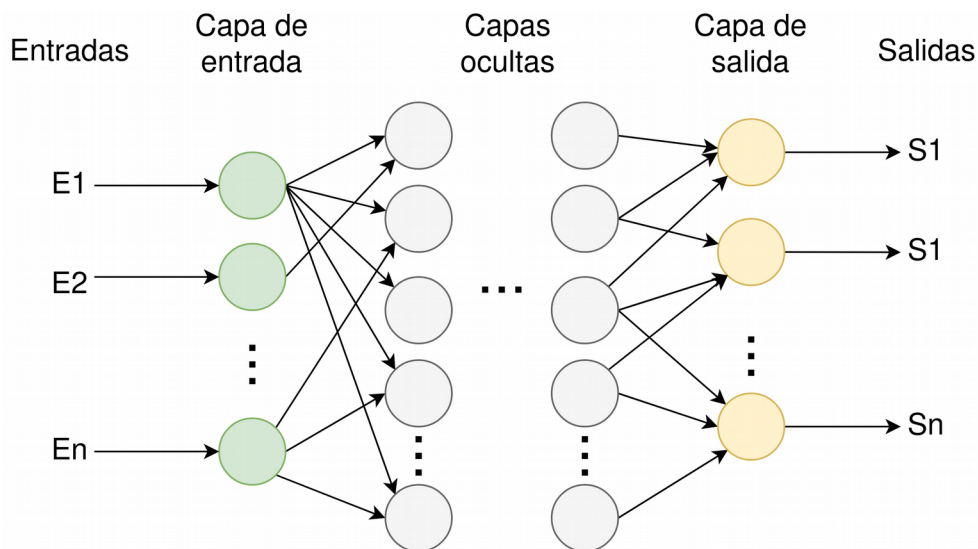


Figura 7: Esquema conceptual de una arquitectura de perceptrón multicapa

El mecanismo que solventa esta tarea es el **algoritmo de retropropagación**. En este algoritmo, en primer lugar, se introduce un ejemplo de entrada, que pasa por las capas intermedias hasta llegar a la capa de salida. Se calcula el coste para ese ejemplo y a través del algoritmo de descenso del gradiente se le indica a la capa de salida como tienen que variar sus pesos para minimizar la función de coste. Cada una de las neuronas que componen la capa de salida indica a las neuronas de la capa anterior su contribución al error total. Las neuronas de esta capa anterior reciben, por lo tanto, la propagación del error de todas las neuronas de la capa de salida a la que están conectadas. Este algoritmo se repite con la siguiente capa hasta llegar a la capa de entrada por lo que se podrá ajustar los pesos de todas las neuronas de la red neuronal para cada uno de los ejemplos de entrenamiento. Una vez el error baja de un determinado umbral fijado previamente se podrá decir que el proceso de aprendizaje de la red neuronal ha concluido y se podrán realizar predicciones.

2.4 Evolución y tipos de redes neuronales

En los apartados anteriores ya se ha expuesto que existen diferentes tipos de arquitecturas de redes neuronales fruto de su evolución y de las características del modelo necesario. En este apartado se analizará con más detalle los diferentes tipos de redes neuronales. Es importante señalar que existe una gran cantidad de arquitecturas y variaciones de estas arquitecturas y no entra dentro del alcance de este trabajo cubrir todas ellas por lo que se incluirán las que se consideraron más relevantes para tener elementos de referencia a la hora de completar la implementación práctica.

- Perceptrón: Es la arquitectura más simple de una red neuronal ya que está compuesta simplemente de una neurona. Se puede ver su representación gráfica en el apartado 2.2 en la Figura 1.

- Perceptrón Multicapa: Arquitectura formada por varias neuronas organizadas en capas. Una capa de entrada, una o más capas intermedias también llamadas capas ocultas y una capa de salida. Cada neurona solo puede conectarse con las neuronas de capa posterior, no puede conectarse ni con las neuronas de una capa anterior ni con neuronas de su misma capa. Esta evolución del perceptrón fue especialmente efectiva para resolver problemas que no son linealmente separables. Se puede ver su representación gráfica en el apartado 2.3 en la Figura 6.
- Red Neuronal Convolutiva: Son un tipo de perceptrón multicapa que introduce variaciones inspiradas en como funciona la corteza visual de un cerebro biológico. Para ello, esta arquitectura introduce dos nuevas características: conexiones locales (agrupaciones de neuronas) y la invarianza espacial (capacidad de abstracción en el reconocimiento de un objeto, se reconoce el objeto aunque varíe en alguna de sus características).

En términos de organización de las neuronas de la red, esta se divide en dos partes. Una primera parte, formada por las capas más próximas a la capa de entrada, que reconoce características de los datos de entrada a través de una estructura de neuronas en la que las neuronas están conectadas sólo con algunas neuronas de la siguiente capa y donde se aplican las operaciones de convolución (que da nombre a este tipo de arquitectura), una función no lineal y operaciones de submuestreo (reduce la dimensionalidad de cada característica) y una segunda parte, formada por las capas más próximas a la capa de salida, que clasifica estos datos de entrada a través de una red donde todas las neuronas están conectadas con todas las neuronas de la siguiente capa. Este tipo de arquitectura funciona muy bien en el reconocimiento y clasificación de imágenes.

- Red Neuronal Recurrente: las arquitecturas definidas anteriormente son arquitecturas de redes neuronales prealimentadas (feed-forward), es decir, las conexiones entre las neuronas no forman ciclos. Sin embargo, las redes neuronales recurrentes no solo tienen como entrada la entrada del ejemplo actual si no que además pueden utilizar estados internos de la entrada previa. Esto permite que sean una arquitectura muy eficaz para modelar series temporales.

2.5 Aplicaciones de las redes neuronales a la criptografía

El aprendizaje automático y en concreto las redes neuronales han sido aplicadas a muchas tareas y áreas del conocimiento como la conducción autónoma, el reconocimiento de voz, la clasificación de imágenes y un largo etc. Este trabajo estudiará el uso de redes neuronales en el criptoanálisis del algoritmo de cifrado AES y en este apartado se introducirá otras aplicaciones relacionadas con la criptografía que han resultado exitosas.

En 1995 Sébastien Dourlens introduce el término *Neuro-Cryptography*[8] refiriéndose a la posibilidad de aplicar las capacidades de las redes neuronales a la criptografía. Ya en ese momento se plantea la posibilidad de realizar un criptoanálisis al algoritmo Vigniere o incluso un algoritmo más complejo, y estándar en aquella época, como el Data Encryption Standard. Posteriormente se han llevado a cabo criptoanálisis a estos algoritmos mediante ataques de texto en claro conocido en los que las redes neuronales se han mostrado un método significativamente más efectivo que otros utilizados hasta la fecha[7].

A partir de ese momento se han desarrollado múltiples aplicaciones de estos sistemas a la criptografía, a continuación se exponen unos cuantos ejemplos:

- En 1998 Clark y Blank introdujeron un nuevo método de cifrado basado en redes neuronales [10].
- En 2001 Hua, Lin y Hwang propusieron un esquema de autenticación remota mediante contraseña para una arquitectura multi-servidor utilizando redes neuronales [11].
- En 2006 Shihab propuso un esquema de criptografía asimétrica basada en redes neuronales [12].
- En 2016 Abadi y Andersen estudiaron el uso de redes neuronales adversarias para el desarrollo de métodos de encriptación y desencriptación [9].

3. Algoritmo de cifrado en bloque Advanced Encryption Standard

3.1. Cifrado en bloque

El cifrado en bloque pertenece a la rama de la criptografía de clave simétrica. Es decir, algoritmos de cifrado donde emisor y receptor comparten la misma clave la cual se utiliza tanto para cifrar como para descifrar los mensajes enviados entre ambos.

Dentro de la criptografía de clave simétrica se diferencia entre la criptografía de flujo y la criptografía en bloque. La principal diferencia entre ambos métodos es que en la criptografía de flujo se cifra bit a bit mientras que en la criptografía en bloque se cifra en grupos de bits de longitud fija llamados bloques.

En términos generales un algoritmo de cifrado de bloque funciona de la siguiente forma: a partir de un bloque de texto en claro de longitud fija y una clave, se ejecutan una serie de operaciones (las cuales dependen de que algoritmo se utilice) y se obtiene un texto cifrado. Estas operaciones deben de permitir recuperar el texto en claro inicial si se posee el texto cifrado y la clave.

3.2 Modos de operación del cifrado en bloque

En el apartado anterior se ha descrito el funcionamiento general de un algoritmo de cifrado en bloque. Sin embargo existen diferentes modos de operación: esquemas que definen relaciones de aplicación entre los diferentes bloques con el fin de garantizar diferentes grados de confidencialidad e integridad en el intercambio de la información cifrada.

- Electronic Code-Book (ECB): Es el modo de operación básico. Se parte el texto en claro a cifrar en bloques de una longitud fija y se cifran todos los bloques con la misma clave. Dos bloques de texto en claro idénticos producirían un bloque de texto cifrado idéntico por lo que es un modo de operación desaconsejable en mensajes largos donde un atacante podría buscar patrones de similitud entre diferentes textos cifrados.
- Cipher-block chaining (CBC): A cada bloque se le aplica una operación XOR con el bloque anterior inmediato de forma que se crea una dependencia de cifrado entre los bloques contiguos. Se utiliza un vector de inicialización para cifrar el primer bloque.
- Cipher feedback (CFB): Este modo de operación emula una unidad de flujo. El vector de inicialización es encriptado y después se le aplica una operación de XOR con el texto en plano. Este proceso se repite con los bloques posteriores.
- Output feedback (OFB): En este modo también se encripta el vector de inicialización el cual opera como una unidad de flujo sincronizada.

3.3 Surgimiento del Advanced Encryption Standard

En 1977 el Departamento de Defensa de Estados Unidos publicó un criptosistema estándar para el uso de datos no clasificados cuyo desarrollo fue llevado a cabo principalmente por IBM y la *National Security Agency* (NSA) llamado *Data Encryption Standard* (DES). Desde ese momento DES se fue ampliamente aceptado y utilizado en todo el mundo en las décadas posteriores.

En la década de los noventa varios ataques contra DES ponen de relieve la necesidad de un sustituto para este algoritmo por lo que, en 1997, el *National Institute of Standards and Technology* (NIST) del gobierno de los Estados Unidos hizo un llamamiento para que se presentaran propuestas para un nuevo estándar de cifrado que se llamaría *Advanced Encryption Standard* (AES). En 1999 el NIST hizo pública la lista de los cinco finalistas los cuales eran: Mars, RC6, Rijndael, Serpent y Twofish. Finalmente, un año más tarde es elegido Rijndael como ganador del concurso y es adoptado como el nuevo *Advanced Encryption Standard* con alguna adaptación.

3.4 Funcionamiento del Advanced Encryption Standard

AES es un criptosistema que cifra bloques de texto en claro de 128 bits con claves de cifrado de 128, 192 o 256 bits de longitud. Las cadenas de bits (tanto texto en claro como claves) se representan en matrices de bytes. Sea una

cadena de 128 bits tal que $\{bit_0, bit_1, bit_2 \dots bit_{127}\}$, se representaría una cadena de bytes tal que $\{B_0, B_1, B_2 \dots B_{15}\}$ donde $B_0\{bit_0, bit_1, bit_2 \dots bit_7\}$, $B_1\{bit_8, bit_9, bit_{10} \dots bit_{15}\}$ etc. Y se representaría de forma matricial siguiendo el orden de bytes de arriba abajo y de izquierda a derecha:

$$\begin{bmatrix} B_0 & B_4 & B_8 & B_{12} \\ B_1 & B_5 & B_9 & B_{13} \\ B_2 & B_6 & B_{10} & B_{14} \\ B_3 & B_7 & B_{11} & B_{15} \end{bmatrix}$$

El algoritmo ejecuta transformaciones (substituciones, permutaciones...) sobre estas matrices. A estas matrices intermedias se las denomina matrices de estado.

A continuación se muestra un esquema del proceso de operaciones que se ejecutan sobre el texto en claro inicial:

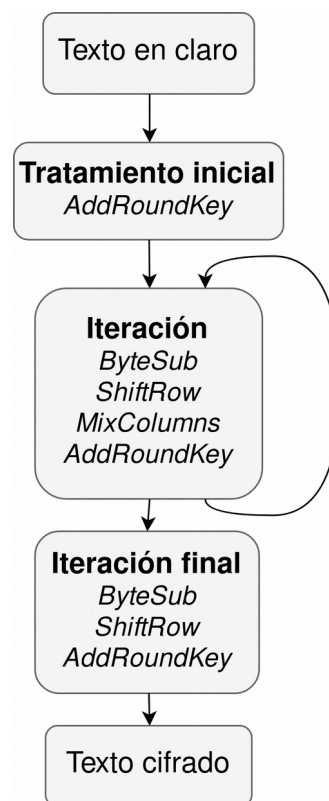


Figura 8: Esquema del funcionamiento general de AES

Como se observa el algoritmo utiliza un método iterativo para obtener el texto cifrado. En cada iteración se utiliza una subclave de cifrado generada a partir de la clave de cifrado principal. Si la clave es de 128 bits el algoritmo constará de 10 iteraciones, si la claves de 192 bits el algoritmo constará de 12 iteraciones y si la clave es de 256 bits el algoritmo constará de 14 iteraciones.

En primer lugar la transformación inicial constará de aplicar la función *AddRoundKey*. Esta función consiste en aplicar una operación XOR entre la matriz estado y la subclave correspondiente. En este caso la matriz estado se corresponde con el texto en claro y la subclave con la subclave inicial.

Posteriormente se aplican las funciones *ByteSub*, *ShiftRow*, *MixColumns* y *AddRoundKey* el número de iteraciones que se corresponden a la longitud de la clave utilizando en cada iteración una subclave distinta. La función *ByteSub* consiste en aplicar una sustitución no lineal de los bytes de la matriz de estado, la función *ShiftRow* aplica una transformación de desplazamiento en las filas de la matriz y la función *MixColumns* mezcla las columnas de la matriz de estado a partir de operaciones polinomiales.

Por último, a la matriz de estado resultante de aplicar las iteraciones pertinentes se vuelve a aplicar las funciones *ByteSub*, *ShiftRow* y *AddRoundKey* dando como resultado la matriz estado que se corresponde con el texto cifrado.

Todas estas funciones tienen una función inversa y si se aplica en orden inverso al aplicado en el proceso de cifrado se obtiene el texto en claro original.

3.5 Criptoanálisis de Advanced Encryption Standard

El criptoanálisis es una rama de la criptografía que estudia las debilidades y los posibles ataques a sistemas criptográficos. Desde su creación AES ha sido puesto a prueba mediante el estudio de diferentes técnicas de criptoanálisis. Muchas de estas técnicas, que no resultaron ser efectivas en la implementación completa del algoritmo, han sido estudiadas en implementaciones reducidas del algoritmo AES con un menor número de rondas sobre las matrices de estado. En este apartado se exponen algunos de estos métodos y su eficacia a la hora de atacar el algoritmo AES.

- Criptoanálisis lineal: Ataque basado en la dependencia lineal entre los bits de entrada y los bits de salida. El método consiste en la utilización de operaciones binarias o-exclusivas sobre el texto en claro y sobre el texto cifrado con el fin de conjeturar mediante una aproximación lineal los bits de la clave. Cuantos más ejemplos de texto en claro / texto cifrado más precisa será esta aproximación y más efectivo será el ataque.
- Criptoanálisis diferencial: Para realizar este criptoanálisis es necesario tener la capacidad de cifrar textos en claro y obtener textos cifrados. El ataque esta basado en la utilización de textos en claro que se diferencian en una variación constante con el fin de encontrar patrones estadísticos en los textos cifrados que permitan conjeturar los bits de la clave.

Estos métodos expuestos previamente eran métodos utilizados en otros algoritmos de cifrado en bloque previos a AES por lo que en la implementación de AES se ha tenido en cuenta la protección contra estos ataques los cuales se vuelven cada vez menos efectivos conforme el número de iteraciones o rondas sobre las matrices estado se incrementa y sus características probabilísticas se difuminan siendo inefectivos en una implementación completa del algoritmo.

Sin embargo, posteriormente a la publicación de AES se introduce un nuevo enfoque en los ataques a los algoritmos de cifrado en bloque que surge de la idea de que las funciones que determinan las transformaciones de las matrices estado pueden ser descritas como un sistema de ecuaciones polinomiales multivariable sobre un cuerpo finito y, por lo tanto, solventar este sistema de ecuaciones permitiría recuperar la clave de cifrado [17]. Los ataques que aprovechan estas propiedades algebraicas de los sistemas de cifrado de bloque se conocen como sistemas de criptoanálisis algebraicos.

- XL y XSL: el método XL (eXtended Linearization)[18] es un algoritmo que permite resolver problemas de ecuaciones multivariables cuadráticas a través de la linealización. Posteriormente, fue extendido al método XSL (eXtended Sparse Linearization) en el cual se exploró la posibilidad de explotar dos propiedades de los sistemas de ecuaciones que se obtienen del criptoanálisis: los sistemas son muy dispersos y están sobredefinidos, conteniendo dependencias lineales entre las ecuaciones que pueden ser simplificadas [19]. No obstante, aunque estas aproximaciones han tenido continuidad en su estudio y apuntan una dirección a seguir en el plano teórico, en el plano práctico ninguna aproximación práctica publicada a obtenido resultados que pongan en peligro la seguridad del algoritmo AES.

4. Implementación práctica de una red neuronal

4.1 Justificación de las tecnologías escogidas para la implementación

Para la implementación de la red neuronal se han utilizado las librerías TensorFlow y Keras. TensorFlow es una librería de código abierto que permite construir modelos de aprendizaje automático representando las computaciones numéricas en forma de grafos donde los nodos del grafo representan las operaciones matemáticas y las aristas del grafo representan las matrices multidimensionales (tensores). Por otra parte, Keras es una librería, también de código abierto, que funciona como API de acceso a otras librerías como TensorFlow, Theano o CNTK construyendo un nivel de abstracción superior y facilitando la construcción de redes neuronales en diferentes entornos y de forma modular.

Se ha elegido la utilización de Keras por su facilidad de uso ya que el objetivo de este trabajo es el estudio de la aplicación de las redes neuronales al criptoanálisis del algoritmo AES y no el desarrollo desde cero de una red neuronal. Y se ha elegido la utilización de TensorFlow como librería complementaria por su extendido uso y su amplia y detallada documentación.

4.2 Diseño y programación de la red neuronal

En el diseño de la red neuronal se han probado diversas arquitecturas con diversos parámetros pero solo se expondrán a continuación las dos arquitecturas que mostraron el comportamiento más óptimo.

Por un lado, se ha implementado una red neuronal perceptrón multicapa. En esta arquitectura los datos, tanto el texto cifrado como el texto en claro, fueron representados como cadenas de 128 bits. La capa de entrada de la red se compone de 128 neuronas que se corresponden a los bits de un bloque del texto cifrado y la capa de salida se compone de 128 neuronas que se corresponden a los bits del texto en claro que dio lugar al bloque de texto cifrado. El diseño que mejor comportamiento ha tenido para las capas ocultas son dos capas de 256 neuronas cada una.

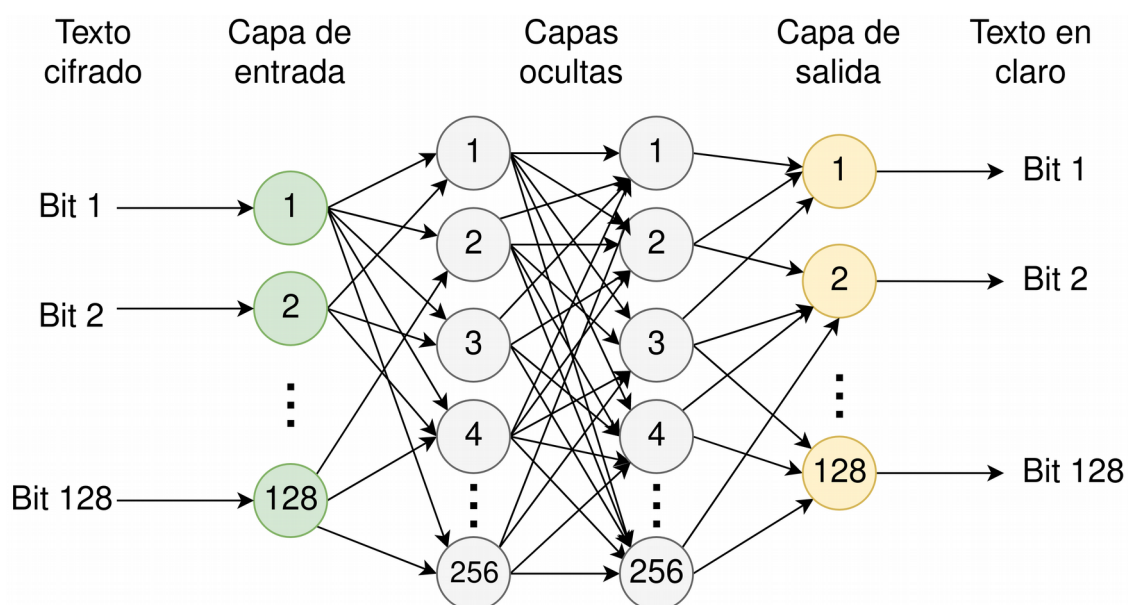


Figura 9: Arquitectura basada en el perceptrón multicapa

Y por otro lado se ha establecido una arquitectura basada en en las capas 3D de las redes neuronales convolucionales pero no se han utilizado otros conceptos de este tipo de redes como las capas de agrupación o las capas de clasificación. Se ha representado cada bloque de cifrado en su estructura matricial como un conjunto de neuronas en 3 dimensiones, una dimensión se corresponde las filas de la matriz, otra con las columnas y la tercera con los bits. Es decir, una estructura de forma 4x4x8. Se han probado varias combinaciones con diferentes números de capas y la que obtuvo mejores resultados fue la arquitectura con ocho capas 3D que se podría corresponder conceptualmente con las matrices estado del algoritmo.

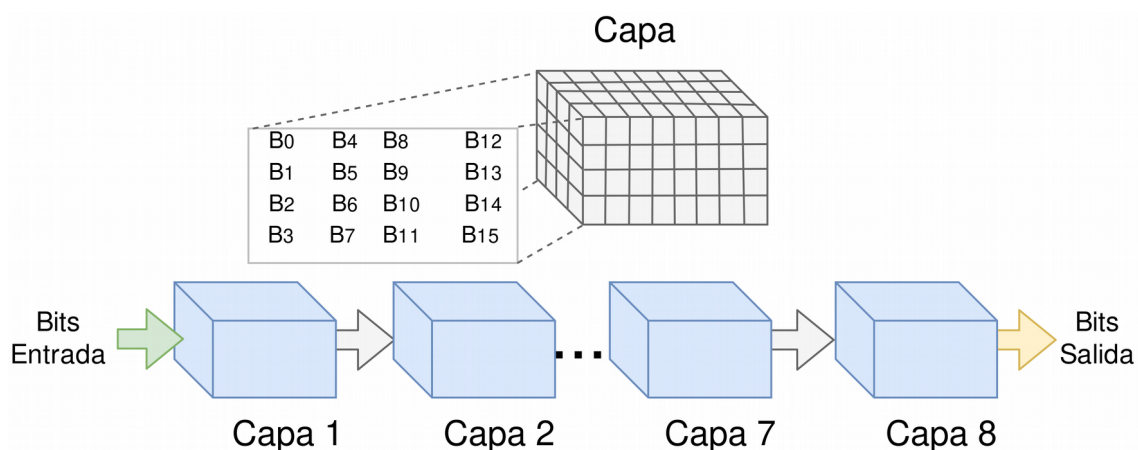


Figura 10: Arquitectura basada en capas de 3 dimensiones

En cuanto a la parametrización de estas arquitecturas se han probado diversas combinaciones y la más exitosa para ambas arquitecturas ha sido la de utilizar como función de coste la media del error cuadrático, como optimizador, el descenso estocástico del gradiente y ninguna función de activación entre las capas intermedias. Los resultados obtenidos como salida de la última capa son valores decimales por lo que se ha definido una función que redondea a valores binarios los valores de salida de la red neuronal. Para ello, en primer lugar se calcula la media del total de los valores y los valores por debajo de la media se harán corresponder con un 0 y los valores por encima de la media se harán corresponder con un 1.

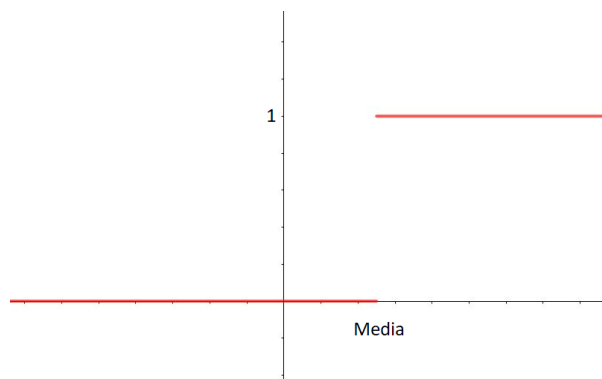


Figura 11: Función que redondea a valores binarios los valores de salida de la red neuronal.

Estas dos implementaciones se pueden consultar en el apartado '9. Anexos', tanto la implementación de la arquitectura de la red neuronal perceptrón multicapa '9.2 Código correspondiente a la red neuronal perceptrón multicapa' como la arquitectura basada en capas en 3 dimensiones en el apartado '9.3 Código correspondiente a la red neuronal basada en capas de tres dimensiones'.

5. Aplicación de la red neuronal al criptoanálisis de AES

5.1 Generación de los datos de entrenamiento y testado de la red neuronal

Para entrenar la red neuronal, en primer lugar es necesario generar los datos con los que se trabajará. En este caso, será necesario generar pares de texto en claro y texto cifrado en dos formatos: pares de cadenas de 128 bits y pares de matrices 4x4 cuyos elementos son cadenas de ocho bits. El primer formato se utilizará para el entrenamiento de la arquitectura de red neuronal perceptrón multicapa y el segundo formato para entrenar la red neuronal basada en capas de tres dimensiones. En ambos casos estos datos se dividen en datos de entrenamiento, que serán utilizados para entrenar la red neuronal y datos de testado, independientes de los datos de entrenamiento, que se utilizan para evaluar la capacidad de predicción de la red neuronal.

Ejemplo de datos de entrada/salida de la red neuronal para la arquitectura perceptrón multicapa:

Formato: $[bit_0, bit_1, bit_2, \dots, bit_{127}]$

Entrada:

[1. 0. 0. 1. 1. 0. 0. 0. 0. 1. 1. 0. 0. 0. 0. 0. 1. 1. 1. 1. 1. 0. 0. 1. 1. 0. 1. 0. 1. 0. 1. 1. 1. 0. 0. 1. 1. 1. 1. 0. 0. 0. 0. 0. 0. 1. 0. 0. 1. 1. 1. 1. 1. 0. 0. 0. 1. 1. 0. 1. 1. 1. 0. 1. 1. 0. 0. 1. 1. 0. 0. 1. 1. 0. 0. 1. 1. 1. 1. 0. 1. 1. 1. 0. 0. 0. 1. 0. 0. 1. 1. 1. 1. 1. 1. 0. 0. 0. 1. 0. 0. 1. 1. 1. 1. 1. 0.]

Salida:

[1. 1. 1. 1. 0. 1. 1. 0. 0. 0. 0. 0. 1. 1. 1. 1. 1. 0. 0. 1. 0. 1. 1. 1. 1. 1. 1. 1. 1. 0. 1. 1. 1. 0. 0. 0. 1. 0. 1. 0. 1. 1. 0. 1. 1. 1. 0. 1. 1. 0. 1. 1. 0. 0. 1. 0. 1. 1. 0. 1. 1. 1. 0. 1. 1. 0. 1. 1. 0. 1. 1. 0. 1. 1. 0. 0. 0. 1. 0. 1. 1. 0. 1. 0. 0. 0. 1. 0. 1. 1. 0. 1. 0. 0. 1. 1. 0. 0. 1. 1. 0. 0. 1. 0. 0. 1. 1. 0. 0. 1. 0. 0. 1. 1. 0. 0. 1. 1. 0. 0. 1. 0. 0. 0. 1.]

Ejemplo de entrada/salida de la red neuronal para la arquitectura basada en capas en tres dimensiones:

Formato:

$[[B_0, B_1, B_2, B_3][B_4, B_5, B_6, B_7][B_8, B_9, B_{10}, B_{11}][B_{12}, B_{13}, B_{14}, B_{15}]]$ donde
 $B_0: [bit_0, bit_1, bit_2, \dots, bit_7]$ y $B_{15}: [bit_{120}, bit_{121}, bit_{122}, \dots, bit_{127}]$

Entrada:

[[[1. 1. 0. 1. 0. 1. 1. 1.]
 [0. 0. 0. 1. 0. 1. 0. 1.]
 [1. 0. 0. 0. 1. 0. 0. 0.]
 [1. 1. 1. 0. 1. 1. 1. 0.]]

[[1. 0. 0. 1. 1. 0. 0. 0.]
 [1. 0. 0. 1. 1. 0. 0. 1.]

[1. 1. 0. 0. 0. 0. 0. 1.]
[0. 1. 1. 1. 1. 0. 0. 0.]]

[[0. 0. 0. 0. 0. 0. 0. 0.]
[1. 0. 1. 1. 1. 1. 0. 0.]
[0. 0. 0. 0. 0. 1. 0. 0.]
[0. 1. 1. 0. 0. 0. 1. 0.]]

[[1. 0. 0. 0. 0. 0. 0. 1.]
[0. 1. 1. 1. 0. 0. 1. 1.]
[1. 0. 1. 1. 0. 0. 1. 1.]
[0. 0. 1. 1. 1. 1. 0. 1.]]]

Salida:

[[[0. 1. 0. 0. 0. 0. 1. 0.]
[0. 1. 1. 1. 0. 1. 0. 0.]
[0. 1. 1. 0. 0. 0. 0. 1.]
[0. 0. 0. 0. 0. 0. 1. 0.]]

[[0. 0. 1. 0. 0. 1. 1. 1.]
[0. 1. 0. 1. 1. 0. 1. 0.]
[0. 1. 0. 0. 1. 1. 1. 1.]
[0. 1. 0. 1. 1. 1. 0. 0.]]

[[1. 1. 0. 0. 0. 0. 0. 1.]
[1. 1. 0. 0. 1. 0. 0. 1.]
[0. 1. 0. 0. 1. 0. 0. 0.]
[0. 0. 1. 1. 1. 0. 0. 0.]]

[[1. 0. 0. 0. 0. 0. 1. 1.]
[0. 1. 1. 0. 0. 0. 1. 1.]
[0. 0. 0. 1. 1. 1. 0. 1.]
[1. 1. 0. 1. 0. 1. 1. 1.]]]

En cuanto al cifrado de los bloques de texto se utilizará una misma clave de 128 bits para todos los bloques generada aleatoriamente en cada prueba y el modo de operación ECB.

El código correspondiente a la generación de datos se puede consultar en el apartado "9.1 Código correspondiente a la generación de datos".

5.2 Entrenamiento y testado de la red neuronal

Durante el entrenamiento y el testado de la red neuronal, se han probado diferentes parámetros y diseños hasta llegar a las dos arquitecturas expuestas anteriormente. Además, al no obtener una mejora substancial en el comportamiento de la red en la implementación del algoritmo completo se ha trabajado con versiones reducidas para observar si se producía alguna mejora. Como ya se ha expuesto en el apartado '3.4 Funcionamiento del Advanced

Encryption Standard', el número de rondas para la versión completa del algoritmo es 10 para las claves de 128 bits, 12 para las claves de 192 bits y 14 para las claves de 256. Sin embargo, las versiones reducidas del algoritmo son versiones donde el número de rondas es menor y su complejidad se reduce. A continuación se analizará la evolución de la función de coste conforme aumentan las iteraciones de entrenamiento en las diferentes implementaciones del algoritmo y en las diferentes arquitecturas propuestas.

Como se ha expuesto anteriormente esta función de coste se corresponde al error medio cuadrático (EMC). Esta función se corresponde a la siguiente expresión:

$$EMC = \frac{\sum_{i=1}^n (\text{valor predicho}_i - \text{valor esperado}_i)^2}{n} .$$

Los mejores resultados se han obtenido en la versión reducida del algoritmo de una ronda. En la siguiente gráfica se puede observar el aprendizaje de la red neuronal a través de la evolución de la función de coste en cada iteración de entrenamiento en la arquitectura de la red neuronal perceptrón multicapa para las versiones reducidas del algoritmo de una ronda, de dos rondas y la implementación completa.

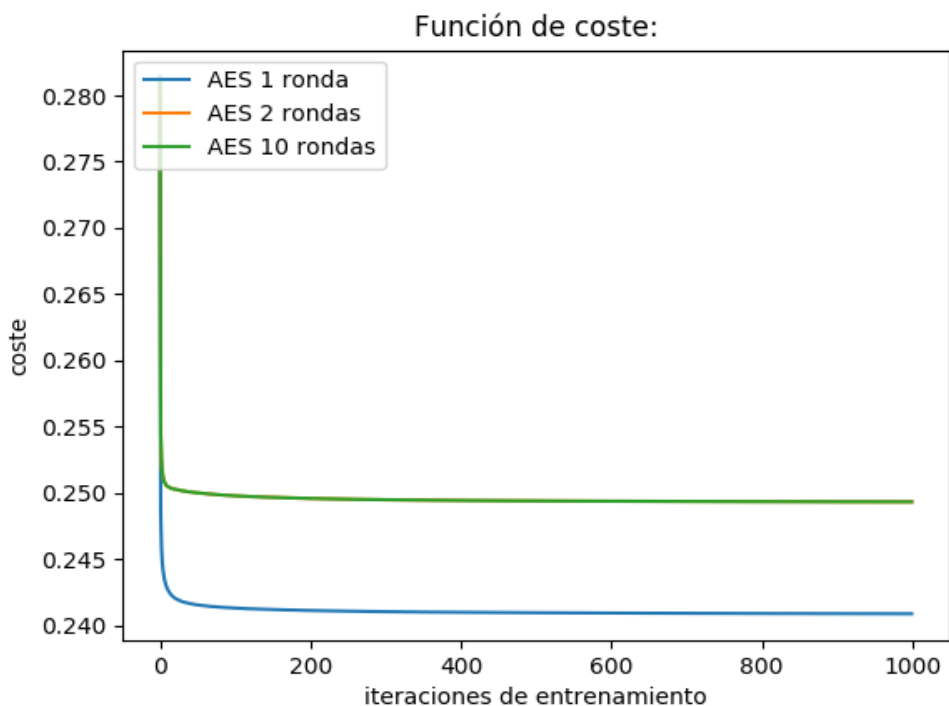


Figura 12: Evolución función de coste perceptrón multicapa

En la gráfica, puede observarse como el proceso de aprendizaje de la red es mucho más eficaz en la versión reducida del algoritmo de una ronda e igualmente ineficaz en la implementación reducida de dos rondas y la implementación completa de 10 rondas, cuyas líneas (solapadas) se sitúan en torno al valor 0.25. Esto se debe a que la red neuronal en lugar de predecir valores cercanos a 1 o a 0, que tras ser normalizados darían lugar a una predicción, no está detectando ningún patrón por lo que los valores de la mayoría de las neuronas de salida son próximos a 0.5 por lo que tras aplicar la fórmula del error cuadrático medio los valores son o $(0.5-1)^2=0.25$ o $(0.5-0)^2=0.25$. Lo cual nos indica que en estos casos la red neuronal no está aprendiendo correctamente.

La siguiente gráfica muestra esta misma evolución de la función de coste, en las mismas condiciones, para la arquitectura basada en capas de tres dimensiones.

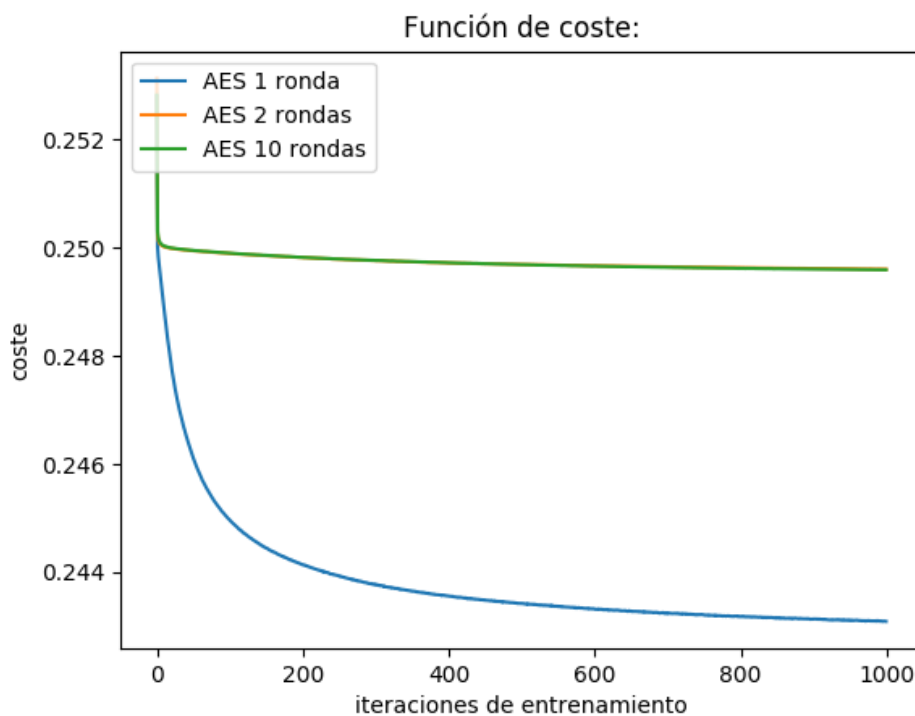


Figura 13: Evolución función de coste red neuronal basada en capas 3D

Se observa que el comportamiento para la implementación reducida de AES de 2 rondas como para la implementación completa de AES de 10 rondas es muy similar en la arquitectura perceptrón multicapa y la arquitectura basada en capas de tres dimensiones donde en ambas arquitecturas se solapan en torno al valor 0.25. Sin embargo, en la implementación reducida de AES de 1 ronda el comportamiento entre las dos arquitecturas difiere teniendo un aprendizaje más rápido la arquitectura perceptrón multicapa.

5.3 Resultados del estudio

Los resultados del estudio fueron obtenidos bajo las siguientes especificaciones de hardware:

Procesador	i5-3337U 3ª generación con Intel Turbo Boost Technology 2.0 clock speed : 1.80 / 2.70 Turbo GHz 3rd level cache : 3 MB
Memoria principal	Estándar: 6,144(2,048 en placa + 4,096)MB Memoria máxima : 6,144 MB Tecnología : DDR3 RAM (1,600 MHz)
Disco duro	Tipo: estado sólido Capacidad : 128 GB interfaz : mSATA

El código de las pruebas realizadas se puede consultar en el apartado '9.4 Código correspondiente a las pruebas de test realizadas'.

En el criptoanálisis de la versión reducida de AES de 1 ronda mediante la aplicación de la red neuronal perceptrón multicapa, se realizan diferentes pruebas cambiando, por un lado, el número de pares de texto claro / texto cifrado disponibles y por otro el número de iteraciones de entrenamiento. Estos parámetros se han elegido observando la evolución de la función de coste en el entrenamiento de la red neuronal ya que, por ejemplo, se observa que a partir de la iteración 300 la función de coste no disminuye por lo que la red no mejorará los resultados.

Nº de pares texto en claro / texto cifrado	Nº de iteraciones de entrenamiento	Tiempo entrenamiento (s)	Precisión¹
2^{13}	50	15.28	55.47%
2^{15}	50	64.19	56.43%
2^{16}	50	179.45	56.95%
2^{13}	100	47.61	55.84%
2^{15}	100	134.98	56.51%
2^{16}	100	273.04	56.82%
2^{13}	300	114.41	55.67%
2^{15}	300	504.39	56.51%
2^{16}	300	1024.45	57.03%

1. Precisión: $\frac{N^\circ \text{ bits correctos}}{N^\circ \text{ bits total}}$

Para el criptoanálisis de la versión reducida de AES de 2 rondas se realiza un solo test ya que apenas se observa evolución en la función de coste por lo que los resultados serán homogéneos.

Nº de pares texto en claro / texto cifrado	Nº de iteraciones de entrenamiento	Tiempo entrenamiento (s)	Precisión
2^{16}	300	785.40	49.80%

Para el criptoanálisis de la versión completa de AES de 10 rondas se vuelve a realizar un solo test por el mismo motivo anteriormente expuesto para la versión reducida de AES de 2 rondas.

Nº de pares texto en claro / texto cifrado	Nº de iteraciones de entrenamiento	Tiempo entrenamiento (s)	Precisión
2^{16}	300	804.24	49.84%

Nº de pares texto en claro / texto cifrado	Nº de iteraciones de entrenamiento	Tiempo entrenamiento (s)	Precisión
2^{13}	100	106.53	52.52%
2^{15}	100	397.18	54.15%
2^{16}	100	815.84	54.89%
2^{13}	300	783.90	54.94%
2^{15}	300	1164.87	55.09%
2^{16}	300	2308.30	55.80%
2^{13}	1000	989.02	54.57%
2^{15}	1000	4043.25	55.48%
2^{16}	1000	7705.12	55.66%

Al igual que en la arquitectura anterior para el criptoanálisis de la versión completa de AES de 2 rondas se vuelve a realizar un solo tes ya que la función de coste no muestra una gran evolución.

Nº de pares texto en claro / texto cifrado	Nº de iteraciones de entrenamiento	Tiempo entrenamiento (s)	Precisión
2^{16}	1000	7619.34	49.79%

Para la versión completa del algoritmo también se realiza un solo test por el mismo motivo anteriormente expuesto

Nº de pares texto en claro / texto cifrado	Nº de iteraciones de entrenamiento	Tiempo entrenamiento (s)	Precisión
2^{16}	1000	7923.27	50.02%

En ambas arquitecturas las neuronas de salida ofrecen valores que posteriormente se redondean a valores binarios. Sin embargo, mediante diversas pruebas se ha observado que los valores más desviados de la media (del total de los valores) ofrecen mejores predicciones que los valores centrales. En la siguiente tabla se muestran los resultados de precisión que obtiene la red neuronal perceptrón multicapa aplicada al criptoanálisis de la implementación reducida de AES de 1 ronda, bajo los mismos parámetros que se muestran en la primera tabla de resultados, pero teniendo en cuenta solo los bits con mejores predicciones.

Para ello se tiene en cuenta una predicción si es menor que la media y menor que la resta de la desviación típica a la media o si es mayor que la media y mayor que la suma de la media y la desviación típica. De esta forma se descartan los valores centrales que son menores que la media pero mayores que la resta de la desviación típica a la media o mayores que la media pero menores la suma de la desviación típica a la media. Por lo tanto, teniendo en cuenta que los valores se distribuyen siguiendo una distribución normal se estaría descartando el 68% de las predicciones que se considera que tienen menos probabilidades de predecir correctamente el valor del bit.

Nº de pares texto en claro / texto cifrado	Nº de iteraciones de entrenamiento	Tiempo entrenamiento (s)	Precisión
2^{13}	50	17.65	60.63%
2^{15}	50	69.53	63.10%
2^{16}	50	144.47	63.70%
2^{13}	100	38.00	61.21%
2^{15}	100	145.15	62.94%
2^{16}	100	290.44	63.55%
2^{13}	300	126.34	61.12%
2^{15}	300	451.63	63.74%
2^{16}	300	1064.12	65.16%

5.4 Conclusiones sobre los resultados

- No se ha conseguido ningún resultado favorable en la implementación completa del algoritmo ni en ninguna versión reducida del algoritmo de más de 1 ronda ya que la precisión se aproxima al 50% que es un resultado esperable de una predicción aleatoria sobre un conjunto de elementos binarios.
- En la versión reducida de AES de 1 ronda la arquitectura que muestra mejores resultados y una capacidad de aprendizaje más rápida con menos recursos es la arquitectura de perceptrón multicapa descrita en el apartado '4.2 Diseño y programación de la red neuronal' .
- Se ha demostrado la mejora en la capacidad de predicción si se selecciona un subconjunto concreto de los resultados lo que permite conjeturar el cifrado de determinados bits con mayor probabilidad de predecir correctamente su valor que otros.
- Los mejores resultados obtenidos para la versión reducida de AES 1 ronda oscilan alrededor de un 57% de precisión teniendo en cuenta todos los bits y un 65% teniendo en cuenta los bits sobre los que se espera obtener mejores predicciones. Por lo que se muestra una mejora en la capacidad de predicción respecto a un sistema aleatorio aunque no es muy significativa.

5.5 Comparación de los resultados obtenidos con otros métodos de criptoanálisis aplicados a AES

Tras la investigación realizada y expuesta en el apartado '3.5 Criptoanálisis de Advanced Encryption Standard' no se han encontrado estudios de sistemas de criptoanálisis aplicados a AES que permitan una comparación cuantitativa con los resultados obtenidos.

6. Conclusiones

- La realización del presente trabajo ha servido para realizar una primera aproximación a las diferentes fases, posibles dificultades y procesos involucrados en un proyecto relacionado con la inteligencia artificial y concretamente con la aplicación de las redes neuronales a un campo determinado como es el criptoanálisis.
- En cuanto al logro de los objetivos planteados se ha conseguido obtener, en líneas generales, los objetivos planteados. No obstante, en un principio se planteó el objetivo de la comparación de los resultados obtenidos con otros métodos de criptoanálisis aplicados a AES y en la investigación posterior sobre estos métodos se comprobó la dificultad de realizar esta comparación por lo que, en este caso, se hubiese necesitado una mejor investigación previa sobre las posibilidades de alcanzar este objetivo.
- Los resultados obtenidos en el estudio no muestran ninguna capacidad de predicción sobre la implementación completa del algoritmo ni en ninguna implementación reducida con un número de rondas mayor a 1 pero estos resultados se corresponden con unas pruebas concretas sobre un reducido número de diseños y arquitecturas de redes neuronales probados lo que no impide que un estudio más exhaustivo con otros diseños en la arquitectura de la red neuronal pueda obtener unos resultados más relevantes en el criptoanálisis de AES.

7. Glosario

- Criptoanálisis: rama de la criptografía que estudia los sistemas criptográficos con el fin de encontrar debilidades en estos sistemas que permitan obtener información sobre la comunicación sin tener todos los datos necesarios para descifrar los mensajes encriptados.
- Ataque de texto en claro conocido: método de criptoanálisis donde el atacante tiene acceso a una serie de ejemplos de textos cifrados con sus correspondientes textos en claro y a partir de esta información obtiene información sobre otros textos cifrados independientes de los anteriores.
- Modo de operación: en un sistema de cifrado de bloque, esquema que define relaciones de aplicación entre los diferentes bloques con el fin de garantizar diferentes grados de confidencialidad e integridad en el intercambio de la información cifrada.
- Función de activación: Función que transforma la salida de un nodo o neurona de una red neuronal para hacerle corresponder un valor en un rango determinado.
- Función de coste: Función que a un ejemplo de entrenamiento de la red neuronal le hace corresponder su coste asociado. Es decir, a un ejemplo dado le hace corresponder un valor que indica cuán lejos está la salida predicha para ese ejemplo de entrada de la salida esperada para este ejemplo.
- Optimizador: Función que dado el coste de un ejemplo indica el módulo, la dirección y el sentido en el que debería variar cada variable para minimizar la función de coste.

8. Bibliografía

1. RUSSELL, S. N., PETER. (2018). *ARTIFICIAL INTELLIGENCE: a modern approach*. [S.I.], PEARSON.
2. M. MICHEL, T. (1997), *Machine Learning*, Redmond,WA, McGraw-Hill Science.
3. PRENEEL, B. (2010), *Understanding Cryptography*, Bochum, Germany, Springer.
4. HAYKIN, S. S. (2016). *Neural networks and learning machines*.
5. GULLI, A., & PAL, S. (2017). *Deep Learning with Keras*. Birmingham, Packt Publishing.
6. DAEMEN, J., RIJMEN, V., (2002), *The design of Rijndael*, Brussels, Belgium, Springer.
7. ALANI M.M. (2012). Neuro-cryptanalysis of des and triple-DES. *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. 7667 LNCS, 637-646.
8. DOURLENS, S., (1997), *Applied Neuro-cryptography*, Université Paris 8
9. ABADI, M., ANDERSEN, D.,(2016) *Learning to protect communication with adversarial neural cryptography*, Google Brain
10. CLARK, M., BLANK, D., (1998), *A Neural-Network Based Cryptographic System*. In: Proceedings of the 9th Midwest Artificial Intelligence and Cognitive Science Conference (MAICS, pp. 91–94)
11. LI, L.-H. (2001). A remote password authentication scheme for multiserver architecture using neural networks. *IEEE Transactions on Neural Networks*. 12, 1498-1504.
12. SHIHAB, K., (2006), *A Backpropagation Neural Network for Computer Network Security*
13. MIRA, J., DELGADO, A.E., BOTICARIO, J.G., Díez F.J., (2003), *Apectos básicos de la inteligencia artificial*, Madrid, Sanz Torres
14. MILLAN D.,B., BOTICARIO J.G., VIÑUELA P.I.,(2006) *Aprendizaje automático*, Madrid, Sanz Torres
15. The asimov institute (2016). Neural Network Zoo. [online] *The neural network zoo*. Disponible en <<http://www.asimovinstitute.org/neural-network-zoo/>> [14/04/2018]

16. KAMINSKY A., RADZISZOWSKI S., & KURDZIEL M. (2010). An overview of cryptanalysis research for the advanced encryption standard. *Proceedings - IEEE Military Communications Conference MILCOM*. 1310-1316.
17. NICOLAS, T., COURTOIS, N., PIEPRZYK, J., (2002) Cryptanalysis of Block Ciphers with Overdefined Systems of Equations, P8 Crypto Lab, SchlumbergerSema, France. 36-38
18. COURTOIS, N., KLIMOV, A., PATARIN, J., & SHAMIR, A. (2000). Efficient Algorithms for Solving Overdefined Systems of Multivariate Polynomial Equations. *Lecture Notes in Computer Science*. 392-407.
19. KIPNIS A., & SHAMIR A. (1999). Cryptanalysis of the HFE public key cryptosystem by relinearization. *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. 1666, 19-30.

9. Anexos

9.1 Código correspondiente a la generación de datos

```
import AES
import random
import numpy as np
BLOCK_SIZE = 16
def __binary_to_unicode(binary_string):
    count = 0
    binary_unicode_char = ""
    unicode_string = ""
    for i in binary_string:
        count = count + 1
        binary_unicode_char = binary_unicode_char + i
        if (count >= 8):
            count = 0
            unicode_char = chr(int(binary_unicode_char, 2))
            unicode_string = unicode_string + unicode_char
            binary_unicode_char = ""
    return unicode_string
def __unicode_to_binary(unicode_string):
    binary_string = ""
    for i in unicode_string:
        binary_string = binary_string + bin(ord(i))[2:].zfill(8)
    return binary_string
def __string_to_npArray(string):
    l = list()
    for c in string:
        if c == "1":
            l.append(1)
        else:
            l.append(0)
    return np.array(l).astype(np.float32)
def __change_dimension(array):
    temp = np.array_split(array, 16)
    temp = np.array(temp).astype(np.float32)
    result = list()
    index = 0
    for a in range(4):
        temp2 = list()
        for b in range(4):
            temp2.append(np.array(temp[index]).astype(np.float32))
            index = index + 1
        result.append(np.array(temp2).astype(np.float32))
    return np.array(result).astype(np.float32)
def __generate_key():
    binary_key = bin(random.getrandbits(128))[2:].zfill(128)
    return __binary_to_unicode(binary_key)
def generateData(num_train_examples, num_test_examples, n_rounds = 10):
    print("Generating data...")
    num_total_examples = num_train_examples + num_test_examples
    x = list()
    y = list()
    key = __generate_key()
    for i in range(num_total_examples):
        binary_plainText = bin(random.getrandbits(128))[2:].zfill(128)
        plainText = __binary_to_unicode(binary_plainText)
        cipherText = AES.Rijndael(key, block_size=BLOCK_SIZE, n_rounds=n_rounds).encrypt(plainText)
        binary_cipherText = __unicode_to_binary(cipherText)
        x.append(__string_to_npArray(binary_cipherText))
        y.append(__string_to_npArray(binary_plainText))
    x = np.array(x).astype(np.float32)
    y = np.array(y).astype(np.float32)
    train_examples_x = x[:num_train_examples]
    test_examples_x = x[num_train_examples:]
    train_examples_y = y[:num_train_examples]
    test_examples_y = y[num_train_examples:]
    print("Data generated.")
    return [train_examples_x, train_examples_y, test_examples_x, test_examples_y]
def generateData3dim(num_train_examples, num_test_examples, n_rounds = 10):
    print("Generating data...")
    num_total_examples = num_train_examples + num_test_examples
    x = list()
    y = list()
    key = __generate_key()
    for i in range(num_total_examples):
        binary_plainText = bin(random.getrandbits(128))[2:].zfill(128)
        plainText = __binary_to_unicode(binary_plainText)
```

```

        cipherText = AES.Rijndael(key, block_size = BLOCK_SIZE, n_rounds=
n_rounds).encrypt(plainText)
        binary_cipherText = __unicode_to_binary(cipherText)
        npArray_plainText = __string_to_npArray(binary_plainText)
        npArray_cipherText = __string_to_npArray(binary_cipherText)
        four_dimension_plainText = __change_dimension(npArray_plainText)
        four_dimension_cipherText = __change_dimension(npArray_cipherText)
        x.append(four_dimension_cipherText)
        y.append(four_dimension_plainText)
    x = np.array(x).astype(np.float32)
    y = np.array(y).astype(np.float32)
    train_examples_x = x[:num_train_examples]
    test_examples_x = x[num_train_examples:]
    train_examples_y = y[:num_train_examples]
    test_examples_y = y[num_train_examples:]
    print("Data generated.")
    return [train_examples_x, train_examples_y, test_examples_x, test_examples_y]

```

9.2 Código correspondiente a la red neuronal perceptrón multicapa

```

from keras.models import Sequential
from keras.layers.core import Dense
from keras.optimizers import SGD
import numpy as np
import time
class MultiPerceptronNeuralNetwork():
    def __init__(self, NB_CLASSES = 128, N_HIDDEN = 0, OPTIMIZER = SGD(lr=0.5, clipvalue=0.5)):
        self.model = Sequential()
        self.model.add(Dense(NB_CLASSES, input_shape=(128,)))
        #if the parameter N_HIDDEN is not 0 then two layer of lenght N_HIDDEN is added to the model
        if N_HIDDEN != 0:
            self.model.add(Dense(N_HIDDEN))
            self.model.add(Dense(N_HIDDEN))
        self.model.add(Dense(NB_CLASSES))
        self.model.summary()
        # compile
        self.model.compile(loss='mean_squared_error', optimizer=OPTIMIZER, metrics=['accuracy'])

    def fit(self, x_train, y_train, NB_EPOCH=300, BATCH_SIZE=128, VERBOSE = 1, VALIDATION_SPLIT=0.2):
        initTime = time.time()
        # training
        history = self.model.fit(x_train, y_train, batch_size=BATCH_SIZE, epochs=NB_EPOCH,
                                verbose=VERBOSE, validation_split=VALIDATION_SPLIT)
        self.training_time = time.time() - initTime
        return history

    def final_round_function(self, prediction):
        #calculation of the mean over all the values predicted
        all_values = list()
        for block in prediction:
            for predicted_value in block:
                all_values.append(predicted_value)
        mean_predicted_values = np.mean(all_values)
        std_predicted_values = np.std(all_values)
        final_prediction = list()
        self.significant_values = list()
        for block in prediction:
            temp_block = list()
            temp2_block = list()
            for predicted_value in block:
                if predicted_value > mean_predicted_values + std_predicted_values:
                    temp2_block.append(1)
                else:
                    temp2_block.append(0)
                if predicted_value >= mean_predicted_values:
                    temp_block.append(1)
                else:
                    temp_block.append(0)
            final_prediction.append(temp_block)
            self.significant_values.append(temp2_block)
        return final_prediction

    def predict(self, x_test):
        prediction = self.model.predict(x_test)
        return prediction

    def print_results(self, true_values, prediction_values, print_only_significant_results = 0):
        correct_values = 0
        incorrect_values = 0

```

```

total_values = len(true_values) * 128
for i in range(len(true_values)):
    for j in range(len(true_values[i])):
        if print_only_significant_results == 0 or self.significant_values[i][j] == 1 :
            if true_values[i][j] == prediction_values[i][j]:
                correct_values += 1
            else:
                incorrect_values += 1
if print_only_significant_results:
    num_significant_values = 0
    for block in self.significant_values:
        for value in block:
            if value:
                num_significant_values +=1
    total_values = num_significant_values
accuracy = (correct_values/total_values)*100
print("RESULTADDS:")
print("-----")
print("Num. Bits: " + str(total_values))
print("Num. Bits correctos: " + str(correct_values))
print("Tiempo entrenamiento (s): " + str(self.training_time))
print("Precision: %.2f"% accuracy + "%")
print("-----")

```

9.3 Código correspondiente a la red neuronal basada en capas de tres dimensiones

```

from keras.models import Model
from keras.layers import Input, Convolution2D
from keras.optimizers import SGD
import time

class NeuralNetwork3Dlayers():
    def __init__(self, num_layers = 8,OPTIMIZER = SGD(lr=0.5, clipvalue=0.5)):
        self.height = 4
        self.width = 4
        self.depth = 8
        conv_depth = 8
        kernel_size = 4
        inp = Input(shape=(self.height, self.width, self.depth))
        conv_layers = list()
        conv_layers.append(Convolution2D(conv_depth,(kernel_size,kernel_size),padding='same')(inp))
        for i in range(num_layers -1):
            conv_layer = Convolution2D(conv_depth, (kernel_size, kernel_size), padding='same')
            (conv_layers[-1])
            conv_layers.append(conv_layer)
        out_conv_layer = conv_layers[-1]
        self.model = Model(inputs=inp, outputs=out_conv_layer)
        self.model.compile(loss='mean_squared_error',
                            optimizer=OPTIMIZER,
                            metrics=['accuracy'])

    def fit(self, x_train, y_train, NB_EPOCH = 100):
        initTime = time.time()
        # training
        history = self.model.fit(x_train, y_train,
                                batch_size=100, epochs=NB_EPOCH,
                                verbose=1, validation_split=0.2)
        self.training_time = time.time() - initTime
        return history

    def final_round_function(self, prediction):
        #calculation of the mean over all the values predicted
        sum = 0
        for matrix in prediction:
            for column in matrix:
                for row in column:
                    for predicted_value in row:
                        sum += predicted_value
        total_values = len(prediction) * self.height * self.width * self.depth
        mean_predicted_value = sum / total_values
        final_prediction = list()
        for matrix in prediction:
            temp_matrix = list()
            for Column in matrix:
                temp_column = list()
                for row in column:
                    temp_row = list()

```



```

        for predicted_value in row:
            if predicted_value >= mean_predicted_value:
                temp_row.append(1)
            else:
                temp_row.append(0)
            temp_column.append(temp_row)
            temp_matrix.append(temp_column)
            final_prediction.append(temp_matrix)
        return final_prediction

def predict(self, x_test):
    prediction = self.model.predict(x_test, verbose=1)
    return prediction

def print_results(self, true_values, prediction_values):
    correct_values = 0
    incorrect_values = 0
    total_values = len(true_values) * self.height * self.width * self.depth
    for index_matrix in range(len(prediction_values)):
        for index_column in range(self.height):
            for index_row in range(self.width):
                for index_block in range(self.depth):
                    if prediction_values[index_matrix][index_column][index_row][index_block] ==
true_values[index_matrix][index_column][index_row][index_block]:
                        correct_values += 1
                    else:
                        incorrect_values += 1
    accuracy = (correct_values / total_values) * 100
    print("RESULTADOS:")
    print("-----")
    print("Num. Bits: " + str(total_values))
    print("Num. Bits correctos: " + str(correct_values))
    print("Tiempo entrenamiento(s): " + str(self.training_time))
    print("Precision: %.2f" % accuracy + "%")
    print("-----")

```

9.4 Código correspondiente a las pruebas de test realizadas

```

import GenerateData as gd
import MultiPerceptronNeuralNetwork as mpNN
import NeuralNetwork3DLayers as NN3D
import matplotlib.pyplot as plt

def test_MultiPerceptronNeuralNetwork(num_train_examples, num_test_examples, num_rounds_aes,
nb_epoch, print_only_significant_results = 0):
    [X_train, Y_train, X_test, Y_test] = gd.generateData(num_train_examples, num_test_examples,
num_rounds_aes)
    neural_network = mpNN.MultiPerceptronNeuralNetwork(N_HIDDEN=256)
    neural_network.fit(X_train, Y_train, NB_EPOCH = nb_epoch)
    prediction = neural_network.predict(X_test)
    final_prediction = neural_network.final_round_function(prediction)
    neural_network.print_results(Y_test, final_prediction, print_only_significant_results =
print_only_significant_results)

def test_NeuralNetwork3DLayers(num_train_examples, num_test_examples, num_rounds_aes, nb_epoch):
    [X_train, Y_train, X_test, Y_test] = gd.generateData3dim(num_train_examples, num_test_examples,
num_rounds_aes)
    neural_network = NN3D.NeuralNetwork3DLayers()
    neural_network.fit(X_train, Y_train, NB_EPOCH=nb_epoch)
    prediction = neural_network.predict(X_test)
    final_prediction = neural_network.final_round_function(prediction)
    neural_network.print_results(Y_test, final_prediction)

def test_LossFunctionComparisonMPNN(num_train_examples, num_test_examples):
    [X_train_1, Y_train_1, X_test_1, Y_test_1] = gd.generateData3dim(num_train_examples,
num_test_examples, n_rounds =1)
    [X_train_2, Y_train_2, X_test_2, Y_test_2] = gd.generateData3dim(num_train_examples,
num_test_examples, n_rounds =2)
    [X_train_10, Y_train_10, X_test_10, Y_test_10] = gd.generateData3dim(num_train_examples,
num_test_examples, n_rounds =10)
    neural_network1 = mpNN.MultiPerceptronNeuralNetwork()
    history1 = neural_network1.fit(X_train_1, Y_train_1)
    neural_network2 = mpNN.MultiPerceptronNeuralNetwork()
    history2 = neural_network2.fit(X_train_2, Y_train_2)
    neural_network10 = mpNN.MultiPerceptronNeuralNetwork()
    history10 = neural_network10.fit(X_train_10, Y_train_10)
    plt.plot(history1.history['loss'])
    plt.plot(history2.history['loss'])
    plt.plot(history10.history['loss'])

```

```

plt.title('Función de coste:')
plt.ylabel('coste')
plt.xlabel('iteraciones de entrenamiento')
plt.legend(['AES 1 ronda', 'AES 2 rondas', 'AES 10 rondas'], loc='upper left')
plt.show()
prediction1 = neural_network1.predict(X_test_1)
final_prediction1 = neural_network1.final_round_function(prediction1)
print("AES 1 RONDA")
neural_network1.print_results(Y_test_1, final_prediction1)
prediction2 = neural_network2.predict(X_test_2)
final_prediction2 = neural_network2.final_round_function(prediction2)
print("AES 2 RONDAS")
neural_network2.print_results(Y_test_2, final_prediction2)
prediction10 = neural_network10.predict(X_test_10)
final_prediction10 = neural_network10.final_round_function(prediction10)
print("AES 10 RONDAS")
neural_network10.print_results(Y_test_10, final_prediction10)

def test_LossFunctionComparison3DNN(num_train_examples, num_test_examples):
    [X_train_1, Y_train_1, X_test_1, Y_test_1] = gd.generateData(num_train_examples,
num_test_examples, n_rounds =1)
    [X_train_2, Y_train_2, X_test_2, Y_test_2] = gd.generateData(num_train_examples,
num_test_examples, n_rounds =2)
    [X_train_10, Y_train_10, X_test_10, Y_test_10] = gd.generateData(num_train_examples,
num_test_examples, n_rounds =10)
    neural_network1 = NN3D.NeuralNetwork3DLayers()
    history1 = neural_network1.fit(X_train_1, Y_train_1)
    neural_network2 = NN3D.NeuralNetwork3DLayers()
    history2 = neural_network2.fit(X_train_2, Y_train_2)
    neural_network10 = NN3D.NeuralNetwork3DLayers()
    history10 = neural_network10.fit(X_train_10, Y_train_10)
    plt.plot(history1.history['loss'])
    plt.plot(history2.history['loss'])
    plt.plot(history10.history['loss'])
    plt.title('Función de coste:')
    plt.ylabel('coste')
    plt.xlabel('iteraciones de entrenamiento')
    plt.legend(['AES 1 ronda', 'AES 2 rondas', 'AES 10 rondas'], loc='upper left')
    plt.show()
    prediction1 = neural_network1.predict(X_test_1)
    final_prediction1 = neural_network1.final_round_function(prediction1)
    print("AES 1 RONDA")
    neural_network1.print_results(Y_test_1, final_prediction1)
    prediction2 = neural_network2.predict(X_test_2)
    final_prediction2 = neural_network2.final_round_function(prediction2)
    print("AES 2 RONDAS")
    neural_network2.print_results(Y_test_2, final_prediction2)
    prediction10 = neural_network10.predict(X_test_10)
    final_prediction10 = neural_network10.final_round_function(prediction10)
    print("AES 10 RONDAS")
    neural_network10.print_results(Y_test_10, final_prediction10)

#TEST AES 1 round multiperceptron neuronal network
test_MultiPerceptronNeuralNetwork(num_train_examples=8192,num_test_examples=1000, num_rounds_aes=1,
nb_epoch=50)
#test_MultiPerceptronNeuralNetwork(num_train_examples=32768,num_test_examples=1000,
num_rounds_aes=1, nb_epoch=50)
#test_MultiPerceptronNeuralNetwork(num_train_examples=65536,num_test_examples=1000,
num_rounds_aes=1, nb_epoch=50)
#test_MultiPerceptronNeuralNetwork(num_train_examples=8192,num_test_examples=1000,
num_rounds_aes=1, nb_epoch=100)
#test_MultiPerceptronNeuralNetwork(num_train_examples=32768,num_test_examples=1000,
num_rounds_aes=1, nb_epoch=100)
#test_MultiPerceptronNeuralNetwork(num_train_examples=65536,num_test_examples=1000,
num_rounds_aes=1, nb_epoch=100)
#test_MultiPerceptronNeuralNetwork(num_train_examples=8192,num_test_examples=1000,
num_rounds_aes=1, nb_epoch=300)
#test_MultiPerceptronNeuralNetwork(num_train_examples=32768,num_test_examples=1000,
num_rounds_aes=1, nb_epoch=300)
#test_MultiPerceptronNeuralNetwork(num_train_examples=65536,num_test_examples=1000,
num_rounds_aes=1, nb_epoch=300)
#TEST AES 2 round multiperceptron neuronal network
#test_MultiPerceptronNeuralNetwork(num_train_examples=65536,num_test_examples=1000,
num_rounds_aes=2, nb_epoch=300)
#TEST AES 10 round multiperceptron neuronal network
#test_MultiPerceptronNeuralNetwork(num_train_examples=65536,num_test_examples=1000,
num_rounds_aes=10, nb_epoch=300)
#TEST AES 1 round 3D layers neuronal network
#test_NeuralNetwork3DLayers(num_train_examples=8192,num_test_examples=1000,num_rounds_aes=1,
nb_epoch=100)
#test_NeuralNetwork3DLayers(num_train_examples=32768,num_test_examples=1000,num_rounds_aes=1,
nb_epoch=100)
#test_NeuralNetwork3DLayers(num_train_examples=65536,num_test_examples=1000,num_rounds_aes=1,
nb_epoch=100)

```

```
#test_NeuralNetwork3DLayers(num_train_examples=8192,num_test_examples=1000,num_rounds_aes=1,
nb_epoch=300)
#test_NeuralNetwork3DLayers(num_train_examples=32768,num_test_examples=1000,num_rounds_aes=1,
nb_epoch=300)
#test_NeuralNetwork3DLayers(num_train_examples=65536,num_test_examples=1000,num_rounds_aes=1,
nb_epoch=300)
#test_NeuralNetwork3DLayers(num_train_examples=8192,num_test_examples=1000,num_rounds_aes=1,
nb_epoch=1000)
#test_NeuralNetwork3DLayers(num_train_examples=32768,num_test_examples=1000,num_rounds_aes=1,
nb_epoch=1000)
#test_NeuralNetwork3DLayers(num_train_examples=65536,num_test_examples=1000,num_rounds_aes=1,
nb_epoch=1000)
#TEST AES 2 round 3D layers neuronal network
#test_NeuralNetwork3DLayers(num_train_examples=65536,num_test_examples=1000,num_rounds_aes=2,
nb_epoch=1000)
#TEST AES 10 round 3D layers neuronal network
#test_NeuralNetwork3DLayers(num_train_examples=65536,num_test_examples=1000,num_rounds_aes=10,
nb_epoch=1000)
#TEST AES 1 round multiperceptron neuronal network only significat predictions
#test_MultiPerceptronNeuralNetwork(num_train_examples=8192,num_test_examples=1000,
num_rounds_aes=1, nb_epoch=50,print_only_significant_results = 1)
#test_MultiPerceptronNeuralNetwork(num_train_examples=32768,num_test_examples=1000,
num_rounds_aes=1, nb_epoch=50,print_only_significant_results = 1)
#test_MultiPerceptronNeuralNetwork(num_train_examples=65536,num_test_examples=1000,
num_rounds_aes=1, nb_epoch=50,print_only_significant_results = 1)
#test_MultiPerceptronNeuralNetwork(num_train_examples=8192,num_test_examples=1000,
num_rounds_aes=1, nb_epoch=100,print_only_significant_results = 1)
#test_MultiPerceptronNeuralNetwork(num_train_examples=32768,num_test_examples=1000,
num_rounds_aes=1, nb_epoch=100,print_only_significant_results = 1)
#test_MultiPerceptronNeuralNetwork(num_train_examples=65536,num_test_examples=1000,
num_rounds_aes=1, nb_epoch=100,print_only_significant_results = 1)
#test_MultiPerceptronNeuralNetwork(num_train_examples=8192,num_test_examples=1000,
num_rounds_aes=1, nb_epoch=300,print_only_significant_results = 1)
#test_MultiPerceptronNeuralNetwork(num_train_examples=32768,num_test_examples=1000,
num_rounds_aes=1, nb_epoch=300,print_only_significant_results = 1)
#test_MultiPerceptronNeuralNetwork(num_train_examples=65536,num_test_examples=1000,
num_rounds_aes=1, nb_epoch=300,print_only_significant_results = 1)
```