Universitat Autònoma de Barcelona

Trabajo de Fin de Grado

Grado en Ingeniería Electrónica de Telecomunicaciones

Red neuronal clasificadora de trazas RTN para la generación de números aleatorios

Manel Marin Plaza

Director: Javier Martín Martínez

Departamento de Ingeniería Electrónica

Universidad Autónoma de Barcelona

Bellaterra, Julio de 2021

ÍNDICE

ABSTRACT

En este trabajo se describe la metodología necesaria para el diseño y desarrollo de una Red Neuronal (a partir de ahora RN) capaz de clasificar trazas de Random Telegraph Noise (a partir de ahora RTN) según la incertidumbre en la repetibilidad de símbolos binarios consecutivos. En el proyecto se engloban todos los aspectos necesarios para su obtención y evaluación, desde la conceptualización hasta los tests de dicha RN.

El uso de estas trazas RTN está pensado para aplicaciones de encriptación y ciberseguridad debido a su capacidad para no seguir patrones descifrables.

Para la programación previa de la RN y su desarrollo posterior se ha usado el software MatLab, así como la 'toolbox' de Deep Learning, junto con una base de muestras de trazas RTN obtenidas en el laboratorio.

1. INTRODUCCIÓN

En este primer apartado introductorio se describirán los fundamentos básicos de los dos conceptos principales del trabajo: las RN y las trazas RTN, junto con toda la metodología que se desarrollará a lo largo del proyecto.

Asimismo, se mencionará de forma breve los objetivos a cumplir en referencia a la RN obtenida al finalizar el proyecto.

1.1. Introducción a las Redes Neuronales

Las Redes Neuronales son un modelo computacional matemático que describe de forma estructural el comportamiento biológico que tendría un sistema neuronal, es decir, se hace una réplica matemática de un cerebro *[1]*. Hay un número muy grande de tipologías y configuraciones en el campo de las RN, así como métodos de entrenamiento, pero para este trabajo se ha usado una Red Neuronal Multicapa con aprendizaje supervisado, una de las configuraciones más sencillas con las que trabajar. Este tipo de aprendizaje para la RN requiere de entradas junto con sus correspondientes salidas para entrenar la RN.

Estas redes se conforman a partir de unidades básicas complejamente interconectadas llamadas perceptrones, neuronas, nodos, etc., las cuales son sistemas fundamentales que aplican una alteración sistemática a la señal que reciben en su entrada y la envían a su salida.
Dichas interconexiones entre perceptrones vienen ponderadas por distintos valores llamados pesos, los cuales pueden ser variables, cosa que le da flexibilidad al sistema completo para realizar una gama de tareas distintas, así como la capacidad de aprender de nuevas entrenando la RN con dicho propósito.

Las RN multicapa se componen de tres capas diferenciadas: la capa de entrada, la capa oculta y la capa de salida.
La capa de entrada recoge las señales o estímulos introducidos en la RN y las envía a la capa oculta. En ella se realizarán todas las alteraciones matemáticas a las señales introducidas a causa de la interacción de los perceptrones y sus pesos con la señal recibida, los cuales están distribuidos por múltiples subcapas interconectadas; para esta RN los perceptrones de la capa oculta provocan una alteración de tipo sigmoidal a la señal que se les introduce, por tanto, la capa oculta está compuesta por perceptrones sigmoidales. Finalmente, la señal resultante de la capa oculta llegara a la capa de salida en las que unos perceptrones lineales se encargan de transmitir la información a la salida de la RN

En la Figura 1 se muestra un esquema abstracto de un grupo de perceptrones interconectados que componen una RN.

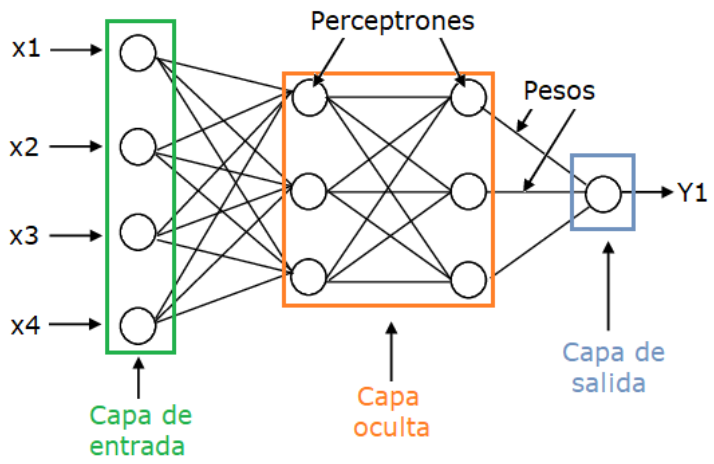

Figura 1: Esquema conceptual de una Red Neuronal Multicapa.

Como se puede observar, se distinguen las tres capas de forma diferenciada y la composición de perceptrones interconectados creando una agrupación compleja de conexiones ponderadas por pesos que en conjunto resultan en una RN.

1.2. Presentación de las trazas RTN

Las trazas RTN son pequeñas alteraciones en la señal de salida de ciertos dispositivos electrónicos caracterizados con tecnología de semiconductores, principalmente transistores MOSFET.
A causa de la miniaturización de estos transistores se ha designado una tipología de estos dispositivos llamados transistores de canal corto. Una de las consecuencias directas de dicha tipología es que el canal por el que circulan los portadores es muy corto y estrecho *[2]* por lo que el número de estos portadores es muy reducido y la contribución de cada uno de ellos tiene suficiente impacto en la señal de salida como para detectar diferencias en la señal resultante por la variación del flujo de portadores.

Durante el proceso de creación de estos transistores se generan pequeñas imperfecciones nanométricas, los cuales pueden atrapar o liberar portadores produciendo las alteraciones en la señal de salida.

En la Figura 2 se muestra el aspecto de las alteraciones provocadas por RTN a la salida de un dispositivo al que se le aplica una señal de entrada constante.

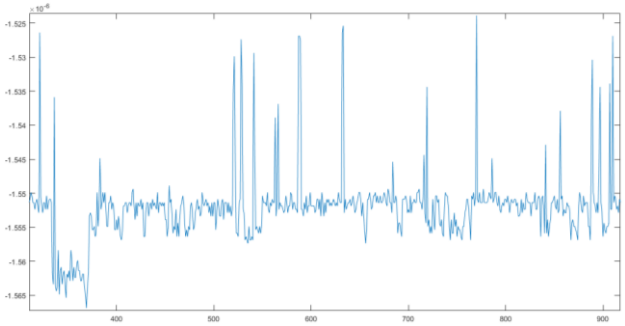

Figura 2: Posible aspecto de una traza RTN.

1.3. Objetivo de la Red Neuronal

Como se ha comentado anteriormente, la RN resultante al final del proyecto debe tener la capacidad de discernir, con un alto ratio de acierto, trazas RTN con un alto índice de incertidumbre en la repetibilidad de símbolos binarios consecutivos, por tanto, aleatorios.

Para entender las características que se describen en estas trazas RTN de interés hay que procesar y digitalizar cada punto de la traza RTN en símbolos binarios 1 y 0, según si dicho valor está por encima o por debajo del valor medio, respectivamente. Una vez digitalizadas, las mejores trazas, es decir, las trazas a identificar, serán en las que al obtener un símbolo determinado haya un 50% de posibilidades de que se repita, la mitad de las veces se repetirá el símbolo binario y la otra mitad no, estadísticamente hablando.

Dichas trazas tienen propiedades muy interesantes por ejemplo en el campo de la encriptación de datos y demás aspectos relacionados con la ciberseguridad, ya que no se les puede asociar un patrón descifrable, de ahí que su identificación sea una herramienta deseada para diversos sectores.

1.4. Descripción general del proyecto

En este apartado se presentan e introducen las fases en las que se ha desarrollado el proyecto. No se darán detalles técnicos, solo una descripción general de las características y el propósito de cada una de las etapas, las cuales sí que se describirán de forma técnica y exhaustiva en futuros apartados.

Éstas van distribuidas en tres secciones principales: El desarrollo de un simulador de trazas RTN, la creación de un criterio de aceptación o rechazo y el diseño y entrenamiento de una red neuronal. Una vez finalizadas estas fases se procederá al test de la RN y el análisis de los resultados.

1.4.1. Diagrama de las etapas

A continuación, en la Figura 3 se muestra el diagrama general de las fases que se ha seguido en el desarrollo en este proyecto a modo de introducción y presentación de cada sección.

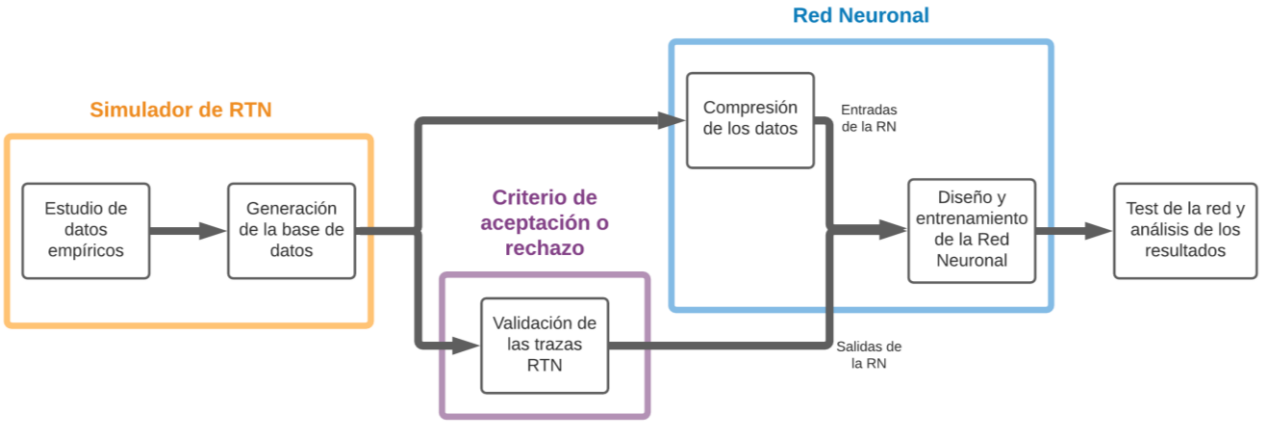

Figura 3: Diagrama general de las etapas del proyecto.

1.4.2. Descripción resumida de las etapas

1.4.2.1. Simulador de RTN

- Estudio de datos empíricos:

El primer paso a realizar es la recopilación de datos referentes a las de las trazas RTN con las que se va a trabajar. Las trazas RTN se pueden parametrizar según unas ciertas características intrínsecas, las cuales cada una de ellas irá determinada por un cierto valor numérico. Dichos valores atienden a distribuciones estadísticas concretas para cada uno de los parámetros, los cuales son los que se desea conocer a la hora de estudiar datos empíricos que describan dichas trazas RTN.
Con este propósito se procederá, mediante diversas funciones y procesos matemáticos, a la extrapolación de los valores característicos de una base de trazas RTN obtenidas en un laboratorio con transistores MOSFET reales.

- Generación de la base de datos:

Una vez obtenidos los datos relativos a las trazas RTN hay que crear un simulador capaz de replicar dichas trazas, dado que se necesita un número muy elevado de muestras para entrenar correctamente una RN y no se tienen suficientes datos experimentales para este proyecto.
A partir de esta base de trazas RTN simuladas se extrapolará el conjunto de entradas y salidas que se irán introduciendo en la red para su entrenamiento.

El simulador de trazas RTN usará el conjunto de datos experimentales obtenidos en el laboratorio para reproducir el comportamiento que tendría una traza de naturaleza similar a las empíricas, por tanto, simula el efecto que producirían un conjunto transistores parecidos a los dispositivos reales del laboratorio.

1.4.2.2. Criterio de Aceptación o Rechazo

- Validación de las trazas RTN:

Para aplicar un entreno supervisado hay que darle a RN, en su fase de entrenamiento, el resultado objetivo que se debería obtener a la salida como consecuencia a una entrada determinada. En este caso la entrada sería una representación matemática de una traza RTN y la salida un símbolo binario (1 o 0) que indique si la traza es válida o no para el propósito perseguido.

Para determinar entonces qué trazas son buenas hay que crear una serie de criterios matemáticos que aseguren, dentro de un cierto rango, que la traza cumple con ciertas especificaciones. Comprobando estas especificaciones se procederá a catalogar la traza como apta o no apta.
Aplicando este proceso de validación a la base de trazas generada podemos obtener un conjunto de datos que representarán las salidas de la RN en su fase de entrenamiento.

1.4.2.3. Red Neuronal

- Compresión de los datos:

Teniendo en cuenta que las RN deben estar dimensionadas de cierto modo coherente y que no es una opción viable introducir las trazas directamente, hay que crear un sistema de compresión de datos en el que se contemple toda la información necesaria para la RN cuando trate de caracterizar una traza RTN concreta.
Hay diversas razones por las que no se puede crear una RN funcional introduciendo los datos sin procesar, la más destacable sería el hecho de que solo se podrían introducir trazas de una determinada longitud, dado que el tamaño de las capas de la RN es un valor fijo una vez creada la red; pero hay otras razones como los costes computacionales muy elevados al augmentar mucho el número de perceptrones en las capas, la imposición de trabajar con bases de datos de tamaños muy elevados para obtener trazas con mucha información, etc.
En vez de eso se condensa la información en una nueva distribución de datos que representen de manera efectiva las características de cada traza, sin perder parte de dichos datos o en su defecto perdiendo el menor posible para que siga siendo una representación adecuada.

Si se aplica esta compresión a la base de trazas RTN obtenemos el conjunto de datos representativos de dicha base con los cuales se entrenará la red, es decir las entradas que usará la RN en su etapa de entrenamiento.

- Diseño y entrenamiento de la RN:

Para obtener una RN equilibrada y coherente con su propósito hay que hacer un buen diseño y entrenamiento de dicha red.

El diseño de la RN es, principalmente, el correcto dimensionamiento de sus capas, por tanto, el tamaño (en número de perceptrones) que tendrá cada capa. Este dimensionamiento es fruto de estudio de muchos expertos en este campo y no atiende a una función matemática específica, sino que se dan un seguido de reglas con las que te aproximas a resultados balanceados, pero dentro de estas reglas hay que hacer diversos ensayos con distintos parámetros para tratar de optimizar la RN que se desea obtener.

En referencia a su entrenamiento, hay diversos modos de entrenar una red dependiendo de las preferencias que se tengan para dicha RN. Acorde con nuestras especificaciones se determinará cuál de las tipologías será la más acertada.

1.4.2.4. Test y análisis de la RN

Una vez obtenida una RN adecuadamente entrenada se procederá a realizar una serie de pruebas que determinen la eficiencia de dicha red. A partir de estos datos se extrapolarán diversas conclusiones con las se podrá seguir optimizando y refinando la metodología de generación de RN.
Teniendo finalmente datos de la eficiencia de cada una de las RN generadas se hará un análisis para determinar si se ha obtenido una red que cumpla con los objetivos establecidos previamente.

2. SIMULADOR DE RTN

Tal y como se ha comentado anteriormente, el desarrollo de una RN que realice una actividad compleja requiere de una base de datos muy grande con la que entrenarla, esto implica el desarrollo de un programa que sea capaz de generar trazas RTN de características similares a las que se obtendrían experimentalmente.

Para obtener las RN deseadas, en este trabajo se ha usado una base de datos con 20.000 muestras las cuales están formadas por 25.000 elementos cada una, es decir, 20.000 trazas RTN de 25.000 puntos.

2.1. Análisis detallado de las trazas RTN

2.1.1. Origen físico

Las trazas RTN son alteraciones de la señal resultante provocadas desintencionadamente en los transistores MOSFET de canal corto y canal estrecho.
Estos transistores tienen particularidades a nivel técnico respecto a los MOSFET de tamaño mayor para las cuales existen diversos modelos matemáticos y computacionales que prevén su comportamiento, pero también se dan ciertas implicaciones físicas que caracterizan efectos propios de estos dispositivos, una de ellas resulta en las trazas RTN.

Los progresos en la miniaturización de transistores han dado lugar a MOSFETs con canales con longitudes de pocas unidades de nanómetros, tanto de ancho, como de largo, lo que genera flujos de corriente en dichos canales con densidades de portadores muy bajas cuando están en funcionamiento. El hecho de tener muy pocos portadores en un canal significa que cada uno de los portadores contribuye de forma tangible a la señal de corriente que se da en la salida del MOSFET y el hecho de que un portador se "perdiese" en el transcurso desde el drenador hasta la fuente del transistor provocaría un cambio apreciable dicha señal.

Para la fabricación de estos transistores MOSFET miniaturizados se emplean múltiples procesos fisicoquímicos cada vez más sofisticados y complejos que dan como resultado geometrías muy exactas. Una de las cualidades más importantes en su fabricación es la capacidad de obtener superficies muy lisas superpuestas dando lugar a tecnologías planares muy precisas, pero no totalmente planas a escala molecular. A este nivel se siguen dando irregularidades nanométricas que forman pequeños huecos en el canal, en la interfície entre el óxido y el semiconductor. Estos huecos son capaces de retener un portador en su transcurso por el canal, a estos huecos se les llama defectos *[3]*.

Sumando estas condiciones tenemos transistores en los que cada portador en el canal es relevante en la señal de salida, junto con irregularidades en los canales capaces de retener portadores o por el contrario liberar portadores previamente retenidos. Es fácilmente deducible que, a la salida del dispositivo, aún y haber aplicado a la entrada una señal de corriente continua, se observarán variaciones en dicho corriente a causa de estos fenómenos. A dichas variaciones

son a las que llamamos trazas RTN. Por su naturaleza, estas trazas podrán adoptar un conjunto limitado de valores posibles dependiendo del estado de cada uno de sus defectos.

2.1.2. Parámetros de caracterización

Las trazas RTN tienen ciertas características físicas intrínsecas que nos permiten poder parametrizarlas y obtener diversos datos útiles para su estudio y manipulación. En este caso dichos parámetros servirán como base para la simulación de nuevas trazas RTN acordes a los datos reales.
Haciendo un estudio macroscópico de las trazas RTN se puede demostrar que cada parámetro viene determinado por una distribución estadística particular que regirá los posibles valores que adopte dicho parámetro.

A continuación se verán en profundidad cada uno de estos parámetros, su origen y contribución en la traza.

2.1.2.1. Offset

Este parámetro no es algo directamente intrínseco a las trazas RTN, pero sí que es un parámetro fundamental a la hora de estudiarlas y simularlas.

El offset es el valor base continuo en una señal, sobre la que se hallan el resto de los componentes de dicha señal. En este caso, el offset de una traza RTN sería el valor que se obtiene cuando el flujo de portadores es totalmente constante, es decir, cuando ningún portador queda atrapado o se libera de un defecto, o visto de otra manera, la señal de salida en un dispositivo con un canal perfecto sin defectos.

Dependiendo de la tipología de transistor MOSFET con el que se trabaje, nMOS o pMOS, los valores de salida serán negativos o positivos, dado que sus portadores mayoritarios son electrones o huecos. Este hecho hace que para determinar el offset en una traza RTN haya que buscar el valor mayor o menor dependiendo del tipo de transistor, pero en ambos casos, el offset siempre será el valor de la traza más cercano a cero siempre que se aplique un offset del mismo signo que la variación que introduzcan los portadores mayoritarios.

Si se obtiene numéricamente este parámetro en un amplio número de dispositivos se puede determinar que éste se distribuye a través de una LogNormal, es decir este conjunto de valores se ve determinado por una distribución Normal Gaussiana de sus valores logarítmicos.
Dado que para trabajar con ella se necesitan los mismos recursos que con una distribución Normal, para poder caracterizar esta distribución se necesitara un valor medio y una varianza o desviación típica.

2.1.2.2. Número de defectos

Este parámetro, como su nombre indica, es el número total de defectos o trampas que contiene un canal del dispositivo con el que se obtenga la traza.
Dependiendo del número de defectos la señal de salida tendrá un conjunto de valores posibles determinado. Dado que los defectos pueden estar ocupados o vacíos se pueden tratar como un

sistema binario de dos símbolos, por tanto, el conjunto posible de estados se rige por este parámetro, donde el número máximo de posibles estados en los que se puede encontrar la traza RTN viene determinado de la siguiente forma:

$$NEP = 2^N \quad (1)$$

NEP: Número de Estados Posibles, N: Número de defectos

Este conjunto de estados posibles es fruto de todas las combinaciones que se pueden dar en el estado de sus defectos.

Por lógica, el valor de este parámetro va a ser un número natural, por lo que de forma casi intuitiva se puede determinar que el número de defectos en un transistor viene establecido a través de una distribución de Poisson, el cual vendrá regido por un valor del que se deriva media y varianza llamado Parámetro Lambda.

En una representación gráfica este parámetro se manifiesta claramente, ya que, si fuésemos añadiendo defectos a una señal y por tanto augmentando sus estados posibles, iríamos viendo una discretización mayor en el eje de ordenadas para los valores de la señal obtenida.

2.1.2.3. Salto

El parámetro de salto hace referencia a la contribución numérica que tiene cada defecto particular en un canal respecto a la señal de salida.
Dependiendo de las posibles geometrías o irregularidades que se puedan dar a la hora de caracterizar un defecto, dicho defecto tendrá una contribución mayor o menor en el valor de corriente de salida, por tanto, cada uno de los defectos en un transistor tendrá asociado un valor de salto, lo que nos establece que cada traza tendrá asociados un número de saltos igual al número de defectos que contenga. Si se hacen todas las combinaciones de superposición entre estados de los defectos ponderados por sus correspondientes valores de salto se pueden extrapolar todos los estados posibles en los que se puede encontrar la traza RTN.

Estudiando las contribuciones de un múltiple número de defectos en las señales se puede determinar que los distintos valores de saltos vienen caracterizados a través de una distribución Exponencial. Esta distribución viene parametrizada solo por un valor medio.

En una representación gráfica, el salto se identifica al observar un cambio de estado, si el cambio de estado en la señal solo involucra un cambio de estado en un solo defecto, la diferencia entre niveles será directamente el salto asociado a ese defecto, pero al tener más de uno será una combinación superpuesta de dichos saltos, con lo cual no se pueden identificar cada uno fácilmente, aun así, se puede apreciar la contribución de los saltos en la señal obtenida.

2.1.2.4. Probabilidad de emisión y captura

Siguiendo la misma argumentación del parámetro anterior, las probabilidades de emisión y captura son consecuencia directa de la geometría especifica de cada defecto en el dispositivo

pertinente, y de la misma forma cada defecto tendrá asociada una probabilidad de emisión y una de captura.
Como su nombre indica, dependiendo de cómo sea el defecto tendrá una cierta capacidad u otra para retener o liberar portadores que circulen por el canal. Estas probabilidades no tienen por qué ser las mismas para la emisión y la captura, ya que las diversas geometrías de los defectos, así como la dirección en que circulen los portadores, pueden hacer que el defecto en cuestión sea más propenso a atrapar o a liberar un portador.

Estas probabilidades pueden ser estudiadas a nivel físico para obtener la distribución estadística que nos ayude a caracterizarlas, pero dado que la geometría de las irregularidades puede ser muy distinta y compleja esto genera unos valores con una altísima dispersión y entropía pese a poder caracterizarlas por una distribución estadística, lo que nos lleva a efectos prácticos a determinar que estas probabilidades de emisión y captura se pueden asemejar en gran medida a un número aleatorio entre 0 y 1.

En la representación gráfica de una traza se pueden detectar las probabilidades de emisión y captura observando la cantidad de veces que un defecto está o no ocupado. Por ejemplo, en una posible traza de un defecto, con probabilidad de captura muy alta y de emisión muy baja, esto se manifestaría en una traza que casi siempre está en el mismo estado por mucho tiempo que pase.

2.1.2.5. Estado inicial

El estado inicial es, como su nombre indica, en qué estado se encontrará cada defecto en el instante inicial, éste puede estar ocupado o libre.
Este parámetro viene estrechamente interrelacionado con el anterior, puesto que, dependiendo de la geometría de cada defecto y por tanto de sus probabilidades de emisión y captura, este defecto estará ocupado o no con distinta probabilidad.

El estado en el que se encontrará cada defecto viene descrito por la siguiente función que determina las probabilidades de que ese defecto esté ocupado:

$$P_O = \frac{1}{1 + \frac{P_E}{P_C}} \tag{2}$$

P_O: Probabilidad de ocupación, P_E: Probabilidad de emisión, P_C: Probabilidad de captura

2.1.2.6. Ruido de la señal

Para finalizar hay que contemplar un fenómeno que no es intrínseco a la fenomenología de las trazas RTN en la teoría, pero sí que es un factor del cual no podremos librarnos al estudiar o caracterizar trazas RTN, o prácticamente cualquier tipo señal electrónica. Este ruido que se contempla en las señales puede provenir de múltiples fuentes posibles y darse a causa de muchos fenómenos diversos.
Dado que cada equipo con el que se obtienen datos empíricos introduce una cantidad o forma de ruido distinta, hay que estudiar este parámetro de forma individualizada para cada caso específico. En el caso de nuestro laboratorio, como en la mayoría de los casos, el ruido que se aprecia en la señal es ruido blanco Gaussiano que como su nombre indica se distribuye según

una función Normal, y que por tanto requiere de media y varianza o desviación típica para ser caracterizado.

2.1.2.7. Resumen gráfico de los parámetros

En este último apartado se muestra de forma gráfica la contribución de cada parámetro en una traza RTN. Para ejemplificarse se ha usado una traza con 2 defectos (4 estados posibles según la eq. (1)) de 200 muestras, puede verse a continuación en la Figura 4.

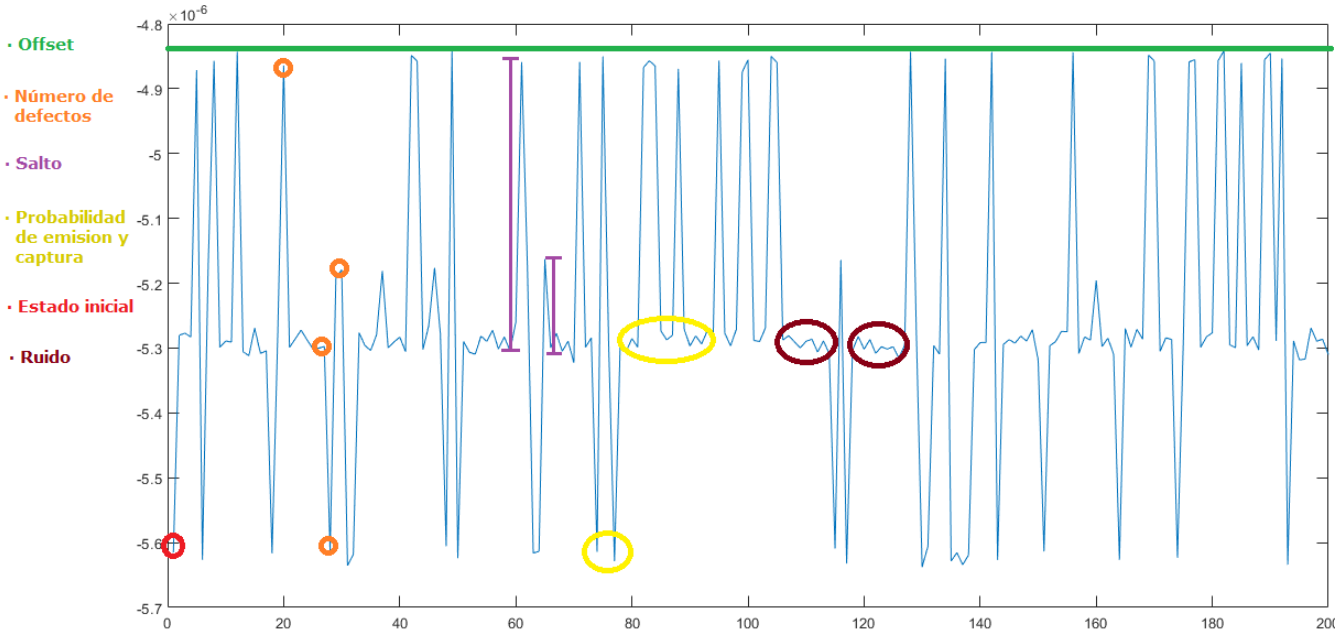

Figura 4: Ejemplo gráfico de la contribución de cada parámetro en una traza RTN. El eje horizontal representa la longitud de la traza y el vertical su valor en corriente (Amperios)

En la imagen se puede observar la contribución de cada parámetro al representar una traza RTN a lo largo del tiempo. Esta figura está elegida y dimensionada adecuadamente para su exhibición en este trabajo y por tanto se aprecian claramente los fenómenos comentados, pero normalmente se tendrá un volumen de trazas muy grande, de muchas muestras y posiblemente con muchos defectos, cosa que inhabilitará la viabilidad de estudiar gráficamente las trazas para algo más que un propósito académico, pero nos ayuda a asentar toda la información establecida anteriormente.

2.1.3. Estudio de los datos empíricos

Conocida la caracterización técnica de los parámetros de las trazas RTN ya se puede proceder a examinar la base de datos obtenidos empíricamente en el laboratorio. Estos datos consisten en un conjunto de 5 grupos de 504 trazas por grupo, por tanto, un total de 2.520 trazas RTN de 25.000 muestras por traza procedentes de 5 fuentes con parámetros distintos.

A partir de esta base de datos se pueden aplicar diversos procesos matemáticos, así como estadísticos, para la extrapolación de los valores clave que caractericen cada una de las distribuciones de probabilidad que rigen los parámetros de las trazas. Dado que hay muchos de esos procesos requieren un estudio muy en profundidad con bases de datos mucho más grandes de las disponibles, la mayoría de datos han sido obtenidos de fuentes externas, trabajos ajenos

a este o indicaciones proporcionadas por el director del TFG, pero uno de ellos sí que es de fácil análisis y con una implicación menor en la simulación de trazas RTN, este parámetro es el Offset. Con él se ejemplifica un posible medio de caracterización de los parámetros.

Para el estudio del valor medio y la varianza del Offset de las trazas en posesión hay que encontrar el valor más cercano a cero de cada traza. En este caso, dado que las trazas tienen valores negativos, aplicamos en MatLab un proceso para obtener el valor máximo de cada traza. Siendo conocedores que estos datos estarán distribuidos según una LogNormal, el siguiente proceso a aplicar es un logaritmo en base 10 a todos estos valores máximos, por consecuente al teórico Offset de cada traza. Habiendo aplicado este proceso ya se dispone de una base de datos distribuida de forma Normal. Para ejemplificarla la se pueden mostrar los datos según un histograma, éstos tendrán un grado mayor o menor de similitud con una campana de Gauss perfecta según lo bien distribuidos que hayan salido. A continuación, en la Figura 5, se puede observar dos histogramas que muestran los Offsets de un conjunto de 504 trazas procedentes de una fuente cada histograma, el primero con un resultado muy fiel a una Gaussiana y el segundo no tan similar.

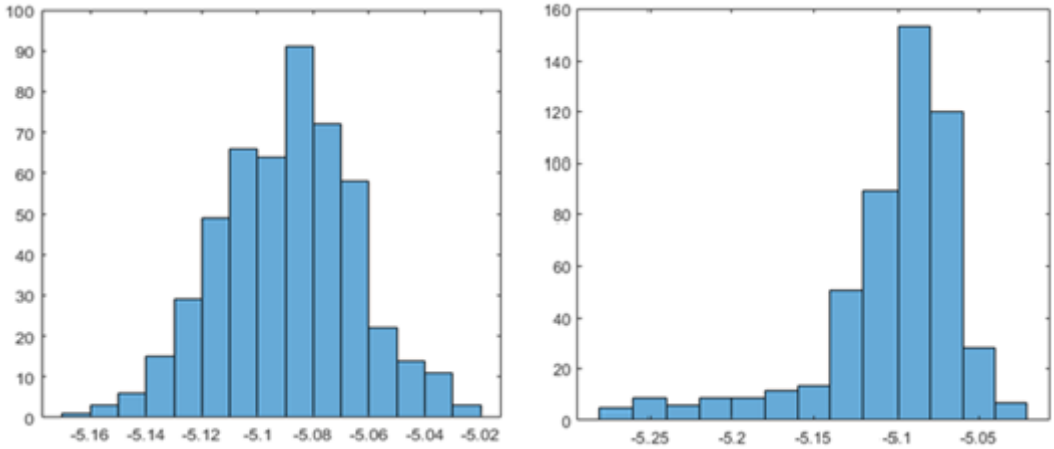

Figura 5: Comparativa entre dos histogramas obtenidos de la base de datos del laboratorio. El eje horizontal representa el valor del offset y el vertical la cantidad de veces que se da ese offset.

Para seguir ejemplificando mejor estos resultados se aplica también un análisis de los datos a través de un Gráfico Q-Q, el cual nos determinará la similitud de estos datos con una distribución Gaussiana según la cercanía de los puntos correspondientes a cada traza al eje diagonal que representa la Normal perfecta *[4]*. A continuación, en la Figura 6, se muestra este proceso aplicado a las mismas bases de trazas representadas de forma análoga en la figura anterior.

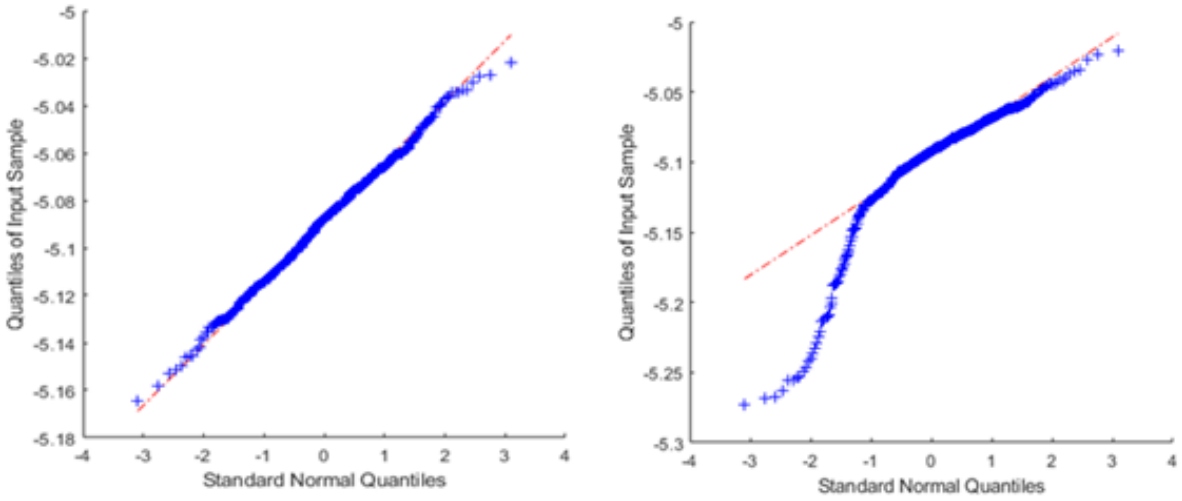

Figura 6: Comparativa de dos Gráficos Q-Q obtenidos de la base de datos del laboratorio.

Como era de esperar los análisis se corresponden entre histograma y Gráfico Q-Q.

Teniendo estos datos, y tratándose de la distribución estadística más estudiada de todas, hay múltiples herramientas o procedimientos con los que extraer los parámetros que caractericen la distribución estadística de los logaritmos de los Offsets de las trazas RTN. Sin ir más lejos, hay funciones básicas de MatLab que te devuelven dichos parámetros. En este caso los valores han sido los siguientes:

- Offset:
  - Media: -5,34188
  - Desviación típica: 0,067

Como se ha mencionado anteriormente, el resto de datos han sido obtenidos externamente de forma experimental, con lo que no se detalla la información precisa de la forma en que se han obtenido *[4]*, pero sí los resultados. A continuación se muestran estos datos con los que se trabajará posteriormente.

- Número de defectos:
  - Parámetro Lambda: 2

- Salto:
  - Media: $2 \cdot 10^{-7}$

- Ruido:
  - Media: 0
  - Desviación típica: $1{,}2 \cdot 10^{-8}$

2.2 Generación de la base de datos

Se ha incidido previamente en la necesidad de una base de muestras considerablemente grande para entrenar de manera efectiva una RN, como en este punto del proyecto ya se poseen datos numéricos que identifiquen un patrón aceptable de trazas RTN obtenidas a partir de datos empíricos, es posible desarrollar un método de simulación para replicar posibles trazas

obtenidas en un laboratorio con bastante similitud y precisión. A continuación se detallara la conceptualización y desarrollo del algoritmo del Simulador de trazas RTN, concluyendo dicho apartado con el diagrama de flujo correspondiente a este algoritmo.

El código íntegro que se usa en el proyecto, junto con todos los demás códigos usados, se encuentran de forma íntegra en el Anexo 1 al final del documento.

2.2.1. Creación del Simulador de trazas RTN

2.2.1.1 Conceptualización

Hasta ahora se ha descrito un modelo físico bastante aproximado para definir el fenómeno conocido de la generación de trazas RTN en dispositivos miniaturizados, pero para poder simular estas trazas, y por tanto crear un algoritmo en MatLab, es necesario describir un modelo matemático adecuado que describa el proceso.
El modelo físico conocido nos dice que las trazas RTN se generan a partir de las variaciones en el flujo de portadores debido a defectos en el canal que se ocupan y liberan variando particularmente la señal de salida estabilizada en un Offset.
Para trasladar este modelo físico al matemático trataremos los defectos como símbolos binarios, donde si el defecto está desocupado vale 0 y, por contra, si está ocupado vale 1. A cada defecto irán asociados los parámetros de salto, estado inicial y probabilidad de emisión y captura. Estos parámetros caracterizan dicho defecto, el cual cada vez que cambie de estado (ocupado o desocupado) variará la señal respecto al offset que se esté obteniendo a la salida. Los parámetros del defecto se trasladan al modelo matemático variando, o no, el estado de la traza en cada punto. Cuando un defecto está desocupado, y por tanto con un valor binario de 0, no estará contribuyendo a una variación de la señal en el modelo matemático, dado que en el físico no estaría alterando el flujo de portadores, lo que hace que la implicación del defecto en este punto de la traza sea nula.
Con el paso del tiempo el defecto acabará ocupándose siguiendo las probabilidades de captura que tenga, lo que dará lugar a una variación en la señal respecto al offset. Esta variación (así como el offset) será de valor positivo o negativo según la tipología del transistor MOSFET que se use para obtener las señales, en nuestro caso tanto el offset como las variaciones han sido de tipo negativo. El tamaño de esta variación irá ponderado por el parámetro salto, el cual da la magnitud de dicha variación. Si sigue pasando tiempo el defecto acabará desocupándose según su probabilidad de emisión lo que devolverá al estado de desocupación al defecto y dejará de contribuir a la señal de salida. Dado que un defecto puede estar inicialmente en cualquiera de los dos estados, el parámetro de estado inicial nos define dicha característica.

Según lo que se acaba de explicar podemos entender cada punto de la contribución de un defecto en una traza RTN según la siguiente ecuación:

$$Salida = Offset_{Tr} - Estado_{Def} \cdot Salto_{Def} \quad (3)$$

Tr: Trazas, Def: Defectos

En este punto se podrían caracterizar trazas RTN de 0 defectos, ya que la salida sería simplemente el offset, y de 1 defecto siguiendo la ecuación (3).
Lo normal será encontrarse dispositivos de múltiples defectos. Si seguimos trasladando el modelo físico al matemático podemos extrapolar que la contribución de los defectos a la señal

de salida es lineal, por tanto, podemos estudiar los defectos por separado y luego obtener la salida completa aplicando superposición entre ellos. Por su naturaleza, un defecto provoca una variación de la señal entre dos niveles de la señal de salida, a la que llamaremos 'Traza Simple'; en cambio, al superponer trazas simples de dos niveles asociadas a un defecto se obtiene una traza con un número de estados posibles que sigue la ecuación (1), a la que llamaremos 'Traza Compuesta'.

Entendiendo este modelo matemático, se pueden describir las trazas RTN (que contengan algún defecto) a partir de la siguiente ecuación:

$$Traza\ RTN\ (t) = Offset_{Tr} - \sum_{i=1}^{N^{o}Def} Estado_{Def_i}(t) \cdot Salto_{Def_i} \qquad (4)$$

Una vez obtenida esta Traza Compuesta le añadiremos el ruido pertinente para así obtener ya nuestra traza RTN definitiva.

Conocida la conceptualización del modelo matemático de descripción de trazas RTN podemos aplicarlo a un algoritmo para así generar el Simulador de trazas RTN.

2.2.1.2. Funcionamiento del código

Siguiendo el modelo desarrollado anteriormente, en esta sección se detalla en particular el algoritmo que seguirá el código generador de trazas RTN, el proceso es el siguiente:

Visto macroscópicamente este simulador se compone de tres fases, la generación de parámetros, la generación de trazas y la fase de adición de ruido.

- Etapa de generación de parámetros:

Partimos de la elección manual del número de trazas RTN a simular y la longitud de dichas trazas, por tanto, se asignan esos valores a unas variables específicas. Acto seguido se estipulan también el resto de los datos que parametricen la tipología de trazas que se quieran emular, como es ya conocido estos datos son los correspondientes al Offset, Número de Defectos, Salto, Probabilidad de Emisión y Captura, Estado Inicial y Ruido.

Habiendo ya asignado los valores de distribución de los parámetros de las trazas generamos el conjunto de parámetros que se asocien a cada una de las trazas que se deseen obtener.
Para generarlos se usa un bucle doble. En el bucle exterior generaremos el offset y el número de defectos siguiendo sus pertinentes distribuciones estadísticas, ya que son parámetros que no requieren de ninguna consideración previa, por lo que el bucle exterior irá de 1 al número total de trazas.
En cambio, dado que cada traza tiene un número variable de defectos y que cada defecto debe que ser parametrizado individualmente, en el bucle interior generaremos los parámetros asociados a cada uno de los defectos, es decir salto, probabilidades de emis. y capt. y el estado inicial. Este bucle interior debe atender a todos los defectos de la traza RTN en particular que se esté parametrizando, por consiguiente, será un bucle de 1 al número de defectos de la traza.

Como se ha mencionado anteriormente cada parámetro se modela según una distribución de probabilidad estadística determinada, por lo que debemos generar dichos parámetros en

consecuencia a su distribución. MatLab ofrece funciones que cubren todas estas necesidades computacionales, por lo que no hay más que buscar la función pertinente a la generación de datos según una distribución estadística concreta y aplicarla.
La única que no viene determinada es la del estado inicial. Esta debe ser generada dentro del bucle interior de defectos después de generar las probabilidades de emis. y capt., ya que depende de ellas. Para generarlo se aplica la ecuación (2), obteniendo una probabilidad de ocupación para ese defecto P_O. Aleatorizamos una variable auxiliar entre 0 y 1 y la comparamos con la P_O obtenida, si este valor aleatorio es menor que la P_O el estado inicial del defecto estará ocupado y por tanto con valor 1 al inicio del muestreo de la traza; por el contrario, si sale mayor el defecto estará libre y con valor 0.

Todos estos valores irán almacenados correlativamente en diversos vectores o matrices, según se vayan generando, para su posterior uso.
Para la coherencia en el tamaño de estos vectores y matrices, y de paso evitar posibles fallos en posteriores etapas, hay que añadir al código una condición la cual, si la traza que se esté parametrizando obtiene un número de defectos de 0, los valores de los parámetros asociados a los defectos de esa traza valdrán 0. Esto no tiene coherencia física, dado que si no hay defectos no se necesita ningún valor que los parametrice, pero sí coherencia computacional.

Al final de esta fase se tienen todos los parámetros matemáticos de cada una de las trazas que se desee simular.

- Etapa de generación de trazas:

Esta fase se compone de un bucle triple: el bucle exterior irá corriendo según cada traza compuesta, por lo que va de 1 al número total de trazas; en el bucle intermedio trabajaremos con cada uno de los defectos de esta traza ya que es donde se irán generando trazas simples, por lo que este bucle irá de 1 al número de defectos; y finalmente el bucle interior ira corriendo por cada una de las muestras de la traza generándolas, por lo que este bucle interior irá de 1 a la longitud de la traza.
El primer paso a realizar, el cual no es imprescindible pero sí recomendable, es generar la matriz vacía donde se almacenarán las trazas RTN una vez generadas, en este caso la matriz tendrá un tamaño de Nº de Trazas x Longitud.

Al principio del bucle exterior inicializamos en 0 las variables auxiliares que se usen en el proceso de forma que nunca haya problemas de uso de datos incorrectos debido a variables que conserven su valor anterior.
Seguidamente se entra al bucle intermedio para cada defecto de la traza. Limpiamos los valores de las variables auxiliares que se usen en ese bucle y cargamos el estado inicial del defecto pertinente de la traza correspondiente, el cual ha sido generado en la etapa de generación de parámetros, en una variable 'Estado'.
A continuación entramos al bucle interior que corre a lo largo de la longitud de la traza, en este bucle se generará cada traza simple asociada a un defecto de la traza compuesta. Para ello, en cada ciclo del bucle se comprueba el valor de la variable Estado. Si el defecto está libre se aleatoriza una variable auxiliar entre 0 y 1 y se compara con la probabilidad de captura, si este valor aleatorio es menor, el Estado cambia de 0 a 1, sino permanece con el mismo valor. En cambio, en caso de que el defecto se encontrase ocupado se haría el proceso análogo con valores invertidos y usando la probabilidad de emisión.
Una vez determinado el estado de ese defecto, en ese instante generamos el valor puntual de la traza simple, este valor se obtiene aplicando la ecuación (3), sin tener en cuenta el offset el cual se añadirá después. Haciendo correr este bucle interior se obtiene una traza simple de la longitud deseada la cual corresponde al defecto pertinente.

Con esta traza simple almacenada saltamos fuera del bucle interior de longitud y volvemos al bucle intermedio de defectos, que repetirá la misma acción que antes para el siguiente defecto de la traza RTN. Una vez está hecho el proceso análogo con cada defecto, a la salida del bucle intermedio, habremos obtenido un conjunto de trazas simples almacenadas, tantas trazas simples como defectos tenga la traza RTN compuesta con la que se esté tratando.
Finalmente volvemos al bucle principal para cada traza RTN. En este punto se realiza superposición de todas las trazas simples almacenadas junto con el offset, lo que da lugar a la traza compuesta. Esta traza compuesta es la traza final que obtener, excepto porque le falta añadir el ruido, cosa que se dará en la última etapa. Esta superposición puede hacerse de diversas maneras, en este caso se ha hecho a través de un bucle que vaya apilando (por tanto, sumándose al resultado que ya había en esa pila) cada una de las trazas simples junto con el offset.
Antes de salir finalmente del bucle exterior y concluir con esta etapa del simulador, hace falta primero añadir una condición por la que, si en esa traza hay 0 defectos, la salida de la traza será su offset sin ninguna variación; para finalizar, antes de salir de cada ciclo del bucle exterior se almacena la traza compuesta en el lugar correctamente indexado de la matriz que se ha creado al principio de la etapa.

- Etapa de adición de ruido:

Esta etapa final no tiene ninguna particularidad en especial. A cada punto de cada traza hay que añadirle el ruido blanco caracterizado por los parámetros introducidos en la primera etapa. Con esa intención se aplica un bucle doble que corra por las filas y columnas de la matriz obtenida en la etapa anterior variando cada punto añadiéndole un valor de ruido que generemos con la función de MatLab dedicada a generar ruido blanco. A la salida de esta última etapa ya se dispone de la matriz final que contiene el número deseado de trazas RTN finales con su pertinente longitud.

Una vez aplicado y ejecutado este algoritmo ya se tendrá en disposición la capacidad de simular trazas RTN. Seguidamente, en la Figura 7, se muestra un ejemplo de cuatro trazas simuladas.

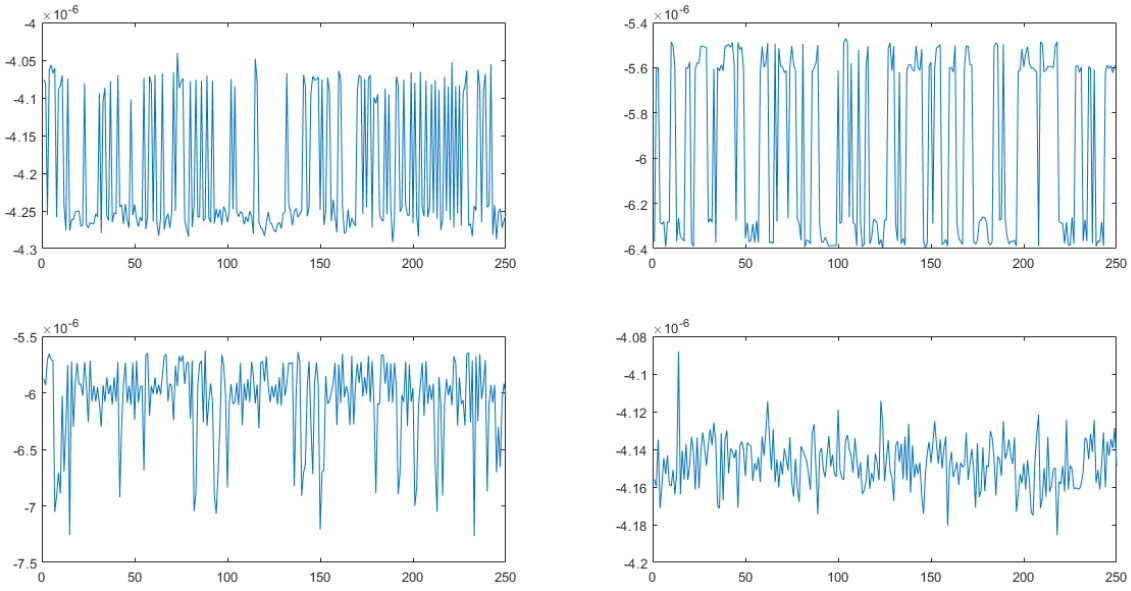

Figura 7: Trazas RTN simuladas a partir del código generado 'Simulador de Trazas RTN'.

2.2.1.3. Diagrama de flujo

Dado que este código puede ser un poco confuso descrito solamente con palabras, para esclarecer su descripción, a continuación en la Figura 8 se muestra el diagrama de flujo completo del Simulador de RTN.

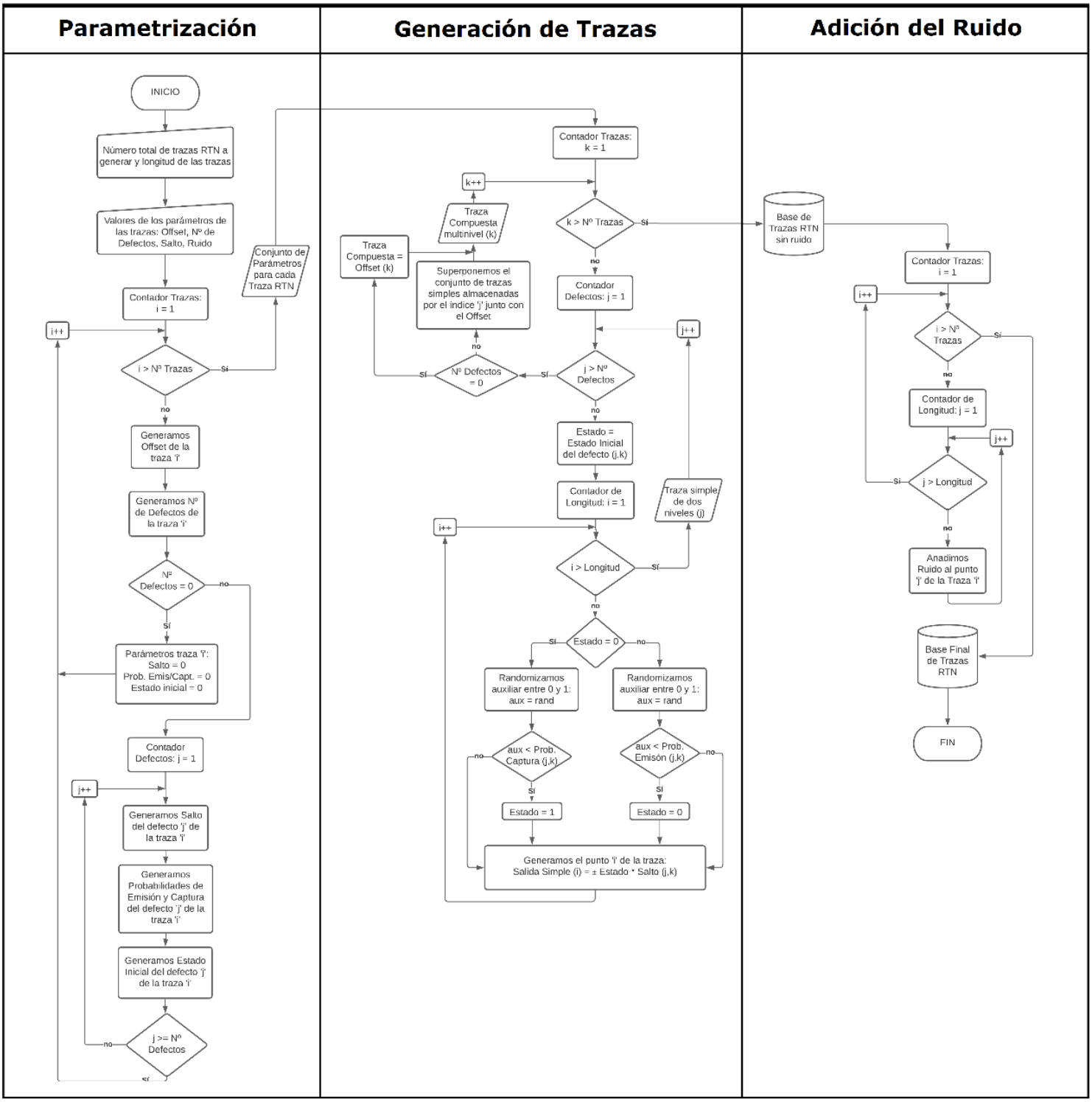

Figura 8: Diagrama de flujo del algoritmo del Simulador de RTN.

2.2.2. Tamaño de interés para la base de datos

Teniendo a disposición un código que nos permite simular todas las trazas que deseemos hay una libertad absoluta para elegir el correcto dimensionamiento de la matriz de datos que usaremos como base de trazas RTN con la que entrenar la RN, a cambio solamente de tiempo de computación y espacio de almacenamiento.
Esta particularidad será una constante a la hora de trabajar con las RN y las trazas RTN. Este proyecto está pensado para fines académicos y se dispone de un tiempo y recursos limitados, en concreto se muestran los componentes del ordenador con el que se ha realizado este trabajo en el Anexo 2 para ilustrar la capacidad de computación y su interrelación con el tiempo que se tarda en ejecutar algunos códigos. En un contexto profesional se podrían destinar muchos más recursos y tiempo para el desarrollo de una RN adecuada, por lo que hay que contextualizar los resultados y hacer un ejercicio de extrapolación para imaginar la capacidad real aplicable de esta metodología.

Para esta base de datos destinada a entrenar una RN que ejecute un proceso complejo, como se ha mencionado anteriormente, se le ha determinado un tamaño adecuado de 20.000 trazas de 25.000 muestras por traza. Esto da lugar a una matriz de datos con un peso de 3,7 GB la cual se ha tardado cerca de una hora y media en generar.
Este tamaño nos dará un buen margen para proporcionar un entrenamiento adecuado a la vez que se manejan unos tiempos de computación sostenibles para un trabajo académico.

A raíz de esta base de trazas, en posteriores apartados obtendremos las entradas para entrenar la RN a partir de la compresión de datos de la base; y las salidas de la RN a partir del establecimiento de un criterio de Aceptación o Rechazo de las trazas de la base de datos.

3. CRITERIO DE ACEPTACIÓN O RECHAZO

Una vez en posesión de la base de datos con la que trabajar habrá que discernir qué trazas debería dar por válidas la RN una vez entrenada y qué trazas no. Obteniendo dicha validación para las trazas de la base se tendrán las salidas objetivo para el entrenamiento de la RN.

Para poder determinar si las trazas son válidas hay que generar un código que automáticamente compruebe si cada una de las trazas sirve para nuestro propósito o no. Este código validador de trazas deberá atender a un seguido de especificaciones matemáticas con las que determinar su validez, a este conjunto de pautas se les ha llamado Criterio de Aceptación o Rechazo.

Para la determinación de este criterio se trabaja de forma distinta a la mayoría de etapas del proyecto, puesto que no se tiene un objetivo tangible en el que trabajar hasta haber establecido definitivamente el criterio que debe seguir la RN para discernir las trazas válidas para el objetivo deseado. Como este proceso no se puede extrapolar o sintetizar siguiendo modelos matemáticos se debe establecer a partir de decisiones lógicas arbitrarias pensadas específicamente para el caso, y después es cuando se le desarrolla un modelo matemático que cumpla dichas decisiones tomadas.
Este criterio no tiene una sola solución unívoca, más bien hay un abanico de posibles soluciones que se aproximan de distinta forma al resultado perseguido, estas posibles soluciones para ser válidas deben, tanto tener coherencia lógica en las características deseadas, como afinidad con la capacidad de una RN de desempeñar eficientemente dicha función una vez entrenada.

En el transcurso del desarrollo de este proyecto se ha trabajado con más de una versión del criterio, la cual ha ido evolucionando para refinar el conjunto de trazas que se daban por válidas. A continuación, en el siguiente apartado, se muestra la versión final del criterio.

3.1. Definición del Criterio

Este criterio focaliza en el control de la aleatoriedad que tendrían los símbolos binarios en los que se traduciría una traza RTN, de los cuales se busca la mayor incertidumbre entre símbolos binarios consecutivos.
Como el uso principal de las trazas en nuestra aplicación objetivo es obtener cadenas de bits aleatorios, para la determinación de la validez de una traza primero tendremos que traducir dicha traza a símbolos binarios con los que poder trabajar. Para lograrlo simplemente se medirá el valor medio de la traza y se determinará si cada punto está por encima o por debajo de su valor medio, asignando a ese punto un 1 o un 0 respectivamente. Este sistema se puede implementar tanto por software, con funciones en el código; como por hardware, con el uso de un comparador y demás componentes electrónicos simples.

A continuación se detallan las características con las que se ha definido este criterio.

3.1.1. Objetivos del criterio

Lo que nos permite el control de la aleatoriedad en los símbolos de una traza es poder determinar cuáles son las que más cerca están de tener una alta incertidumbre en la generación de los siguientes símbolos binarios al digitalizar la traza. Al tener control de dicho indicador, podemos tender a aceptar solo las trazas con más incertidumbre y, obviamente, la que generaría los símbolos binarios de forma más aleatoria posible.
Aclarado el objetivo al que dirigirse hay que establecer el modelo matemático que efectúe el discernimiento de trazas válidas.

3.1.2. Autocorrelación simplificada

Para plasmar este modelo matemático se ha realizado una abstracción a nivel computacional de la autocorrelación, al estar trabajando con señales es sencillo aplicar esta herramienta o recrearla de forma aproximada. La información que nos dará este proceso es la frecuencia con la que se repite consecutivamente un símbolo binario, precisamente la información que se buscaba para tener una idea de cuán aleatoria es una señal basada en una traza RTN.

En este caso se ha desarrollado un modelo simplificado basado en código. Este código, después de convertir una traza RTN en símbolos binarios como se ha explicado anteriormente, compara signos consecutivos. Si la comparación sale positiva se define esta interacción como tipo A, en cambio si sale negativa se define como B. Una vez realizadas todas las comparaciones entre signos consecutivos obtenemos la relación de la cantidad de coeficientes A entre el número de coeficientes B, por tanto, A/B. Si este número sale cercano a 1 significará que hay una cantidad muy similar de veces que el signo siguiente se repite o cambia, lo que axiomáticamente nos indica que hay una alta incertidumbre en el valor del siguiente símbolo.

3.1.3. Límites de Aceptación/Rechazo

Es cierto que el resultado óptimo que sería deseable tener en cada traza RTN es A/B = 1, pero si únicamente se aceptasen trazas de este valor prácticamente nunca se obtendría ninguna válida, evidentemente es imprescindible aceptar un rango de resultados alrededor del valor óptimo.
Para este caso se ha determinado que las trazas válidas tendrán como máximo un 15% más de un tipo de coeficiente de autocorrelación que de otro. Esto se traduce en que el límite superior para el criterio a partir de la relación A/B se establece en un valor de 1,15 y el límite inferior en 0,87. Estos límites pueden flexibilizarse al diseñar la RN puesto que dependiendo de la aplicación puedes permitirte priorizar entre cantidad de trazas válidas o la alta aleatoriedad de las trazas validadas.

3.1.4. Muestreo óptimo para la aceptación

La última característica que particulariza este criterio de aceptación o rechazo es el hecho de tener en cuenta cada cuánto se tomara una muestra para compararla con la siguiente para aplicar la autocorrelación. Hay algunas propiedades de las trazas RTN que se reflejan cada cierto tiempo, como por ejemplo los llamados defectos lentos, defectos que son muy poco propensos a cambiar de estado. Si se estudia qué muestreo nos proporciona un valor de A/B más cercano

a 1 para cada conjunto de trazas RTN podremos obtener una RN que mejore la cantidad de trazas que serían útiles, tan solo variando cada cuánto se coge una muestra de una traza RTN para generar una clave aleatoria.
En este trabajo se ha estudiado los valores de aceptación o rechazo cuando se muestrea la traza cada 1, 10, 1000 y 1000 muestras. Como se comprobará en futuros apartados el muestreo más adecuado para el tipo de trazas que se han generado en este proyecto es cada 10 muestras.

3.2. Validador de trazas

Habiendo establecido finalmente la definición formal y técnica de nuestro Criterio de Aceptación o Rechazo ya es posible generar un algoritmo que realice todo el proceso para obtener las salidas que representen los vectores objetivos con las que entrenar la RN a partir de nuestra base de datos, es decir podemos programar un código validador de trazas RTN.
A continuación se detalla el proceso que sigue este algoritmo junto con un diagrama de flujo.
Como el resto de los códigos, éste se halla íntegramente al final del documento en el Anexo 1.

3.2.1. Funcionamiento del código y diagrama de flujo

El funcionamiento de este algoritmo es bastante simple, se inicia estableciendo los límites de aceptación o rechazo pertinentes. Se carga la base de datos que se quiera evaluar y se calcula su tamaño, por tanto, el tamaño de la matriz que contiene esos datos.

Seguidamente se convierte la traza a valores binarios con un bucle doble que recorra cada punto de cada traza. Para ello se calcula la media de cada traza y se compara con cada elemento de la misma, si sale por encima de ese valor se le da a ese punto valor 1, sino valor 0.
A partir de estas trazas de binarias, en un bucle que recorra cada traza de la base, se hace la autocorrelación comparando símbolos consecutivos según un muestreo cada 1, 10, 100 y 1000 muestras, obteniendo los coeficientes A y B de la traza pertinente. Finalmente se realiza la relación A/B para cada traza y se determina si está dentro del rango aceptable entre 1,15 y 0,87, de ser así se determina la traza como válida y por tanto en el vector de aceptación o rechazo de las trazas introducidas habrá un 1 en la posición correspondiente a la traza evaluada; de no ser aceptada habrá un 0 en dicho puesto del vector. Estos vectores de salidas se almacenan por separado para cada tipo de muestreo, lo que nos permite usar separadamente el que mayor eficiencia nos otorgue.

A continuación, en la Figura 9, se ve el diagrama de flujo del proceso.

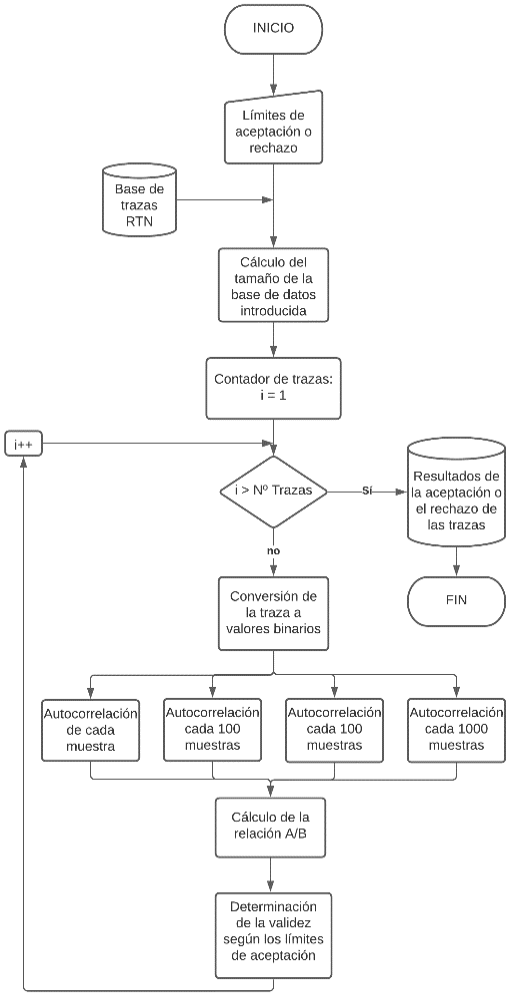

Figura 9: Diagrama de flujo del algoritmo del Validador de Trazas.

3.3. Generación de salidas de la RN, Salida Negada y ejemplo gráfico

Como se ha anticipado anteriormente, una vez aplicado el validador de trazas a la base de 20.000 trazas RTN con la que queremos entrenar la RN, obtendremos los resultados de autocorrelación para los diferentes muestreos aplicados. Con esta información podremos determinar qué resultados son los más eficientes y qué muestreo queremos aplicar a la traza RTN.

Los resultados de aceptación una vez procesada la base de datos son los siguientes:
Para muestreo de cada muestra se obtiene un 29,115% de aceptación, cada 10 muestras un 64,745%, cada 100 muestras un 54,505% y finalmente cada 1000 muestras un 14.025% de tasa de aceptación.
Como ya se había comentado, el muestreo que proporciona un mayor número de trazas válidas en este caso es cada 10 muestras, este será entonces el muestreo que se corresponderá al vector de resultados que usaremos como salidas objetivas en el entrenamiento de la RN. Estas salidas serán 1 y 0 determinando la validez o no de la traza introducida.

Una técnica para augmentar la precisión de la RN con respuestas binarias es la de añadir una segunda salida que corresponda al valor negado de la salida original, ya que polarizando los resultados a obtener se augmenta la eficiencia de acierto de la RN. Aplicando esta técnica se

obtendrá una red con 2 salidas, las cuales sólo sería válida la primera salida dado que la otra será su valor negado y por tanto incorrecto.

A continuación, a modo de información visual, se muestran los resultados del proceso realizado a una base de ejemplo de 100 muestras a modo de poder representar gráficamente la distribución de datos. Esta base no da los mismos valores que la base grande, pero sí debería dar una aproximación de éstos. Se muestra en la Figura 10.

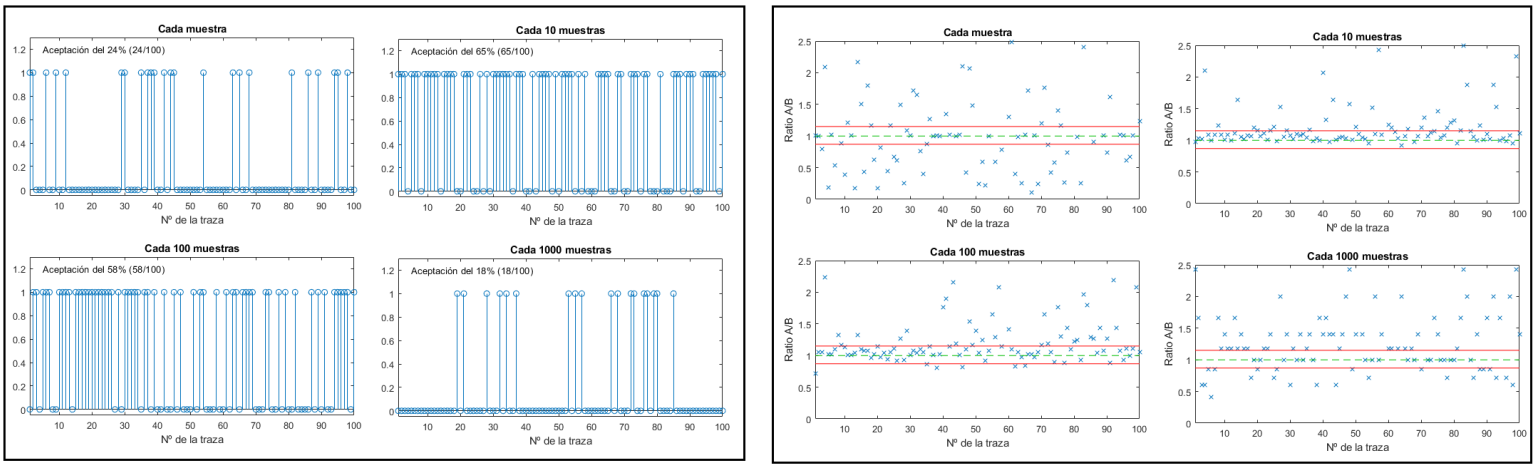

Figura 10: Datos de validación o rechazo de la base de ejemplo de 100 trazas. La sección izquierda de la figura muestra si la traza ha sido aceptada o no, y la parte derecha su correspondiente relación A/B y su posicionamiento en el rango de validez.

4. RED NEURONAL

En este punto del proyecto ya se está en posesión de una base de datos con la que entrenar la RN pertinente, junto con los resultados objetivos que se necesitan para un entrenamiento supervisado. Teóricamente con estos dos elementos se podría entrenar una RN, pero tal y como se ha mencionado antes hay que manipular los datos de entrada con el fin de comprimirlos ya que hay que dimensionar correctamente la capa de entrada de la RN y a la vez no es eficiente trabajar con un tamaño de datos grande como el de una base de trazas RTN.

Una vez comprimidos estos datos se obtendrán las entradas de la RN. Dado que se ha entrenado así, la RN sólo entenderá entradas comprimidas de la misma forma que con las que se ha entrenado.

Una vez se haya aplicada esta compresión se procederá al diseño y creación de la RN.

4.1. Compresión de los datos de entrada

Si bien es cierto que en este punto en teoría se podría generar una RN usando como entradas para el entrenamiento las trazas RTN en sí, a la práctica no tiene sentido usar trazas sin tratar como entradas de la red por las limitaciones que ello implicaría. Hacerlo de dicha manera supondría, entre otras cosas, la necesidad de usar siempre trazas de la misma longitud para el uso de la RN, teniendo que ser dicha longitud bastante corta dado que el número de entradas de la RN debe ser coherente con los valores adecuados de dimensionamiento de capas de una RN; así mismo dicha red solo serviría para un tipo de transistor dado que el signo de la señal sería un factor a tener en cuenta. Sumado a todo esto también se encuentran inconvenientes de segundo orden como la incomodidad de trabajar con bases de datos muy grandes o el coste computacional que ello implica.

Para poder obtener un conjunto de entradas válidas para entrenar la RN hay que tratar la base de datos de trazas RTN y modificarla a través de distintos procesos matemáticos para obtener una representación adecuada dichas trazas, reduciendo mucho el volumen de datos, pero sin perder la información clave con la que identificar cada traza.
Estos procesos matemáticos no están estipulados ni tienen una solución unívoca, tal y como sucedía con el Criterio de Aceptación o Rechazo. Dependiendo de la metodología que se tome al sintetizar los datos se obtendrá una representación de mejor o peor forma del objeto que se desea emular. Esta representación puede conservar la información de distintas características, así como desechar otras, ofreciendo mucha flexibilidad al sistema.
Dependiendo de la aplicación para la que se vaya a usar habrá que tomar una estrategia u otra para la compresión de los datos, en los siguientes apartados se desglosa la metodología que se ha usado en este proyecto.

4.1.1. Valor Absoluto

La primera modificación a realizar es la de aplicar el valor absoluto a la base de trazas RTN. Esta operación eliminará el signo de las trazas por lo que no tendrá importancia si la traza viene de una fuente como un pMOS o un nMOS. En este caso el uso del valor absoluto nos aporta la

flexibilidad de que las trazas sean de cualquier signo, y como que las cadenas de trazas RTN están destinadas a acabar siendo símbolos binarios, la información de si la traza RTN es positiva o negativa no tiene valor.
Como se ha comentado unas líneas atrás cada operación que se aplique para sintetizar datos conservará unas características y desechará otras, en este caso, si las trazas estuviesen cercanas al 0 y la traza incluyese valores positivos y negativos al mismo tiempo, realizar esta operación distorsionaría la información de la traza resultando en errores en nuestro sistema. Para resolver esto hay que cerciorarse de que los offsets de las trazas sean suficientemente grandes para que ningún posible valor de la traza sobrepase el 0 en ninguno de los dos sentidos. Como en nuestro caso es así, el uso del valor absoluto nos ofrece solo ventajas, flexibilizando las fuentes de datos, así como simplificando la dificultad para la RN para discernir las adecuadas, y sin ninguna penalización en la información útil de la traza.

4.1.2. Histograma

Dado que cada señal RTN puede estar posicionada en un número determinado de valores, pero estos valores no serán siempre exactamente iguales a causa del ruido experimentado, hay que hacer una síntesis de la cantidad de veces que la traza está en un punto u otro y esto se puede obtener a partir de la realización de un histograma para representar una traza RTN. En este caso, un histograma cuantificaría los valores en los que se puede hallar la traza y mostraría la frecuencia en que se dan cada uno de ellos.
Si bien podría parecer que aplicar un histograma directamente a la traza puede ser una representación adecuada, hay información relevante para caracterizar a las trazas RTN que se perdería, como por ejemplo la posible detección de defectos lentos o la percepción de la autocorrelación de la traza y sus cambios de estado.
Dado que el uso de un histograma no es adecuado, pero sí que necesitamos de esa cuantificación de niveles para representar la información comprimida, hay que llevar el concepto de histograma a un siguiente nivel.

4.1.3. Time-Lag Plot e Histograma bivariado

Para solucionar las carencias del uso del histograma simple se ha usado el análisis matemático Time-Lag Plot, este proceso es una herramienta útil para el tratamiento de datos en múltiples ámbitos. Este análisis se basa en el estudio de la correlación de dos series de datos distintas, en nuestro caso, como queremos saber la autocorrelación, comparamos cada traza con ella misma desplazada una unidad en longitud. Esto nos ofrece un mapa gráfico bidimensional que dependiendo de la forma que tenga implicara unas características de autocorrelación u otras, y por tanto la caracterización efectiva de cara traza RTN. Esta información puede usarla una RN para aprender si la traza RTN nos sería adecuada o no.
Este sistema sí tiene en cuenta la información que se perdía usando un histograma simple, como los defectos lentos y demás factores, conservando los datos de correlación que nos son útiles. Pero al tratarse de un análisis gráfico en dos dimensiones deberemos aplicar un histograma bivariado, es decir de dos variables, que tenga en cuenta la posición de cada dato tanto en el eje horizontal como en el vertical, el cual aplicará la cuantificación de datos por niveles que nos interesaba tener en el mapa bidimensional creado a partir del Time-Lag Plot.

Al estar representando un tercer dato (la cantidad de veces que se da un evento) en un plano de coordenadas, la visualización de estos datos debe hacerse en 3 dimensiones o tener

herramientas gráficas para su entendimiento. A continuación, a modo de apoyo visual, se muestra en la Figura 11 una representación del histograma bivariado del Time-Lag Plot de una traza RTN.

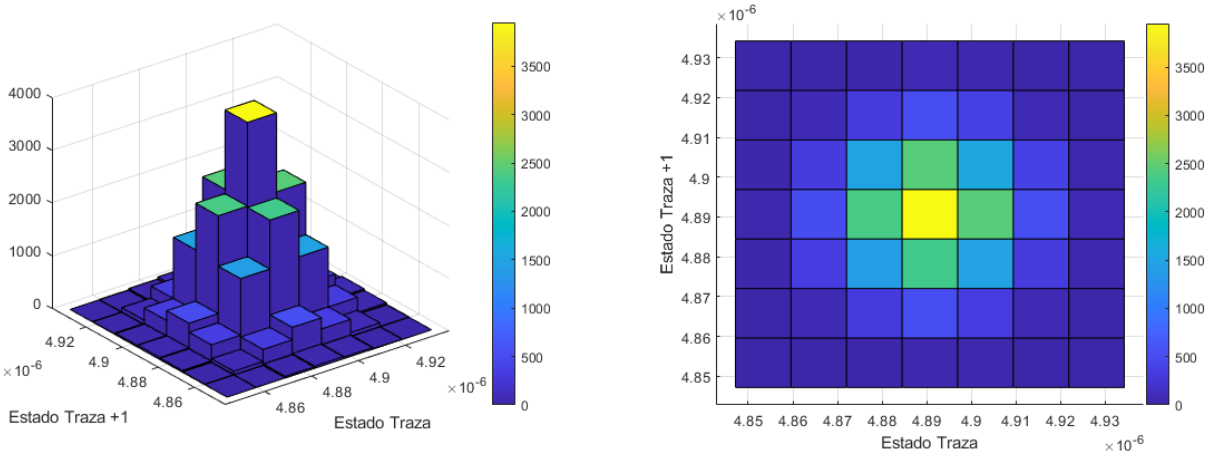

Figura 11: Representación 3D y 2D del histograma bivariado de un Time-Lag Plot de una traza RTN aleatoria.

Pese a que a nosotros nos ayuda esta representación visual de los datos, la RN no necesita ver coherencia estadística en los datos de entrada, y funcionará siempre y cuando las trazas RTN que se introduzcan se manipulen de la misma forma que las que se usaron para entrenarla.

Una vez aplicado el histograma bivariado obtenemos una matriz de datos que representa bastante adecuadamente una traza RTN determinada, pero no es posible usar una matriz como entrada a una RN, por lo que habrá que fragmentar esta matriz resultante y construir un vector con sus datos.
Como que los histogramas bivariados dan lugar a una cantidad cuadrática de datos en referente a la cantidad de secciones que se quieran usar, en el histograma las entradas crecerán exponencialmente como N^2, es decir si el histograma es de 2x2 nuestra RN será de 4 entradas, histograma de 3x3 serán 9 entradas, 4x4 serán 16 entradas, y así en adelante, lo que implica un compromiso entre lo precisos que sean los datos y la cantidad de entradas que necesitará la RN, precisión a cambio de complejidad y viceversa.

4.1.4. Normalización

La última modificación por aplicar a la base de datos antes de considerarlas como entradas válidas para la red neuronal es la normalización de los valores respecto a longitud de la traza introducida, esta operación hará pasar de valores absolutos a valores relativos. Este cambio aporta flexibilidad a la RN respecto a la longitud de la traza RTN que se introduzca, ya no será necesario que las trazas tengan longitudes similares para ser correctamente analizadas por la RN a causa de que los números estén en valor absoluto y no en proporción al número total de muestras de la traza.
En vez de eso, al trabajar con datos porcentuales, la longitud de la traza solo tendrá relevancia en cuan bien representada está, obviamente contra más muestras se tengan de dicha traza mejor representada estará, pero normalizando las entradas se quita la necesidad de introducir trazas de longitudes similares para que la RN las entienda.

4.2 Compresor de datos

Siguiendo los criterios de compresión anteriores hay que desarrollar un código que los aplique a una base de trazas RTN, dando como resultado una base de datos comprimida representativa de la original.

4.2.1. Funcionamiento del código y diagrama de flujo

Como las dos anteriores, este último código aquí explicado se encuentra de forma íntegra en el Anexo 1 junto a los demás códigos.

Este algoritmo debe modificar la base de trazas RTN para entrenar la red, así como modificar cualquier otra traza que se quiera introducir a la RN, ya que ésta sólo entenderá el formato de datos con la que la hemos entrenado. En esencia hay que generar un código que aplique cíclicamente a cada traza de una matriz de datos las modificaciones que se han mencionado anteriormente.

El primer paso es determinar la cantidad niveles que discriminará el histograma bivariado, al ser un cuadrado se usa la longitud del lado. Esto determinará el número de cuadrantes en los que hallarse al aplicar el histograma bivariado, por tanto, en la cantidad de datos necesaria para representar una traza RTN y asimismo la cantidad de entradas que requerirá la RN. Como ya se ha mencionado este valor sostiene un compromiso entre la precisión de los datos y la complejidad de la RN.
Una vez determinado dicho valor se introduce una base de trazas RTN y se calcula el tamaño de la matriz que contiene estos datos. Seguidamente se genera un bucle para tratar por separado cada traza. En este bucle lo primero será copiar y almacenar la traza pertinente dos veces en una matriz, pero a la primera de las copias le quitaremos el primer valor y a la otra le quitaremos el último valor, esto simulará el hecho de estar desplazadas por una unidad. A estas copias se les aplica seguidamente el valor absoluto y se genera con ellas el histograma bivariado a partir del Time-Lag Plot. En este punto tenemos la matriz de datos en forma de histograma bivariado en valor absoluto, por tanto, dividimos todos los datos por la longitud de las trazas introducidas, normalizando así las muestras.

Aquí ya se disponen de los datos ya sintetizados, pero aun en forma de matriz, y nos interesa que esté en un sólo vector. Para ello hay que ir cortando cada fila de datos de la matriz y concatenarlas en un vector. Para implementar esto forma de código hay que inicializar un contador auxiliar y generar un bucle que vaya de 1 al número determinado al principio que representa la longitud del cuadrado del histograma bivariado. En este bucle simplemente se cortará una de las filas de la matriz y se concatenará al lugar adecuado del vector resultante el cual estará indexado por el contador auxiliar.
Para ejemplificarlo, si se quiere que el histograma bivariado sea de 3x3, este bucle irá del 1 al 3, en la primera iteración copiará la fila de arriba de la matriz de 3x3 y las pondrá en las posiciones 1, 2 y 3 del vector de salida; en la segunda iteración copiará la fila intermedia de la matriz en las posiciones 4, 5 y 6 del vector; y finalmente en la última iteración copiara la fila inferior de la matriz en las posiciones 7, 8 y 9 del vector, habiendo concluido con todas las operaciones necesarias.
Para acabar sólo queda almacenar el resultado en su sitio pertinente y se habrá obtenido el código completo.

A continuación, en la Figura 12, se muestra el diagrama de flujo pertinente al código que se acaba de detallar a modo de apoyo visual a la explicación.

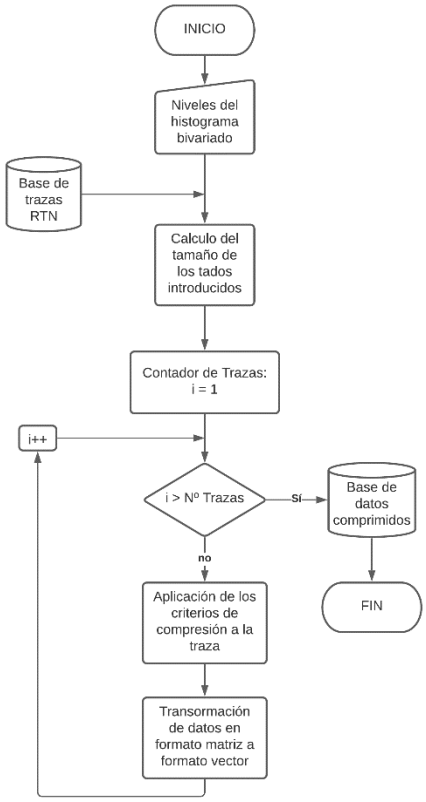

Figura 12: Diagrama de flujo del algoritmo del Compresor de datos.

4.2.2. Generación de las entradas de la RN

Para obtener las entradas finales con las que entrenar la RN habrá que someter a la base de datos con 20.000 trazas RTN a las modificaciones para su compresión a través del algoritmo previamente explicado. Como ya se ha argumentado, dependiendo del tamaño del histograma bivariado tendremos un número estipulado de datos de entrada distribuidos cuadráticamente.
Como a priori no se tienen datos del compromiso de precisión y eficiencia de la RN vs la complejidad y coste de computación, en este proyecto se ha hecho un estudio con un abanico de posibilidades, desde histogramas de 2x2 que genera una RN de 4 entradas hasta histogramas de 7x7 que dan lugar a RN de 49 entradas. Con los datos de eficiencia y tiempo que nos ofrezca el uso de diversos tamaños de compresión de datos se podrá estudiar y graficar cómo se comporta la RN respecto a su complejidad, así como extrapolar datos que no podamos obtener por nosotros mismos. En futuros apartados se mostrarán estos datos y su comparativa con otras pruebas de eficiencia.
Una vez aplicados los distintos niveles de compresión (2x2, 3x3,…,7x7) a la base de trazas, tendremos 6 bases de datos que representan de distinta forma a la original, que será la misma para todos, por tanto, se corresponden las mismas salidas. Para cuantificar brevemente la compresión realizada en el proceso, partimos de una base de trazas RTN de 20.000x25.000 que contiene 500 millones de datos, hacia un conjunto de bases la cual la más grande es de 20.000x49 con tan solo 980.000 datos, lo que nos dice que la base ha reducido algo más de 500 veces su tamaño original sin perder la información característica de la base original.

Estas bases de datos serán las diversas entradas para entrenar las distintas tipologías de RN que se generarán.

4.3. Diseño y entrenamiento de la RN

A partir de este punto ya se han obtenido todos los recursos que demanda una RN para ser correctamente entrenada, por lo que de ahora en adelante el trabajo estará orientado al estudio y análisis de las distintas RN que se generen con los datos que han sido generados a lo largo del proyecto.
En los siguientes apartados se verán los conceptos teóricos intrínsecos a las RN, como las posibles metodologías para obtener una RN funcional, así como distintas técnicas para augmentar la eficiencia de éstas junto con otras características relevantes.
Seguidamente se generarán los distintos tipos de RN y se estudiarán los resultados obtenidos.

4.3.1. Tipos de RN

MatLab ofrece 4 tipos de RN distintos, pero solo dos de ellos son para RN multicapa con entrenamiento supervisado, lo que en el aplicativo llaman “two-layer feed-forward”.
Los dos tipos de redes que nos interesan son las de tipo “Fitting”, la concepción original de RN que ajusta valores para estimar la salida más correcta, y las de tipo “Pattern Recognition”, dedicadas a buscar patrones numéricos en una base de datos.
Como ambas vertientes podrían ser útiles en el reconocimiento de trazas RTN en este proyecto se ha trabajado con ambos tipos de RN con el fin de determinar de forma empírica cuál es la más idónea para este caso.

4.3.2. Estructura de la RN

Como ya se ha explicado las RN se componen de tres capas de perceptrones: la capa de entrada, la capa oculta y la capa de salida.
Por norma general, y también en este caso, el tamaño de las capas de entrada y salida vendrán estipulados por características no intrínsecas a una RN, y aun teniendo cierta libertad, siempre irá ligado a las condiciones externas involucradas. En cambio, para la capa oculta no hay una restricción en número de perceptrones que contenga, sino que volverá a ser un compromiso entre el rendimiento de la RN y su complejidad. En los límites, si se tiene una red con pocos perceptrones en la capa oculta, el rendimiento de ésta será muy bajo; por el contrario, si se tienen demasiados perceptrones se estará haciendo un sobreuso de recursos con las consecuencias negativas que ello comporta. Es por eso por lo que la elección de un número correcto de perceptrones en la capa oculta es un aspecto importante al generar una RN.

4.3.2.1. Análisis de modelos para la capa oculta

La búsqueda del mejor modelo a seguir para medir correctamente el tamaño de la capa oculta en una RN sigue siendo objeto de estudio de todos los entendidos en la materia, puesto que

solo hay unas pocas reglas empíricas que funcionan en mayor o menor medida dependiendo de la aplicación para la que estén destinadas.
En su libro de introducción a las RN, el doctor en ciencia computacional y divulgador Jeff Heaton condensa estas reglas usadas experimentalmente en tres posibles metodologías a seguir, las cuales algunas pueden ser exclusivas entre ellas. Estas son dichas metodologías *[5]*:

- El número de perceptrones en la capa oculta debe estar entre el tamaño de la capa de entrada y el tamaño de la capa de salida: $\mathrm{N_{CE}} > \mathrm{N_{CO}} > \mathrm{N_{CS}}$

- El número de perceptrones en la capa oculta debe ser 2/3 del tamaño de la capa de entrada, más el tamaño de la capa de salida: $\mathrm{N_{CO}} = \frac{2}{3}\mathrm{N_{CE}} + \mathrm{N_{CS}}$

- El número de perceptrones en la capa oculta debe ser menos del doble del tamaño de la capa de entrada: $\mathrm{N_{CO}} < 2\mathrm{N_{CE}}$

Aun estar estipulas estas reglas, su autor nos advierte que sólo son normas generalistas las cuales hay que adaptar correctamente a cada caso específico. Con ese propósito, usaremos la segunda regla mencionada para el cálculo del tamaño de la capa oculta (aunque se cumple también la primera regla), y la compararemos con ella misma si fuese un 25% más grande o un 25% más pequeña que la regla original a fin de poder obtener datos de la implicación del tamaño de la capa oculta.

Para aplicar esta segunda regla hay que conocer previamente el tamaño de la capa de entrada y de salida. Para la capa de entrada ya se ha comentado que se usará 6 tipos distintos de compresión de datos, lo cual genera distintas entradas para la red que ya se conocen: 4, 9, 16, 25, 36 y 49. Como cada nivel de compresión genera un tamaño de capa de entrada distinto, los tamaños de la capa oculta tendrán que ir acordes a estos tamaños de entrada.
Respecto a la salida, a lo largo del trabajo se ha trabajado con una salida que determinase si la traza es válida o no, a la cual se le añade su homónima negada correspondiente. Esto conlleva que por cada entrada a la RN se obtendrán 2 salidas, con lo que la capa de salida tendrá un tamaño fijo de 2 perceptrones.

Seguidamente, en la Figura 13, se muestra en una gráfica el resultado del cálculo del tamaño de la capa oculta de cada tipo de red, siguiendo las tres versiones que se desea estudiar del modelo de Jeff Heaton, la normal, la +25% y la -25% (a las que a partir de ahora llamaremos como JH, JH+25% y JH-25%, respectivamente).

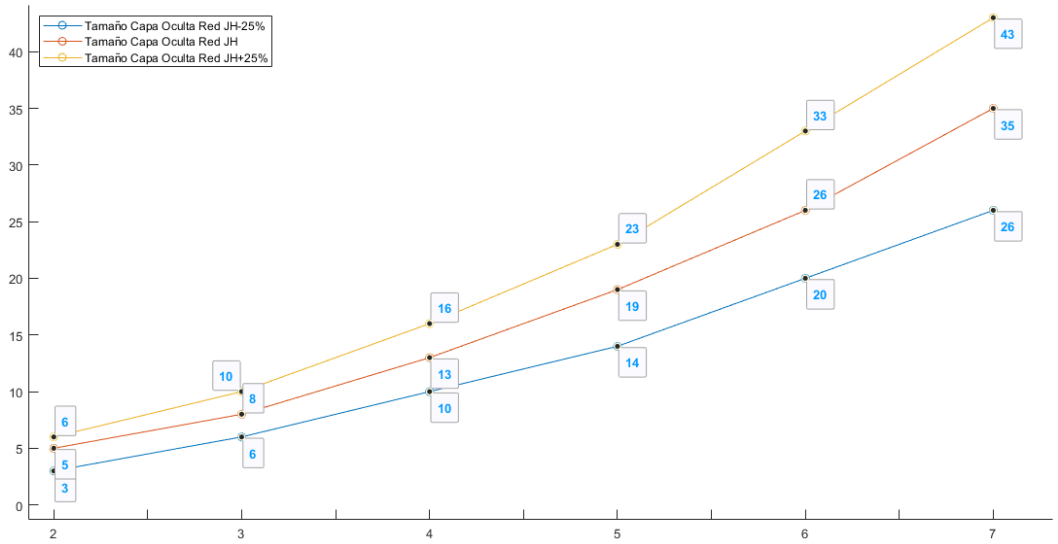

Figura 13: Tamaños de la capa oculta de las distintas RN a generar. El eje horizontal representa el tamaño de compresión y el vertical el tamaño de la capa oculta.

4.3.3. Tipo de entrenamiento

Entrenar una RN es en sí el ajuste de valores en los pesos de las conexiones entre perceptrones de la capa oculta que hacen que a través del procesado matemático de la señal entrada se tienda a una salida correcta. Hay diversas maneras de encontrar esa combinación de valores en los pesos, cada una con sus características y aplicaciones concretas.
El aplicativo de la toolbox de Deep Learning de MatLab que se usa en este trabajo para generar y entrenar RN permite tres métodos de entrenamiento: el Levenberg-Marquardt, la Regularización Bayesiana y el Escalado de Gradiente Conjugado.

El primer y tercer tipo están destinados a trabajar con bases de datos sencillas y amplias, uno consumiendo bastante memoria a cambio de una computación muy rápida y el otro con un requerimiento de memoria muy bajo, respectivamente. Estos dos tipos de entrenamiento se paran cuando detectan que ya no se sigue mejorando el resultado para evitar el sobreentrenamiento y están destinados a entrenar RN con funciones sencillas, por lo que no nos serán útiles.
En cambio, el método de la Regularización Bayesiana está pensado para trabajar con bases de datos complejas, relativamente pequeñas, con cierto nivel de ruido y con un ajuste bastante óptimo del valor de los pesos en la capa oculta, lo que será muy útil en una base de datos como la que se maneja en nuestro caso *[6]*. Esta tipología de entrenamiento está también diseñada para evitar el sobreentrenamiento, por lo que no habrá problemas de ese tipo, pero pueden darse tiempos de entrenamiento algo mayores al resto de métodos.

4.4. Creación y análisis de los distintos tipos de RN

En esta fase del proyecto ya nos es posible creación de diversos tipos de RN. Como aún no se dispone de información sobre qué configuración es mejor para la RN en esta aplicación habrá que crear un abanico de diferentes tipologías de RN y compararlas entre ellas.
El primer análisis a realizar será el de cuál es la RN más adecuada en esta aplicación, si una de tipo “fitting” o “pattern recognition”. Para ello se crearán RN de ambos tipos con los niveles de

compresión del 2x2 al 7x7 y con la cantidad de perceptrones en la capa oculta según la regla de JH.
Una vez se haya determinado qué tipo de RN se usará volveremos a comparar todos los niveles de compresión según los 3 tamaños de capa oculta: JH, JH+25% y JH-25%.
Con los resultados que se obtengan podremos extraer una serie de conclusiones y predecir el comportamiento de la RN fuera del rango de las pruebas realizadas.

4.4.1. Test, resultados y comparativas

Una vez generadas las RN, para poder comparar sus prestaciones habrá que diseñar un pequeño test. Éste no será más que la creación, validación y compresión de una pequeña base de datos que hayamos generado, obviamente distinta a la base con la que se haya entrenado las RN. Cuando se pasen los datos por la RN y se comparen con los datos de validación de esas mismas trazas se puede comprobar la eficiencia de la red comparando el número de aciertos respecto a la clasificación de las trazas de prueba. Para realizar estas pruebas se ha creado una base de datos de test de 2000 trazas RTN.

4.4.1.1. Comparativa de tipos de red

Como se ha comentado previamente, lo primero será discernir la tipología de RN entre “fitting” o “pattern recognition”. A continuación, en la Figura 14, se muestran los resultados de la comparativa tras el test de eficiencia de los dos tipos de RN para distintas compresiones con tamaño JH para la capa oculta.

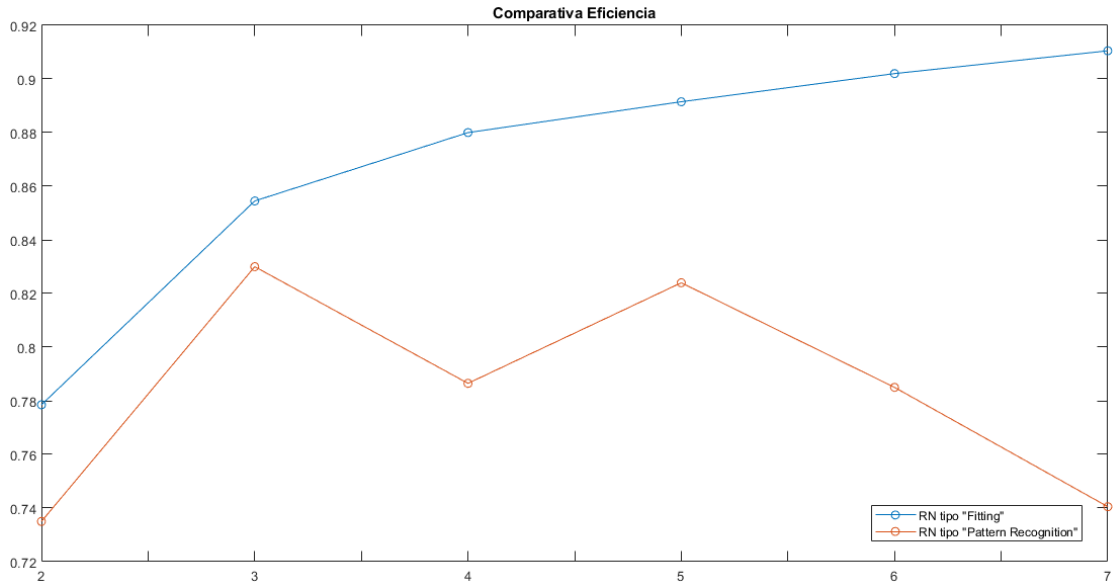

Figura 14: Comparativa entre RN tipo Fitting y tipo Pattern Recognition. El eje horizontal representa el tamaño de compresión y el vertical la eficiencia.

Como se puede comprobar fácilmente en la gráfica, pese que hay compresiones que hacen que el tipo Pattern Recognition parece que funcione relativamente bien, no hay duda de que la tipología más adecuada en este caso es la de Fitting, la cual es mejor y más regular a lo largo de todos los tamaños de compresión, por lo que será la que se elija para continuar con el proyecto.

4.4.1.2. Comparativa de tamaños de capa oculta

Con la tipología de RN determinada, la siguiente prueba será la variación y comparación de distintos tamaños de capa oculta (JH, JH+25% y JH-25%) para cada uno de los tamaños de compresión. En la Figura 15 se muestran los resultados obtenidos.

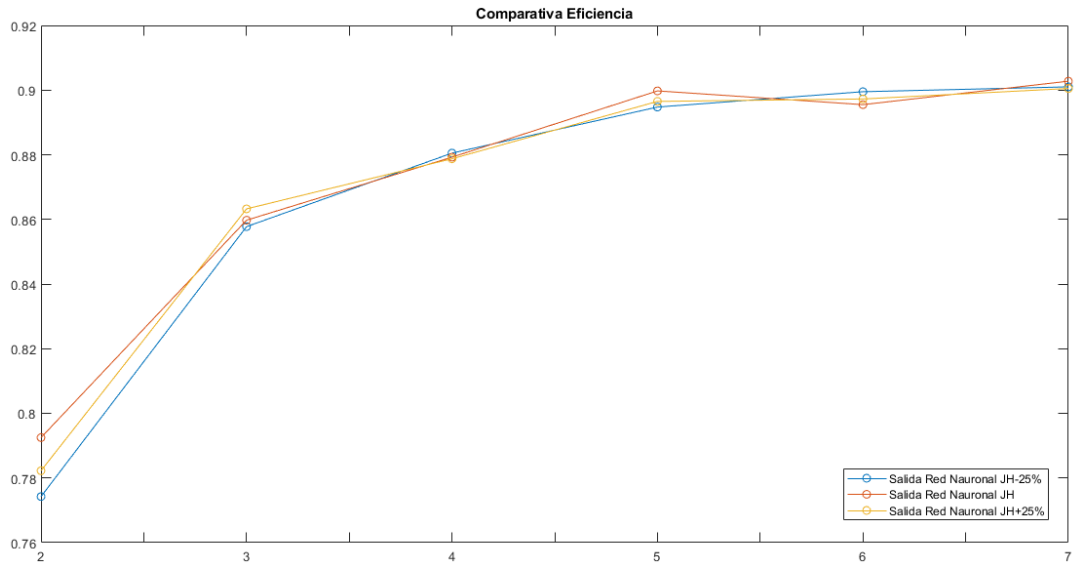

Figura 15: Comparativa de los distintos tamaños de capa oculta. El eje horizontal representa el tamaño de compresión y el vertical la eficiencia.

De esta gráfica se pueden deducir datos muy interesantes. El primero de todos y más obvio, como también ya se podía ver en la comparativa entre tipos de RN, a menor nivel de comprensión (es decir, contra más datos discierna el histograma bivariado, siendo 7x7 la configuración con menos compresión de las estudiadas) mayor será la eficiencia de la RN. Esto era algo que ya se podía suponer anteriormente, en sí, contra más precisión se tenga al cuantificar los niveles en la etapa de compresión de datos, mejor se puede categorizar una traza.

A destacar que el comportamiento de esta función tiende asintóticamente a un punto máximo de eficiencia posible para la RN. La máxima eficiencia que se ha logrado con las estipulaciones de las redes que se han generado es algo más del 90%. Teniendo en cuenta que este es un trabajo puramente académico para el que se disponen recursos y tiempo limitados es un rendimiento ciertamente alto, pero más importante aún es destacar que con la metodología desarrollada en este trabajo queda margen para augmentar la eficiencia de la RN hasta lograr una eficiencia tan alta como RN que se usan profesionalmente en distintos ámbitos, las cuales pueden llegar a unos máximos de entre 92% y 94% de eficiencia.

También se pueden observar particularidades como que la RN de tamaño JH-25% es la que da unos resultados más predecibles ya que parece que la curva de eficiencia no presenta oscilaciones o ruido como las otras dos y tiene una forma más suave que el resto. El hecho de añadir perceptrones a la capa oculta tiene tendencia, por lógica, a augmentar la eficiencia de la RN, pero como se puede ver no siempre que se aumente la capa oculta significará una eficiencia mayor. Por ejemplo, en la gráfica se puede observar como para diversos puntos el modelo de red JH tiene una eficiencia mayor a la JH+25% pese a tener un tamaño menor, lo que nos dice que las reglas que recogió Jeff Heaton tienen cierto nivel de optimización resultando en una red bastante bien dimensionada.

4.4.1.3. Comparativa de tiempos de computación

Como se ha podido observar, el hecho de augmentar el número de perceptrones en las capas (tanto la de entrada a causa del tamaño de compresión, como la oculta variando según la regla de JH) da como resultado mejores RN. Este aspecto, tal y como se ha comentado anteriormente, sostiene un compromiso entre la eficiencia de la red y su complejidad. Como es deducible, estas redes muy complejas se tardarán mucho en crear y entrenar, y dependiendo de la aplicación puede ser un factor relevante el coste en tiempo y computación. Para poder hacernos a la idea de este coste se ha contabilizado el tiempo que se ha tardado en obtener cada RN, estos se muestran en la Figura 16.

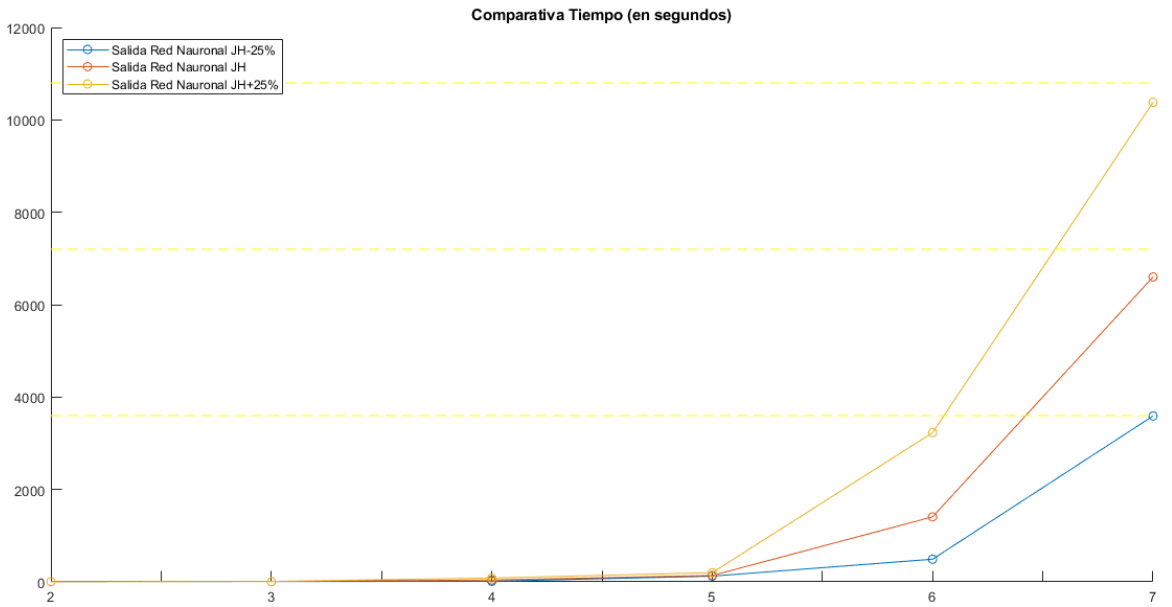

Figura 16: Comparativa de tiempo de entrenamiento para las distintas RN. El eje horizontal representa el tamaño de compresión y el tiempo en segundos. Las secciones separadas por líneas punteadas amarillas horizontales representan una hora de tiempo (3600 segundos).

Con esta gráfica sí que se puede ver es coste del augmento de perceptrones en las capas. Dado que el aumento de tiempo de computación crece a modo exponencial, nuestra RN de compresión 7x7 y con capa oculta JH+25% tardará casi 3 horas en entrenarse con 20.000 muestras, lo que no se refleja de manera tan diferencial al comparar eficiencias con sus redes homónimas, lo que indica que la eficiencia no crece proporcionalmente a las diferencias de tiempo que se producen entre los entrenamientos de distintos tipos de RN.

Si aplicásemos una regresión polinómica a estas curvas obtenidas experimentalmente podríamos predecir con cierta precisión la eficiencia y tiempos de computación de RN mayores. Con esta información se puede decidir a qué eficiencia queremos que llegue nuestra RN y cuánto tiempo dedicaremos a la computación y entreno de dicha RN. Estos datos pueden ser útiles al querer generar un número amplio de RN para la identificación de trazas de distintas fuentes, ajustando un compromiso adecuado entre el tiempo de computación y la eficiencia deseada en dicha red.

5. RESULTADO FINAL Y EXTENSIÓN DEL PROYECTO

Con todo el espectro de RN ya estudiado, en este apartado se determinará cuál es la red más adecuada a elegir. Una vez expuesto este resultado se aplicará la misma metodología para generar una RN esta vez partiendo de los datos obtenidos empíricamente.
Para finalizar se comentarán brevemente posibles maneras de extender el proyecto y aumentar la eficiencia de la RN.

5.1. Mejor resultado obtenido

El resultado más óptimo entre los que se tienen es el que se obtiene generando una RN tipo "Fitting", con compresión de 7x7 y tamaño de capa oculta JH.
Esta RN es la que más eficiencia tiene de las que se han creado, llegando a un 90,275% para una base de datos de test de 2000 trazas RTN.
Asimismo, cabe destacar que, si siguiésemos tratando de ampliar la eficiencia de la RN añadiendo más niveles de compresión, en la red tipo JH el aumento en el tiempo de computación para producirla es sustancialmente más bajo que comparado con las JH+25%. Dado que el comportamiento del tiempo necesario crece exponencialmente, tener una tendencia de augmento relativamente controlada puede ser un factor práctico para tener en cuenta.

5.2. RN con datos empíricos

En los primeros apartados del documento se menciona una base de trazas RTN reales obtenidas en el laboratorio, como se comenta en dicho apartado esta base se compone de 2520 trazas de 25.000 puntos divididas en 5 grupos procedentes de distintas fuentes de 504 trazas cada uno. Dado que generar una RN de este tipo requiere una cierta similitud en los parámetros de caracterización de las trazas RTN, de ahí que hayamos fijado unos parámetros en nuestro simulador de trazas RTN, no podemos usar los datos de entrenamiento de distintas fuentes, solamente de una de ellas, lo que se restringe la base de datos de entrenamiento a 504 trazas RTN. Esta restricción provocará por consecuente que la RN generada a partir de estos datos tenga un rendimiento bastante pobre (o como mínimo poco contrastado) a causa del corto tamaño de la base de entrenamiento. A recalcar que para el desarrollo de una RN con un 90% de eficiencia se ha usado una base de datos con 20.000 trazas de la misma longitud que las experimentales. Aun así, la simple comprobación de la posibilidad de generar una RN que funcione medianamente bien a partir de datos empíricos tiene cierto valor académico.
Así como en los datos obtenidos para simular trazas, la mejor fuente a escoger es una que tenga un offset distribuido lo más normalmente posible, esto se puede comprobar con el Gráfico Q-Q como ya se ha hecho anteriormente. Escogemos las 504 trazas provenientes de esta fuente.

Como se ha decidido partiendo de los resultados del proyecto la RN que generemos tendrá una compresión de 7x7 y un tamaño en la capa oculta de JH (por tanto 35 perceptrones en este caso).

De forma homónima al resto de RN generadas, partiendo de esta base de datos habrá que aplicarle una compresión para obtener les datos de entrada a la RN; y validarlas con el Criterio de Aceptación o Rechazo, obteniendo las salidas de la RN.

Para observar las diferencias entre la base de datos experimental y la simulada, así como estudiar el comportamiento de las trazas empíricas en nuestro sistema, en la Figura 17 se muestra el resultado de la validación de las 504 trazas RTN.

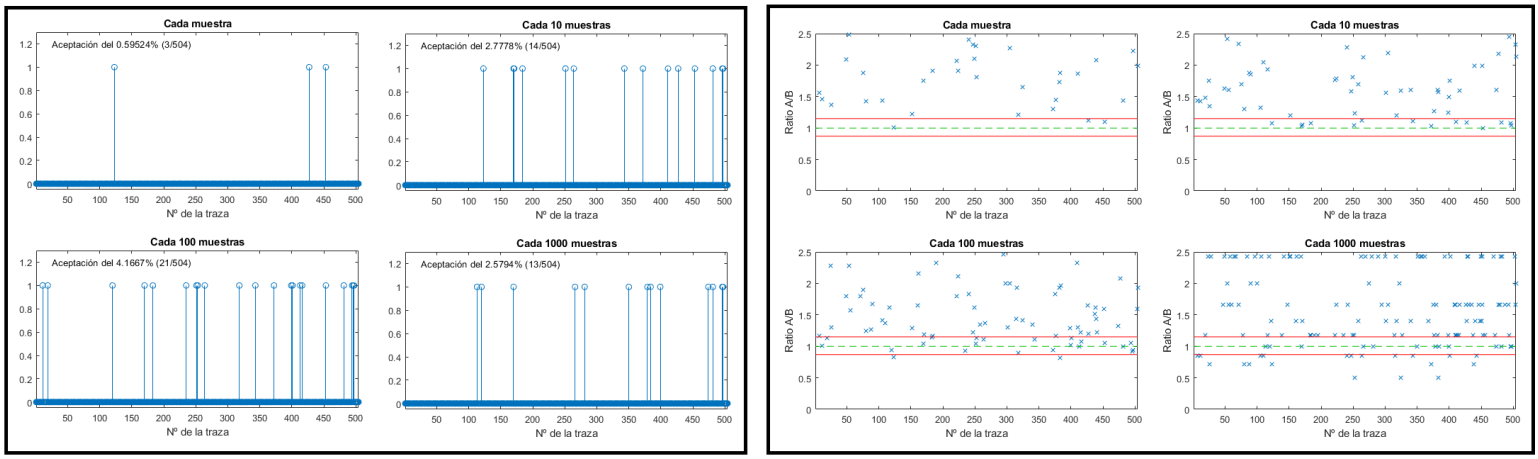

Figura 17: Análisis de la validación de la base de trazas experimentales. La sección izquierda de la figura muestra si la traza ha sido aceptada o no, y la parte derecha su correspondiente relación A/B y su posicionamiento en el rango de validez.

El primer rasgo que destacar es que el porcentaje de aceptación de las trazas es mucho menor a las trazas simuladas. Obviamente al ser trazas reales son algo más complejas y han experimentado más ruido que las trazas simuladas, cosa que hace que su categorización para la validez de la traza parezca más restrictiva.
Otro detalle es que en estas trazas el mejor muestreo para obtener la mayor incertidumbre en la repetibilidad de signos consecutivos en este caso es cada 100 muestras en lugar de cada 10 como en los datos simulados.
Se podrían destacar otros detalles, pero dado que la base es de un tamaño tan pequeño no se pueden extraer conclusiones muy veraces, solo posibles especulaciones basadas en la observación.
Una vez se tienen los datos de compresión y validación se genera y entrena la RN de 7x7 con capa oculta tamaño JH.
No tiene coherencia testear la RN con los datos con las que se han entrenado, por lo que no se puede hacer una comprobación a mano muy veraz de la eficiencia de la RN obtenida, pero MatLab en su proceso de entrenamiento de RN destina unas pocas muestras al testeo de la RN que está entrenando, lo que sí que puede indicarnos ciertas características sobre su comportamiento.

A continuación, en la Figura 18, se muestran los resultados del análisis de la Regresión y el Histograma de Error en el entrenamiento y test de la RN generada.

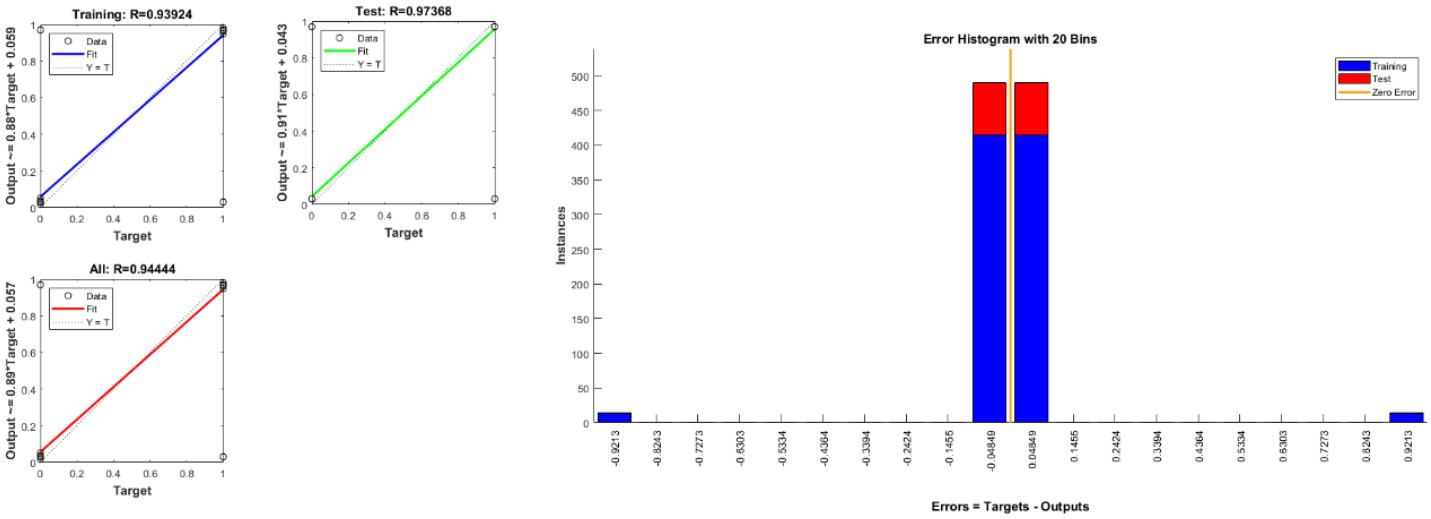

Figura 18: Regresión (a la izquierda) e Histograma de Errores (a la derecha) de la RN con datos empíricos.

Como se observa en la casilla para todos los datos del análisis de regresión, la de la línea roja, el rendimiento total de la RN obtenida es del 94.444%. El histograma de error respalda los datos de la regresión ya que se puede ver que la mayoría de los errores están muy cerca del punto correcto, es decir, hay muy poca dispersión en el histograma, lo que sugiere que los errores nunca son muy grandes, excepto algún caso puntual.

Este rendimiento probablemente solo sea consecuencia de una serie de buenas coincidencias en las características de las trazas le la base de datos escogida, ya que un rendimiento tan alto se debe a causa de que no hay suficientes trazas para analizar con precisión el rendimiento real de la RN, posiblemente con más datos este rendimiento de la RN bajaría un poco.

Aun así, pese a no ser los datos mejor contrastados, el altísimo rendimiento obtenido confirma que el proceso de compresión y validación de datos desarrollado en este proyecto es realmente útil para la generación de RN de alto rendimiento con datos empíricos, y con más datos y recursos se podría llegar a obtener una RN con aplicaciones en distintos campos de estudio y ámbitos profesionales.

5.3. Posibles mejoras y ampliaciones

Como ya se ha comentado, este proyecto está sesgado por el hecho de ser un trabajo con finalidades puramente académicas. Pese a ello, hay un amplísimo abanico de modificaciones y adiciones posibles con las que extender el correcto desarrollo de esta tipología de RN.

Ya que no es materialmente posible explorar estas opciones, en este apartado se listan y comentan brevemente algunas posibles líneas de desarrollo por explorar.

5.3.1. Aumento del tiempo y capacidad de computación

La ampliación más sencilla que realizar es simplemente disponer de más tiempo y recursos. La RN más larga de producir han sido cerca de 3h. Ésta ha sido generada en un ordenador personal con una capacidad de procesado adecuada para su función, pero no puede compararse a la tecnología y recursos de los que dispondría una empresa o institución de investigación.

A la práctica, cuando se producen RN con fines profesionales o de alto nivel de estudio, éstas se realizan con duraciones de días, incluso semanas.
Si dispusiésemos de dichos recursos las ampliaciones axiomáticas de este proyecto serían las dos siguientes:

5.3.2. Aumento del tamaño de las capas

El aumento del tiempo y capacidad de computación nos daría la posibilidad de crear RN con una cantidad mucho mayor de perceptrones en la capa de entrada y la capa oculta, puesto que la capa de salida seguiría fija ya que debe atender a las condiciones deseadas en la señal de salida. En cambio, el tamaño de la capa de entrada va relacionado con la discretización del histograma bivariado al comprimir los datos de entrada. Como se ha podido comprobar en diversas gráficas, el augmento de esta compresión genera más datos, lo que significa más precisión para la RN y por tanto mayor eficiencia de acierto. El tamaño máximo de compresión que se ha realizado ha sido de 7x7, y teniendo en cuenta que había aun margen de mejora ya que no se repetían resultados al aumentar la compresión. Una RN de por ejemplo de 10x10, aunque se tarde días en entrenar, tendría sin duda mejor eficiencia que la de 7x7.
Para finalizar, la capa oculta va acorde al tamaño de las otras capas por lo que el aumento del tamaño de la capa entrada también producirá un aumento en el tamaño de la capa oculta. Como se ha explicado, se ha seguido uno de los métodos que expone Jeff Heaton y lo hemos variado en cierto rango. Seguir estudiando la mejor calibración para el número de perceptrones en la capa oculta también podría dar lugar a un aumento en la eficiencia final de la RN.

5.3.3. Aumento del tamaño de datos de entreno

Contra más compleja es una RN y difícil la tarea que debe realizar, más muestras son requeridas para entrenarla adecuadamente. Por lógica, si una tarea es más compleja, lo que involucra más variables y más precisión en su tratamiento, se necesitarán más ejemplos o muestras para calibrar correctamente los pesos de la capa oculta.
Si se aumentase el tamaño de las capas, serían necesarias bases de muestras mayores para cubrir la mayoría de los casos posibles (es decir, de tipos de trazas válidas o no) que se pueden dar.
La base actual de 20.000 trazas RTN de 25.000 puntos se tardó alrededor de una hora y media para realizarse. A diferencia del caso de la compresión de datos de entrada de la RN, aumentar el tamaño de la base de trazas RTN no significará un augmento exponencial en el tiempo, sino lineal. Aplicando una sencilla regla de tres, el simulador de trazas RTN de este proyecto tiene una capacidad de producción, en el PC en que se ha realizado, de alrededor de 13.300 trazas/h, que significa unas 222 trazas/min de longitud de 25.000 puntos. Con estos datos es muy sencillo calcular cuánto tiempo requerirá generar una base de dato el tamaño que se desee, incluso hacer diversas pruebas de cuál es el tamaño a partir del cual la RN ya no mejora.

5.3.4. Reajuste del Criterio de Aceptación y sus límites

Dado que los criterios de aceptación de la traza son un aspecto del proyecto que requiere cierta experiencia y mucho conocimiento del campo para hacerse lo más adecuados posibles, con mucha seguridad hay margen de mejora en ese aspecto.

Si se hiciese siguiendo la misma metodología que la usada aquí, se podría hacer un estudio del rendimiento de la RN reajustando el valor de los límites de aceptación, o adaptándolos a unas necesidades específicas para una aplicación concreta de la RN.
De no querer seguir este método, con mucha seguridad hay otro modelo matemático válido que discrimine correctamente las trazas válidas.

5.3.5. Recorte de posibles datos irrelevantes

Otro posible enfoque podría ser el de examinar si hay cierta redundancia o sobreextensión de los datos en ciertos puntos del desarrollo. Hacer pruebas como el acortamiento de la longitud de las trazas o el estudio en la forma de los Time-Lag Plots, los cuales podrían contener áreas en las que nunca se vaya a producir un evento por lo que sería irrelevante contemplar esos datos en el histograma bivariado, podría dar lugar un sistema un poco más eficiente en el tratamiento y gestión de los datos usados en el desarrollo de las redes.

5.3.6. Análisis de tendencias y estimación de resultados

Para finalizar, a los datos que se han obtenido a partir del test de las RN podría aplicárseles distintas técnicas estadísticas como regresiones polinómicas para estimar la tendencia que sigue el aumento de eficiencia al hacer RN mayores. Estos datos podrían ilustrarnos sobre hasta qué punto se puede exprimir esta metodología y generar redes mejores, así como el tiempo y recursos que requerirán para obtenerse.

6. CONCLUSIONES

En este documento se desarrolla y aplica una metodología con la que se obtiene una Red Neuronal capaz de discernir trazas RTN con alto nivel de incertidumbre en símbolos binarios consecutivos. El uso de las trazas obtenidas estaría destinado a crear cadenas de bits aleatorios usados para la encriptación.

Esta metodología parte del estudio y extrapolación de datos referentes a trazas RTN reales con los que, a través de un simulador de trazas diseñado a lo largo del proyecto, podemos replicar dichas trazas.

Una vez obtenido un sistema generador de trazas se ha creado una base de datos formada por 20.000 trazas RTN destinadas a entrenar la Red Neuronal. A esta base se le aplica un criterio de validación desarrollado específicamente para este proyecto, con esta validación se obtiene el conjunto de salidas objetivas con la que entrenar a la red. También se le aplica un sistema de compresión o sinterización a la base de datos obteniendo así las entradas de entrenamiento para la red.

Teniendo las entradas y salidas para las Redes Neuronales se han generado distintas tipologías de redes basadas en diferentes premisas y condiciones, estas redes han sido comparadas entre ellas para obtener la configuración más óptima posible.

Finalmente se ha aplicado esta misma metodología a un conjunto de trazas RTN empíricas y se ha diseñado una Red Neuronal con ellas. Se ha finalizado el proyecto con la comprobación de la eficiencia de dicha red y el análisis de posibles extensiones del proyecto.

REFERENCIAS

[1] C.J. Tablada, G.A. Torres. 2009. Redes Neuronales Artificiales. Revista de Educación Matemática. 24, 3 (oct. 2009). DOI: https://doi.org/10.33044/revem.10280.

[2] A. Espin Sanz. 2020. RTN Characterization in stressed ultra-scaled CMOS devices. Master Thesis. Departament d'Enginyeria Electrònica, Universitat Autònoma de Barcelona.

[3] J. Martin-Martinez, R. Rodriguez, M. Nafria, G. Torrens, S.A. Bota, J. Segura, F. Moll A. Rubio. 2017. Statistical characterization and modeling of random telegraph noise effects in 65nm SRAMs cells. 2017 14th International Conference on Synthesis, Modeling, Analysis and Simulation Methods and Applications to Circuit Design (SMACD). IEEE Transactions on Signal Processing. DOI: 10.1109/SMACD.2017.7981610

[4] P. Saraza-Canflanca, J. Martin-Martinez, R. Castro-Lopez, E. Roca, R. Rodriguez, F. V. Fernandez, M. Nafria. 2021. Statistical Characterization of Time-Dependent Variability Defects Using the Maximum Current Fluctuation. IEEE Transactions on Electron Devices. DOI: 10.1109/TED.2021.3086448.

[5] J. Heaton. 2008. Introduction to Neural Networks for Java, 2nd Edition (2nd. ed.). Heaton Research, Inc.

[6] J. Lopez, E. F. Caicedo Bravo. 2005. Entrenamiento Bayesiano para Redes Neuronales Artificiales. Escuela de Ingeniería Eléctrica y Electrónica. Grupo de Percepción y Sistemas Inteligentes, Universidad del Valle, Colombia.

ANEXOS

ANEXO 1: Códigos de MatLab

- SIMULADOR DE RTN:

```
%% SIMULADOR DE TRAZAS RTN                                  Manel Marín – TFG

% Este código genera trazas RTN a partir de datos generados aleatoriamente
% según diversas distribuciones parametrizadas manualmente. A la salida se
% obtiene una matriz de trazas RTN generada a partir de los parámetros
% introducidos llamada 'MatrizTrazas'.
% Para su uso rellenar todas variables de los parámetros y ejecutar.

Close all
clear all
warning off

%% PARÁMETROS DE SIMULACIÓN


% PARAMETROS GENERALES

    % N° TOTAL DE TRAZAS
    Trazas = 10;


    % LONGITUD DE LA TRAZA (número de muestras por traza)
    Longitud = 25000;

    % SELECCIÓN DEL MODO DE IMPRESIÓN
    Imprimir = 'no';      % Escribir 'si' o 'no'

  % Modo = 'figure';   % Impresión en diversas figuras por separado
    Modo = 'holdon';   % Impresión en una sola figura con leyenda


% PARÁMETROS ESPECÍFICOS DE LAS TRAZAS

    % OFFSET – Distribución LogNormal
    % Parámetros:
    % - Media: OffsetMean
    % - Varianza: OffsetDevitaion
    OffsetMean = -5.34188;
    OffsetDeviation = 0.067;

    % N° DE DEFECTOS – Distribución de Poisson
    % Parámetro:
    % - Parámetro de media y varianza: Lambda
    Lambda = 2;

    % SALTO – Distribución Exponencial
    % Parámetro:
    % - Media: Mu
    Mu = 2E-7;

    % PROB. DE EMISIÓN Y CAPTURA – Distribución Aleatoria

    % RUIDO – Distribución Normal
    % Parámetros:
    % - Media: NoiseMean
    % - Varianza: NoiseDevitaion
    NoiseMean = 0;
```

```
    NoiseDeviation = 1.2E-8;

    % ESTADO INICIAL - Función dependiente de las prob. Emis/Capt.


% Bucle generador de los parámetros de las trazas
for k=1:Trazas

    Offset(1,k) = -(10^(normrnd(OffsetMean,OffsetDeviation)));

    Defectos(1,k) = poissrnd(Lambda);


    for j=1:Defectos(1,k)

        Salto(j,k) = exprnd(Mu);

        Pemision(j,k) = rand;
        Pcaptura(j,k) = rand;

        auxEstado1 = 1/(1+Pemision(j,k)/Pcaptura(j,k));
        auxEstado2 = rand;
        if auxEstado2<auxEstado1
            Estado_inicial(j,k) = 1;
        else
            Estado_inicial(j,k) = 0;
        end
    end

    if Defectos(1,k)==0   % Condición para cuando no hay defectos

        Salto(j,k) = 0;

        Pemision(j,k) = 0;
        Pcaptura(j,k) = 0;

        Estado_inicial(j,k) = 0;
    end
end


%% FASE DE GENERACIÓN Y ALMACENAMIENTO DE LAS TRAZAS


% Inicialización de la matriz de resultados
MatrizTrazas = zeros(Longitud,Trazas);

for k=1:Trazas      % Bucle para cada traza compuesta

    OutputVect = zeros(Longitud,Defectos(1,k)); % Inicializaciones de
    auxOutputVect = zeros(Longitud,1);          % auxiliares

    for j=1:Defectos(1,k)      % Bucle para cada traza simple
                               % no entrará si hay 0 defectos en la traza
        Output = zeros(Longitud,1);

        Estado = Estado_inicial(j,k); % Estado inicial de cada defecto

        for i=1:Longitud   % Bucle para cada punto de la traza simple

             if Estado==0        % Posibles cambios de estado
                aux = rand;
                if aux<Pcaptura(j,k)
                    Estado = 1;
                end
             else
                aux = rand;
                if aux<Pemision(j,k)
                    Estado = 0;
                end
             end
```

```
                    % Cálculo y registro del punto actual de la traza
                Output(i,1) = - Estado*Salto(j,k);

            end

            OutputVect(:,j) = Output(:,1);    % Almacenamiento trazas simples

        end

        % Generador de trazas compuestas por 50ondición50ión de trazas simples
        auxOutputVect = Offset(1,k);
        for r=1:Defectos(1,k)
            auxOutputVect = auxOutputVect + OutputVect(:,r);
        end

        % Condición técnica para cuando no hay defectos en la traza
        if auxOutputVect(1,1)==0
           auxOutputVect(:,1) = Offset(1,k);
        end

        % Almacenamiento de la traza compuesta en la matriz final
        MatrizTrazas(:,k) = auxOutputVect;

    end

    % Bucle para añadir el ruido de la traza compuesta
    for m=1:Trazas
        for n=1:Longitud
            MatrizTrazas(n,m) = MatrizTrazas(n,m) + normrnd(NoiseMean,NoiseDeviation);
        end
    end

    %% ETAPA DE IMPRESIÓN Y GRAFISMOS

    aux1 = 'si';      % 50ondición de impresión
    aux2 = strcmp(Imprimir,aux1);

    if aux2==1      % Condición de entrada a la fase de impresión

        aux3 = 'figure';       % Condición para el tipo de impresión
        aux4 = strcmp(Modo,aux3);
        t = 1:Longitud;           % Vector auxiliar de tiempo

        if aux4==1      % Caso 'figure'

            for s=1:Trazas
                figure
                plot(t,MatrizTrazas(:,s))

                if Defectos(1,s)==1  % Distinción caso título plural/singular
                    txt = ['Traza N° ',num2str(s) ', ' num2str(Defectos(1,s)) ' defecto'];
                    title(txt)
                else
                    txt = ['Traza N° ',num2str(s) ', ' num2str(Defectos(1,s)) ' defectos'];
                    title(txt)
                end
            end
        end

        if aux4==0      % Caso 'holdon'

            hold on
            for s=1:Trazas
                txt = ['Traza N° ',num2str(s)];
                plot(t,MatrizTrazas(:,s),'DisplayName',txt)
            end
            hold off
            legend show

        end
    end
```

- VALIDADOR DE TRAZAS:

```
%% VALIDADOR DE TRAZAS
% Alerta: Trazas de un mínimo de 1001 muestras de Longitud

% Este código genera una matriz de aceptación o rechazo (1 o 0) de una
% matriz de trazas siguiendo un criterio de aceptación de cada traza
% basado en un rango alrededor de una autocorrelación equilibrada entre el
% nombre de símbolos consecutivos que se repiten o que cambian.
% La matriz de salida contiene 4 vectores en los que cada muestra de la
% autocorrelación se toma cada 1, 10, 100 y 1000 puntos respectivamente.

% SELECCIÓN DEL MODO DE IMPRESIÓN
Imprimir = 'si';       % Escribir 'si' o 'no'

% LÍMITES DE ACEPTACIÓN
Optimo = 1;              % Valor óptimo
UpLimit = 1.15;          % Límite superior
DownLimit = 0.87;        % Límite inferior

% Cálculo de variables auxiliares
Size2 = size(MatrizTrazas);
Trazas = Size2(2);
Longitud = Size2(1);

A1 = zeros(1,Trazas);    B1 = zeros(1,Trazas);
A10 = zeros(1,Trazas);   B10 = zeros(1,Trazas);
A100 = zeros(1,Trazas);  B100 = zeros(1,Trazas);
A1000 = zeros(1,Trazas); B1000 = zeros(1,Trazas);


% CONVERSOR DE TRAZAS RTN A VALORES BINARIOS
for j=1:Trazas
    Media(1,j) = mean(MatrizTrazas(:,j));  % Media de cada traza

    for i=1:Longitud
        if MatrizTrazas(i,j)>Media(1,j)
            MatrizBinaria(i,j) = 1;
        else
            MatrizBinaria(i,j) = 0;
        end
    end
end


% CÁLCULO DE LA AUTOCORRELACIÓN
for j=1:Trazas              % Cada Muestra
    for i=1:1:Longitud-1
        if MatrizBinaria(i,j)==MatrizBinaria(i+1,j)
            A1(1,j) = A1(1,j)+1;
        else
            B1(1,j) = B1(1,j)+1;
        end
    end

    MatrizCriterios(1,j) = A1(1,j)/B1(1,j);
    if MatrizCriterios(1,j)>DownLimit && MatrizCriterios(1,j)<UpLimit
        MatrizAceptaciones(1,j) = 1;
    else
        MatrizAceptaciones(1,j) = 0;
    end
end

for j=1:Trazas              % Cada 10 Muestras
    for i=1:10:Longitud-10
        if MatrizBinaria(i,j)==MatrizBinaria(i+10,j)
            A10(1,j) = A10(1,j)+1;
        else
            B10(1,j) = B10(1,j)+1;
        end
    end

    MatrizCriterios(2,j) = A10(1,j)/B10(1,j);
```

```
    if MatrizCriterios(2,j)>DownLimit && MatrizCriterios(2,j)<UpLimit
        MatrizAceptaciones(2,j) = 1;
    else
        MatrizAceptaciones(2,j) = 0;
    end
end

for j=1:Trazas              % Cada 100 Muestras
    for i=1:100:Longitud-100
        if MatrizBinaria(i,j)==MatrizBinaria(i+100,j)
            A100(1,j) = A100(1,j)+1;
        else
            B100(1,j) = B100(1,j)+1;
        end
    end

    MatrizCriterios(3,j) = A100(1,j)/B100(1,j);
    if MatrizCriterios(3,j)>DownLimit && MatrizCriterios(3,j)<UpLimit
        MatrizAceptaciones(3,j) = 1;
    else
        MatrizAceptaciones(3,j) = 0;
    end
end

for j=1:Trazas              % Cada 1000 Muestras
    for i=1:1000:Longitud-1000
        if MatrizBinaria(i,j)==MatrizBinaria(i+1000,j)
            A1000(1,j) = A1000(1,j)+1;
        else
            B1000(1,j) = B1000(1,j)+1;
        end
    end

    MatrizCriterios(4,j) = A1000(1,j)/B1000(1,j);
    if MatrizCriterios(4,j)>DownLimit && MatrizCriterios(4,j)<UpLimit
        MatrizAceptaciones(4,j) = 1;
    else
        MatrizAceptaciones(4,j) = 0;
    end
end

%% ETAPA DE IMPRESIÓN Y GRAFISMOS

Titulos = {'Cada muestra','Cada 10 muestras','Cada 100 muestras','Cada 1000 muestras'};

aux1 = 'si';     % Condicion de impresión
aux2 = strcmp(Imprimir,aux1);

if aux2==1     % Condición de entrada a la fase de impresión

    for k=1:4
        subplot(2,2,k);
        plot(MatrizCriterios(k,:),'x')
        xlabel('N° de la traza')
        ylabel('Ratio A/B')
        title(string(Titulos(k)))
        axis([1 Trazas 0 2.5])
        hold on
        plot(xlim, [1 1]*Optimo,'color',[0 0.75 0],'linestyle','--')
        plot(xlim, [1 1]*UpLimit,'r')
        plot(xlim, [1 1]*DownLimit,'r')
        hold off
    end
    figure
    for k=1:4
        subplot(2,2,k);
        stem(MatrizAceptaciones(k,:))
        xlabel('N° de la traza')
        title(string(Titulos(k)))
        txt = ['Aceptación del
',num2str((sum(MatrizAceptaciones(k,:))/length(MatrizAceptaciones(k,:)))*100) '%
','(',num2str(sum(MatrizAceptaciones(k,:))),'/',num2str(length(MatrizAceptaciones(k,:)))
')'];
        text(0.05*Trazas,1.2,txt)
        axis([1 Trazas -0.05 1.3])
```

```
    end
end
```

- SINTETIZADOR DE DATOS:

```
%% SINTETIZADOR DE DATOS

% Convierte una matriz de trazas RTN en una matriz de vectores generados
% por concatenación del resultado de un time-lag plot de cada traza, los
% vectores resultantes tienen un tamaño de N^2 donde N es el tamaño del
% lado de la matriz cuadrada al dividir el time-lag plot (entradas de la
% vector deseado (en nuestro caso el de 10 en 10) de la matriz de
% RN); y elige el aceptaciones y le añade una salida negada (salidas de
% la RN).
% A la salida se obtiene una matrix de los datos sintetizados de las trazas
% y un vector de una matriz de aceptaciones con la salida real y la salida
% complementaria negada.

% SELECCIÓN DEL MODO DE IMPRESIÓN
Imprimir = 'no';     % Escribir 'si' o 'no'

% TAMAÑO DE LOS VECTORES DE SALIDA (número natural al cuadrado)
SQRTInputs = 7;      % Raíz cuadrada de las entradas (número natural)

% GENERADOR DE LOS DATOS SINTETIZADOS

% Time-Lag y Histograma
Size2 = size(MatrizTrazas);
Trazas = Size2(2);
Longitud = Size2(1);

for j=1:Trazas

    auxVectHist3(:,1) = abs(MatrizTrazas(1:Longitud-1,j));
    auxVectHist3(:,2) = abs(MatrizTrazas(2:Longitud,j));

    auxMatrizHist3Norm = hist3(auxVectHist3,[SQRTInputs SQRTInputs])/Longitud;

    cont = 1;
    for n=1:SQRTInputs
        VectHist3Norm(cont:n*SQRTInputs,1) = auxMatrizHist3Norm(:,n);
        cont = cont+SQRTInputs;
    end

    MatrizHist3Norm(:,j) = VectHist3Norm;

end


% Salida negada
VectorAceptaciones(1,:) = MatrizAceptaciones(2,:);

for i=1:Trazas
    if VectorAceptaciones(1,i)==1
        VectorAceptaciones(2,i)=0;
    else
        VectorAceptaciones(2,i)=1;
    end
end

%% ETAPA DE IMPRESIÓN Y GRAFISMOS

aux1 = 'si';     % Condicion de impresión
aux2 = strcmp(Imprimir,aux1);

if aux2==1     % Condición de entrada a la fase de impresión
    for j=1:Trazas
```

```
            auxPlotVectHist3(:,1) = abs(MatrizTrazas(1:Longitud-1,j));
            auxPlotVectHist3(:,2) = abs(MatrizTrazas(2:Longitud,j));

            figure

            subplot(1,2,1);
            hist3(auxPlotVectHist3,[SQRTInputs SQRTInputs],'CdataMode','auto');
            xlabel('Estado Traza')
            ylabel('Estado Traza +1')
            colorbar

            subplot(1,2,2);
            hist3(auxPlotVectHist3,[SQRTInputs SQRTInputs],'CdataMode','auto');
            xlabel('Estado Traza')
            ylabel('Estado Traza +1')
            colorbar
            view(2)

        end
    end
```

ANEXO 2: Especificaciones del PC usado

Información del sistema

Sistema operativo:	Windows 10 Pro 64 bits (10.0, compilación 19041)
Idioma:	español (configuración regional: español)
Fabricante del sistema:	Gigabyte Technology Co., Ltd.
Modelo del sistema:	To be filled by O.E.M.
BIOS:	F6
Procesador:	Intel(R) Core(TM) i5-6600 CPU @ 3.30GHz (4 CPUs), ~3.3GHz
Memoria:	8192MB RAM
Archivo de paginación:	6847MB usados, 4989MB disponibles
Versión de DirectX:	DirectX 12