Tecnológico Nacional de México
Instituto Tecnológico Superior de Purísima del Rincón

ESTUDIOS CON RECONOCIMIENTO DE VALIDÉZ OFICIAL

NÚMERO: 11EIT0006A

TITULACIÓN INTEGRAL

TIPO DE PROYECTO:

TESIS

TÍTULO:

INTEGRACIÓN Y DESARROLLO DE UN NUEVO MODELO DE PERCEPTRÓN MULTICAPA PARA LA CLASIFICACIÓN DE PATRONES NO LINEALES

PARA OBTENER EL GRADO DE

INGENIERÍA INFORMÁTICA

PRESENTA:

CARLOS FERNANDO OROZCO SOLIS

RS17110263

ASESOR:

DR. VALENTÍN CALZADA LEDESMA

PURÍSIMA DEL RINCÓN, GTO. MAYO 2022

DEDICATORIA

Esta tesis está dedicada a:

Para mi madre Yolanda que con su amor, paciencia y esfuerzo me ha permitido llegar a cumplir hoy un logro más, gracias por inculcar en mí el ejemplo de esfuerzo y valentía, de no rendirme nunca y luchar hasta lograr mis objetivos.

Mis hermanos Bryan y Edwin por su cariño y compañía, durante todo este proceso, por estar conmigo, alegrarme y por compartir buenos momentos, gracias.

A mi novia Sarahi, que de manera sincera e incondicional me mostro su amor y me impulso a seguir trabajando por mis metas, gracias por creer en mí y darme el empujón que necesitaba para avanzar y seguir creyendo en que los sueños se hacen realidad.

Finalmente quiero dedicar esta tesis a todos mis amigos que me acompañaron en este camino, por extender su mano en momentos difíciles, de verdad gracias a todos.

AGRADECIMIENTOS

Mi profundo agradecimiento a todas las autoridades y personal que hacen el Instituto Tecnológico Superior de Purísima del Rincón, por confiar en mí, abrirme las puertas y permitirme estudiar mi carrera universitaria.

De igual manera, mis agradecimientos a toda la carrera de Informática, a mis profesores, en especial al M.R.S.I. Diego Eduardo Morales López y al ISC. Luis Emmanuel Carreón Aranda quienes con la enseñanza de sus valiosos conocimientos hicieron que pueda crecer día a día como profesional, gracias a cada uno de ustedes por su paciencia, dedicación y apoyo.

Asimismo, quiero expresar mi más grande y sincero agradecimiento al Dr. Valentín Calzada Ledesma y al Dr. Juan De Anda Suárez, principal tutor y colaborador, bajo su dirección, conocimiento y enseñanza permitió el desarrollo de este trabajo. Gracias por su amabilidad para facilitarme su tiempo y sus ideas.

Finalmente, gracias a mi familia y mi novia, por su paciencia, comprensión y solidaridad con este proyecto, por el tiempo que me han concedido. Sin su apoyo este trabajo nunca se habría escrito y, por eso, este trabajo es también suyo.

A todos muchas gracias.

DECLARACIÓN DE ORIGINALIDAD

25 de mayo de 2022, Purísima del Rincón, Guanajuato, México.

Por medio de la presente yo, Carlos Fernando Orozco Solis en calidad de autor declaro y manifiesto que este documento de tesis es producto de mi trabajo original y no infringe los derechos de terceros, tales como derechos de publicación, derechos de autor, patentes y similares.

Además, declaro que en las citas textuales que he incluido y en los resúmenes que he realizado de publicaciones ajenas, indico explícitamente los datos de los autores y las publicaciones.

En caso de infracción de los derechos de terceros derivados de este documento de tesis, acepto la responsabilidad de infracción y relevo de ésta a mi director de tesis, así como al Instituto Tecnológico Superior de Purísima del Rincón y sus autoridades.

Atentamente

Carlos Fernando Orozco Solis

Nombre y firma

RESUMEN

En la actualidad, se utilizan innumerables sistemas de inteligencia artificial basados en Redes Neuronales Artificiales (RNAs) para resolver problemas como la clasificación, la detección de defectos, hacer predicciones, etc., sin embargo, cada problema tiene sus condiciones y restricciones, y dependiendo de la complejidad del mismo, el diseño de la arquitectura de una RNA puede variar, y en ocasiones, crecer desmesuradamente, lo que conduce a la maldición de la dimensionalidad (curse of dimensionality), lo cual implica un costo computacional elevado.

Por este motivo, en esta tesis se propone la integración y desarrollo de un nuevo modelo de perceptrón multicapa para resolver problemas con características no lineales, el cual permite crear arquitecturas compactas de RNAs, las cuales a su vez, disminuyen el costo computacional.

En el presente trabajo se realizó un diseño experimental para comparar empíricamente la propuesta contra una RNA tradicional, los resultados muestran que la RNA propuesta presenta un rendimiento similar (medido a través de la precisión en la exactitud) al de una RNA tradicional, en contraste, la propuesta de este trabajo de tesis permite el diseño de arquitecturas compactas, lo cual puede disminuir el costo computacional al lidiar con problemas cuya complejidad es mayor.

ABSTRACT

At present, countless artificial intelligence systems based on Artificial Neural Networks (RNAs) are used to solve problems such as classification, defect detection, predictions, etc., however, each problem has its conditions and restrictions, and depending on the complexity of it, the design of the architecture of an RNA can vary, and sometimes, grow excessively, which leads to the curse of dimensionality, which implies a high computational cost.

For this reason, this thesis proposes the integration and development of a new multilayer perceptron model to solve problems with non-linear characteristics, which allows the creation of compact architectures of RNAs, which in turn, reduce the computational cost.

In the present work an experimental design was made to empirically compare the proposal against a traditional RNA, the results show that the proposed RNA presents a similar performance (measured through precision in accuracy) to that of a traditional RNA, in contrast, the proposal of this thesis work allows the design of compact architectures, which can lower the computational cost when dealing with problems whose complexity is greater.

ÍNDICE DE CONTENIDO

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

CAPÍTULO I INTRODUCCIÓN

1.1 INTRODUCCIÓN

En la actualidad se utilizan innumerables sistemas de inteligencia artificial basados en Redes Neuronales Artificiales (RNAs) para resolver problemas de clasificación, de detección de defectos, de predicciones, etc., todos estos problemas se abordan de la misma manera, entre más complicado sea el problema, más compleja será la arquitectura de la RNA utilizada para resolverlo.

Las RNAs han sido ampliamente utilizadas en los últimos años, sin embargo, uno de los principales problemas de este enfoque es que, para cierto tipo de problemas, el costo computacional es elevado, esto se debe a la maldición de la dimensionalidad (curse of dimensionality), donde la arquitectura de la red tiende a crecer desmesuradamente. Esto no es un problema para las grandes industrias, ya que generalmente tienen a disposición hardware para realizar procesamiento, pero bajo un enfoque industrial comercial y local, donde todos quieren adaptarse y utilizar esta tecnología, esto se vuelve complicado por los recursos de hardware necesarios para utilizarse.

Por este motivo, en esta tesis se propone la integración y desarrollo de un nuevo modelo de perceptrón multicapa para resolver problemas con características no lineales, el cual permite crear arquitecturas compactas de RNAs, las cuales a su vez, disminuyen el costo computacional.

La metodología utilizada en este trabajo se dividió en tres partes, en la primera se muestra la deducción matemática de los algoritmos de una RNA y la modificación de estos, en la segunda se observa la implementación computacional de dichos algoritmos mediante diagramas, y en la última, se muestra una comparativa del rendimiento entre una RNA tradicional y la RNA propuesta.

Los resultados de esta comparativa muestran que el enfoque propuesto obtiene resultados similares a los de una RNA tradicional, pero logrando a su vez disminuir la arquitectura de la RNA (es decir, el número de capas y/o neuronas), para la resolución de problemas de separabilidad lineal o no lineal.

El análisis de estos resultados muestra empíricamente que se puede disminuir la complejidad de la arquitectura de una RNA.

1.2 ANTECEDENTES

En el estado del arte se pueden distinguir claramente tres generaciones de Redes Neuronales Artificiales (RNAs). La primera también llamada Perceptrón, basada en el modelo de (McCulloch & Walter, 1943), su rasgo característico es que solo pueden dar como salida un digito booleano (0 y 1). Sin embargo, cada función booleana se puede calcular en multicapas dentro en una sola capa oculta. La segunda generación se basa en el mismo modelo y amplia el comportamiento haciendo uso de una “función de activación” que se aplica a los valores de entrada, la más común es la función sigmoidea (Rumelhart & MacClelland, 1988). Por último, las de tercera generación también llamadas pulsantes, que mediante una descripción matemática modelan con más realismo las neuronas biológicas (Maass, 1997).

Los trabajos previos enfocados en la segunda generación se han centrado en encontrar el mejor diseño de la arquitectura de una Red Neuronal Artificial (RNA) para resolver determinados problemas. Esto implica que la arquitectura de una RNA crezca desmesuradamente ocasionando un coste computacional alto, claro ejemplo de esto son las redes de aprendizaje profundo, si bien éstas funcionan

adecuadamente para problemas como la clasificación de imágenes, implica el uso de hardware especializado para su correcta ejecución.

También se han identificado trabajos cuyo enfoque se centra en remplazar el mecanismo clásico de entrenamiento de una RNA llamado *Backpropagation*, por otro tipo de algoritmos, por ejemplo: algoritmos genéticos, optimización por enjambre de partículas, optimización por hormigas, algoritmos de estimación de distribución, entre otros. Esto con el fin de calibrar efectivamente los pesos del perceptrón multicapa independientemente del problema de entrada. Esto conduce a un problema de identificación del mejor algoritmo de calibración y un alto conocimiento en optimización computacional.

Otros trabajos se han centrado en determinar el mejor conjunto de características que describen el problema para que una RNA simple pueda resolverlo. Esto implica un análisis matemático minucioso del problema para conocer su geometría. Para problemas de baja dimensionalidad esto podría ser factible, sin embargo, para problemas de alta dimensionalidad el análisis matemático formal puede ser imposible humanamente hablando debido a la gran cantidad de variables.

Pocos trabajos se han centrado actualmente en modificar el modelo matemático actual del perceptrón multicapa, es por eso que, en este trabajo se realiza una propuesta para llevarlo a cabo.

1.3 DESCRIPCIÓN DEL PROBLEMA

El esquema tradicional de una red neuronal artificial nos dice que, para resolver problemas de separabilidad lineal o no lineal es necesario utilizar, dependiendo de la complejidad del problema, un gran número de capas en la arquitectura y cada una de estas contara con una gran cantidad de neuronas, lo que se traduce a un mayor coste computacional para la correcta clasificación de estos problemas.

1.4 JUSTIFICACIÓN

Hoy en día, con el esquema actual de una RNA se resuelven problemas de clasificación, detección y predicción, haciendo uso de arquitecturas neuronales cuyo procesamiento implica hardware muy sofisticado y costoso.

Actualmente, con el surgimiento del loT, gran parte del sector industrial hace uso de arquitecturas embebidas debido a su bajo costo en el mercado, lo negativo de utilizar estas herramientas es que no son capaces de soportar una gran carga computacional, por lo que las deja fuera de algunas aplicaciones de las RNAs.

En este trabajo de tesis se propone un nuevo esquema de RNA cuya unidad básica, el perceptrón, implementa en su núcleo una función generalizada para resolver problemas de separabilidad lineal o no lineal, esto con el fin de reducir el número de capas y neuronas que utilizaría el enfoque tradicional, disminuyendo el coste computacional.

1.5 HIPÓTESIS

Es posible diseñar una red neuronal artificial cuya unidad básica, el perceptrón, implemente una función generalizada que permita reducir la arquitectura de la red neuronal artificial, para resolver problemas de separabilidad lineal o no lineal con bajo coste computacional.

1.6 OBJETIVOS

- **General:** Proponer una red neuronal artificial cuya unidad básica, el perceptrón, implemente una función generalizada que permita resolver problemas de separabilidad lineal o no lineal con bajo coste computacional.

- **Específicos:**

1. Realizar la deducción matemática del nuevo enfoque de redes neuronales artificiales.
2. Modificar el algoritmo *Backpropagation.*
3. Implementar computacionalmente el algoritmo.
4. Generar *Datasets* sintéticos.
5. Realizar comparativa del esquema tradicional de RNAs y el esquema propuesto.
6. Reportar resultados obtenidos.

1.7 ESQUEMA METODOLÓGICO

En la Figura 1 se muestra un diagrama de la implementación del nuevo enfoque de la RNA donde se puede observar que la única información que se brinda a la RNA desde el exterior son los datos de entrada y la arquitectura de la red; la información restante es generada por la misma red. También se pueden resaltar, los elementos de color rojo (arquitectura de la red, entrenamiento de la RNA, *Feedforward*, *Backpropagation*) en estos es donde se encuentra una modificación en el enfoque tradicional para obtener el enfoque propuesto. La deducción matemática para estos cambios se puede ver en el Capítulo IV.

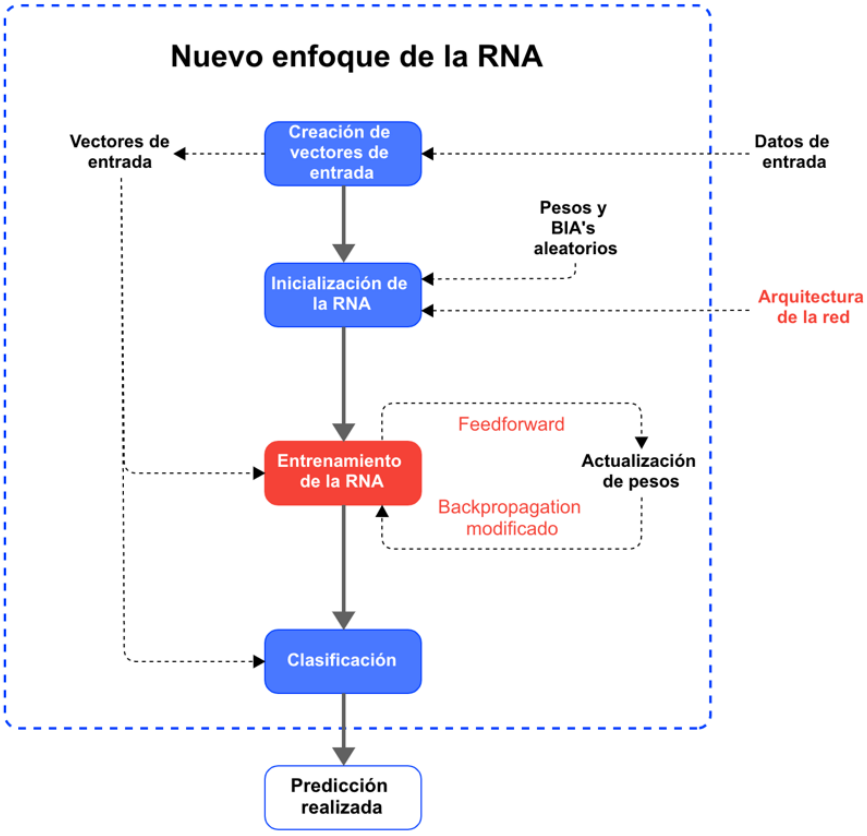

Figura 1. Diagrama de la implementación del nuevo enfoque de la RNA.

1.8 ESTRUCTURA DE LA TESIS

Este documento está dividido en siete capítulos donde se abarcan todos aspectos necesarios para el desarrollo del proyecto.

En este el primer capítulo, se analizan los antecedentes, la descripción del problema, la justificación, la hipótesis, el objetivo general y específicos, el esquema metodológico y la arquitectura de la tesis.

El estado del arte se muestra en el segundo capítulo, en este se mencionan algunos de los artículos científicos que trabajaron con distintos enfoques, para atacar la misma problemática que en este trabajo. Estos trabajos nos permiten comprender y conocer el entorno con respecto a la modificación de una Red Neuronal Artificial (RNA).

Para mejorar el entendimiento de este proyecto, en el marco teórico que se encuentra en el tercer capítulo, se mencionan todos los términos y temas que hay que tener en cuenta para comprender el desarrollo del mismo.

La modificación de los algoritmos de la RNA se muestra en el cuarto capitulo, en este se desglosa en tres secciones, en la primera se encuentra una explicación con ecuaciones de los conceptos (perceptrón simple, perceptrón multicapa, propagación hacia adelante, propagación hacia atrás), en la segunda sección se muestra la propuesta del nuevo enfoque, y en la última se puede observar el algoritmo de *Backpropagation* (BP) bajo un enfoque matricial.

En el quinto capitulo se muestra la implementación computacional, para esto se desarrollaron algunos diagramas, el primer diagrama es de casos de usos, éste muestra las actividades que son necesarias para la elaboración del proyecto. Los siguientes diagramas son de bloques, estos diagramas muestran el proceso para realizar de cada una de las actividades. Todos los diagramas mostrados en esta sección están codificados, este código se puede ver en el Anexo 1.

En el sexto capítulo, se realiza la comparativa empírica del rendimiento entre el esquema tradicional y el esquema propuesto, este capítulo está dividido en cuatro secciones, en la primer sección se muestra la creación de los *Datasets* que se utilizarán en la experimentación, en la segunda sección se realiza el diseño experimental, estos diseños son los que serán ejecutados y los resultados de los mismos se muestran en la tercer sección, en la última sección del capítulo se hace un análisis de los resultados.

Las conclusiones de este proyecto se muestran en el último capítulo, a partir de los resultados obtenidos y el análisis de estos, se concluye de manera general el desarrollo de este proyecto.

1.9 CONTRIBUCIONES

Durante la elaboración de este trabajo, se trabajó de manera simultánea en otro proyecto, este consistió en el desarrollo de un sistema de inteligencia artificial industrial sobre arquitecturas embebidas para identificar defectos en materiales de piel, con el fin de hacer más eficaz y eficiente el proceso de producción de productos en la industria del cuero y calzado.

Los resultados obtenidos al poner a prueba las predicciones hechas por el sistema de inteligencia artificial industrial fueron relativamente satisfactorios, considerando que para el entrenamiento de la red se utilizaron muy pocas imágenes, lo que reduce en gran medida la capacidad de clasificación.

Con el código realizado en la elaboración de este proyecto, se realizó el registro de software “Programa de cómputo para la identificación automática de defectos en materiales de cuero y piel” en el programa INDAUTOR.

También se participó en el programa “23 Verano de la Ciencia de la Región Centro”, donde se redactó un artículo del proyecto y se realizó una presentación del mismo.

Por último, fuimos invitados a presentar un prototipo del proyecto, en un evento internacional llamado “Summit Internacional de Vehículos Híbridos y Eléctricos” organizado por el gobierno de Guanajuato. Este prototipo consistía en mostrar predicciones realizadas con el sistema de detección de defectos, para esto se utilizaron algunas imágenes que tenían algunos defectos como cortes, manchas o líneas y el sistema tenía que señalar donde se encontraban estos defectos.

CAPÍTULO II ESTADO DEL ARTE

En 1992 se propuso un algoritmo rápido para entrenar perceptrones multicapa como alternativa al algoritmo de retro propagación de errores o *Backpropagation* (BP). Éste reducía el número de iteraciones requeridas en un 20% con respecto algoritmo de BP; para esto se utilizó un sistema de ecuaciones lineales en cada nodo (Scalero & Tepedelenlioglu, 1992).

Utilizando un modelo dinámico no lineal en un perceptrón, se logró mejorar el orden de magnitud en la velocidad de convergencia sobre un algoritmo de aprendizaje estático usado para entrenar la misma red. El desarrollo de este modelo obtuvo resultados muy efectivos a pesar de la presencia de ruido en el conjunto de entrenamiento y prueba (Parlos, et al., 1994).

Resolviendo el problema de clasificación de dos espirales se describió un método de entrenamiento para un clasificador de perceptrón multicapa, utilizando la idea de máquinas de vector soporte (SVM) en una sola capa oculta. Los pesos de salida se determinaron de acuerdo con el método SVM (Suyken & Vandewalle, 1999).

Se mejoró el rendimiento en la convergencia del algoritmo de entrenamiento BP, haciendo uso del algoritmo *The nearest centroid neighbourhood* que toma muestras de manera inteligente cerca del límite de decisión sin conocer realmente la posición de este límite (Chaudhuri & Bhattacharya, 2000).

Un nuevo enfoque para el proceso de aprendizaje de una red neuronal multicapa se presentó en 2001, este enfoque minimiza una forma modificada del criterio utilizado en el algoritmo de BP. Este criterio se basa en la suma de los errores cuadráticos lineales y no lineales de la neurona de salida. La elección del parámetro de diseño se evalúa mediante el análisis de series de convergencia de rangos y valores de error asintóticos constantes (Abid, et al., 2001).

Modificando el algoritmo de mínimos cuadrados recursivos (RLS), se presentó un nuevo algoritmo de aprendizaje para un perceptrón multicapa denominado mínimos cuadrados recursivos modificados (MRLS). El algoritmo MRLS difiere de RLS en la forma en que se estima el peso de las conexiones lineales para la red. los resultados de la comparación entre los dos algoritmos muestran una mejora significativa en el algoritmo MRLS (Subhi Al-Batah, et al., 2010).

Una nueva red llamada *Hybrid Multilayered Perceptron* (HMLP) se presentó para mejorar el rendimiento de una red de perceptrón multicapa (MLP), con esta nueva red se mejora la tasa de convergencia, se disminuye el número de nodos necesarios en una capa, lo que reduce el coste computacional que requiere él modelo MLP estándar (Mashor, 2010).

Con el fin de encontrar la estructura óptima para una Red Neuronal Artificial (RNA) se propuso un algoritmo basado en el análisis de sensibilidad de la varianza que elimina secuencialmente las neuronas ocultas, entradas y pesos. El criterio de parada se basa en una evaluación del rendimiento de los resultados de la red con los conjuntos de aprendizaje y validación (Thomas & Suhner, 2014).

Para entrenar un perceptrón multicapa se utilizó el algoritmo *Ant Lion Optimizer* (ALO), con el fin de encontrar los pesos y sesgos del MLP y lograr un error mínimo. Los resultados que se obtuvieron con el algoritmo ALO son muy competitivos ya que logro una alta tasa de precisión (Waleed, y otros, 2015).

Se presento un modelo de perceptrón multicapa modificado basado en árboles de decisión (DT-MLP), de acuerdo con los resultados, el valor del error cuadrático medio del modelo DT-MLP disminuye en un 9% (Gholami, et al., 2016).

CAPÍTULO III MARCO TEÓRICO

El marco teórico, que se desarrolla a continuación, describe todos los términos y conceptos para la realización de este proyecto, brindando así un panorama amplio permitiendo su entendimiento.

Se inicia con el concepto básico de una Red Neuronal Artificial (RNA), este es utilizado durante todo el documento. A partir de este concepto se desglosan más términos y definiciones, estas se describen a continuación.

3.1 DEFINICIÓN DE UNA RED NEURONAL ARTIFICIAL

En la literatura se pueden encontrar diferentes definiciones de lo que es una RNA, estas pueden ir desde lo más simple hasta las que detallan ampliamente su descripción. Veamos algunos ejemplos que podemos destacar.

"… un sistema de computación hecho por un gran número de elementos simples, elementos de proceso muy interconectados, los cuales procesan información por medio de su estado dinámico como respuesta a entradas externas" (Hecht-Nielsen, 1988).

"Redes neuronales artificiales son redes interconectadas masivamente en paralelo de elementos simples (usualmente adaptativos) y con organización jerárquica, las cuales intentan interactuar con los objetos del mundo real del mismo modo que lo hace el sistema nervioso biológico" (Kohonen, 1988).

En cada una de las definiciones se puede observar que una RNA es un sistema interconectado que recibe información, la procesa y retorna, matemáticamente se podría expresar como una serie de productos puntos, donde tenemos un vector de entrada que realiza una operación y cada resultado es un nuevo vector de entrada para la siguiente operación hasta obtener una salida.

3.2 VENTAJAS QUE OFRECEN LAS REDES NEURONALES ARTIFICIALES

Las redes neuronales presentan un gran número de características semejantes a las del cerebro. Estas son capaces de aprender de la experiencia a partir de casos anteriores, generalizan la información con el fin de extraer características esenciales a partir de entradas que representan información irrelevante. Esto hace que ofrezcan numerosas ventajas, entre estas incluyen (Hilera & Martínes, 1995).

- **Aprendizaje adaptativo.** A partir de un entrenamiento o una experiencia inicial tienen la capacidad de aprender a realizar tareas específicas.
- **Auto organización.** En la etapa de aprendizaje una RNA puede crear su propia organización o representación de la información.
- **Tolerancia a fallos.** La RNA tiene la capacidad de retener algunas de sus funciones a pesar de la destrucción parcial de esta.
- **Operación en tiempo real.** Se diseñan y fabrican máquinas con hardware especial para realizar operaciones en paralelo, necesarias en la ejecución de una RNA.

3.3 ELEMENTOS BÁSICOS QUE COMPONEN UNA RED NEURONAL ARTIFICIAL

Una RNA está constituida por neuronas interconectadas dentro de 3 capas. Los datos ingresan por medio de la "capa de entrada", estos datos son procesados dentro de las "capas ocultas", el número de capas ocultas está dado por la arquitectura de la red y por último los datos procesados salen por la "capa de salida".

En la Figura 2 se puede observar, la arquitectura de una RNA:

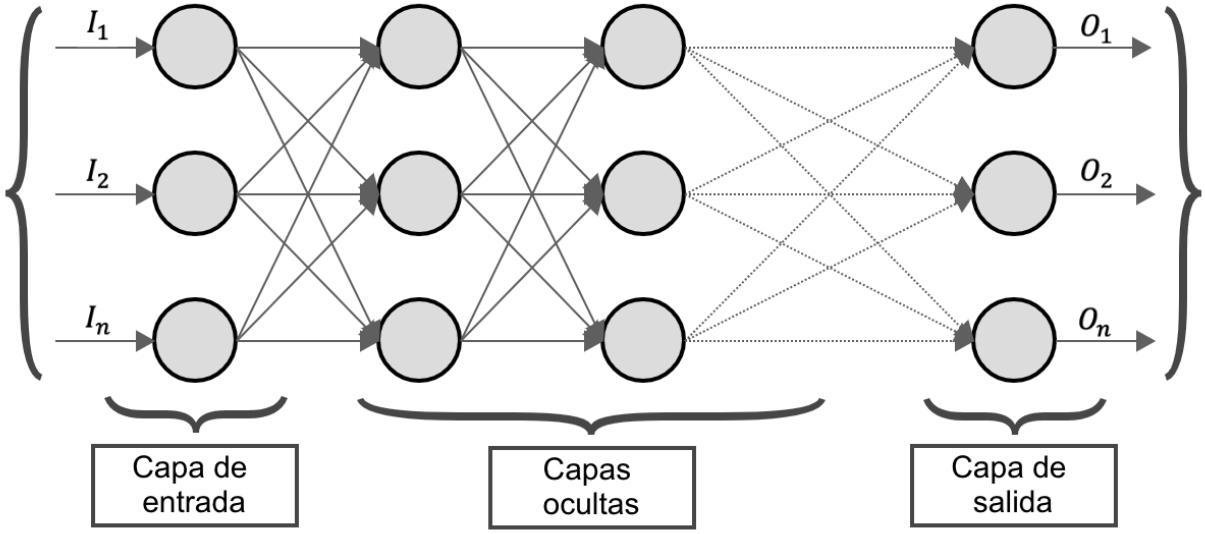

Figura 2. Arquitectura de una RNA.

3.3.1 Función de entrada (*Input Function*)

Todos los valores de entrada (in_{i1}, in_{i2}, …) son tratados como si fueran uno solo, por eso recibe el nombre de *"entrada global"*. Para combinar estas

entradas es utilizada la *"función de entrada"*, la cual se calcula a partir del vector de entrada. Esta función puede escribirse:

$$input_i = (in_{i1} \cdot w_{i1}) * (in_{i2} \cdot w_{i2}) * \dots (in_{in} \cdot w_{in})$$

Donde: $*$ representa el operador apropiado (por ejemplo: máximo, sumatoria, productoria, etc.), n al número de entradas a la neurona N_i y w_i al peso.

Los valores de entrada se multiplican por los pesos ingresados a la neurona. Por consiguiente, los pesos cambian de valor por la influencia que tienen los valores de entrada.

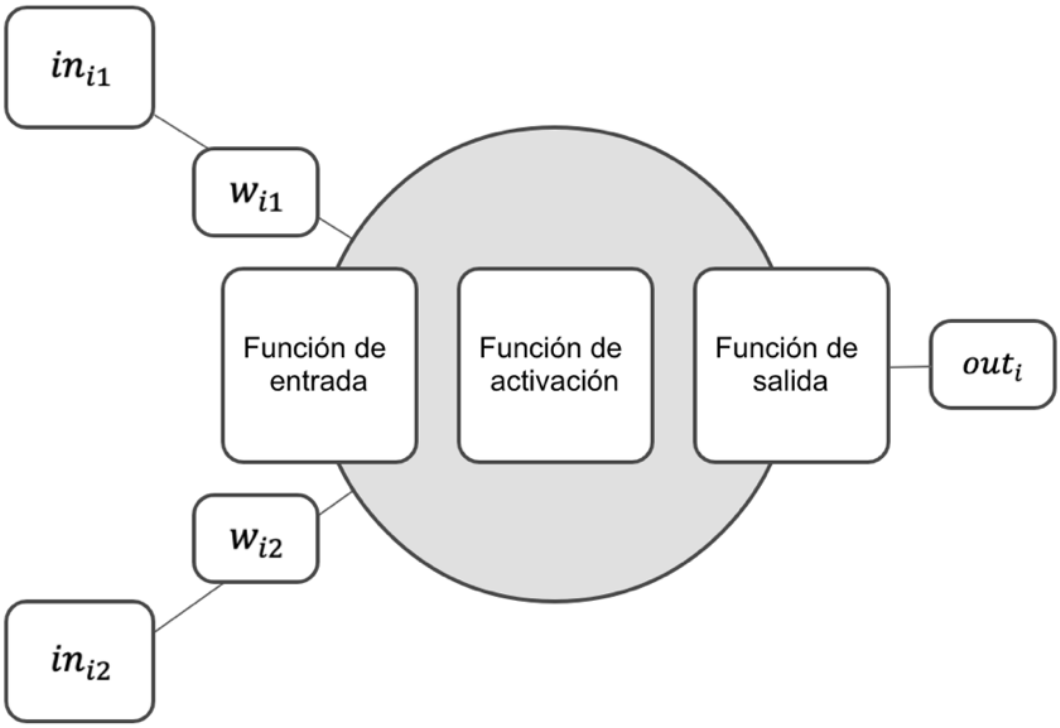

Figura 3. Ejemplo de una neurona con 2 entradas y 1 salida.

La nomenclatura utilizada en la Figura 3 es la siguiente: para las entradas de cada una de las neuronas es in_{i1} y w_{i1} para sus pesos correspondientes. El conjunto de todos las n entradas $in = (in_{i1},\ in_{i2}, \dots, in_{in})$ es comúnmente llamado *"vector entrada"*.

Algunas de las funciones de entrada más comúnmente utilizadas y conocidas son:

1) **Sumatoria de las entradas pesadas:** es la suma de todos los valores de entrada a la neurona, multiplicados por sus correspondientes pesos.

$$\sum_j (n_{ij}\, w_{ij}), \qquad con\ j = 1, 2, \dots, n$$

2) **Productoria de las entradas pesadas:** es el producto de todos los valores de entrada a la neurona, multiplicados por sus correspondientes pesos.

$$\prod_j (n_{ij}\, w_{ij}), \qquad con\ j = 1, 2, \dots, n$$

3) **Máximo de las entradas pesadas:** solamente toma en consideración el valor de entrada más fuerte, previamente multiplicado por su peso correspondiente.

$$Max_j (n_{ij}\, w_{ij}), \qquad con\ j = 1, 2, \dots, n$$

3.3.2 Función de activación (*Activation Function*)

Una neurona biológica tiene un *"estado de activación"*; es decir puede estar activa (excitada) o inactiva (no excitada). Las neuronas artificiales también tienen diferentes estados de activación, en la mayoría de los casos 2, al igual que las biológicas.

La función de activación calcula el estado de actividad de una neurona; tomando el valor de entrada y comparando si es mayor o menor con el umbral Θ_i, para transformarlo en un valor de estado de activación, cuyo rango normalmente va de (0 a 1) o de (-1 a 1).

Las funciones de activación más comúnmente utilizadas se detallan a continuación:

1) Función lineal

$$f(x) = \begin{cases} -1 & x \leq 1/a \\ a * x & -\frac{1}{a} < x < 1/a, \\ 1 & x \geq 1/a \end{cases} \qquad con\ x = gin_i - \Theta_i, y\ a > 0$$

Los valores de salida obtenidos por medio de esta función de activación serán: $a \cdot (gin_i - \Theta_i)$, cuando el argumento de $(gin_i - \Theta_i)$ esté comprendido dentro del rango $(-1/a,\ 1/a)$. Por encima o por debajo de esta zona se fija la salida en 1 o -1, respectivamente. Cuando $a = 1$ (siendo que la misma afecta la pendiente de la gráfica), la salida es igual a la entrada.

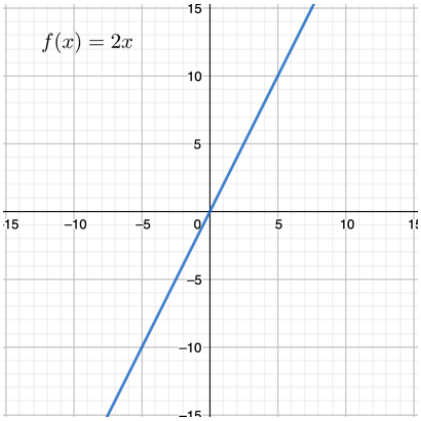

Figura 4. Función de activación lineal.

2) Función sigmoidea

$$f(x) = \frac{1}{1 + e^{-g\,x}}, \qquad con\ x = gin_i - \Theta_i$$

Los valores de salida que proporciona esta función están comprendidos dentro de un rango que va de 0 a 1. Al modificar el valor de g se ve afectada la pendiente de la función de activación.

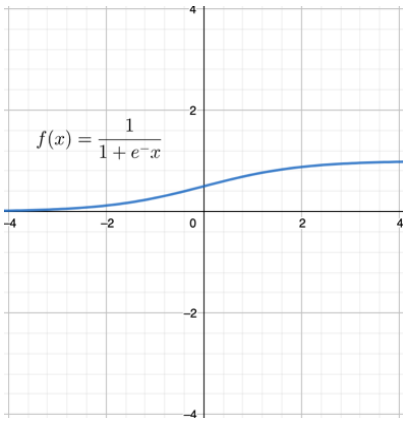

Figura 5. Función de activación sigmoidea.

3) Función tangente hiperbólica

$$f(x) = \frac{e^{g\,x} - e^{-g\,x}}{e^{g\,x} + e^{-g\,x}}, \qquad con\ x = gin_i - \theta_i$$

Los valores de salida de la función tangente hiperbólica están comprendidos dentro de un rango que va de -1 a 1. Al modificar el valor de g se ve afectada la pendiente de la función de activación.

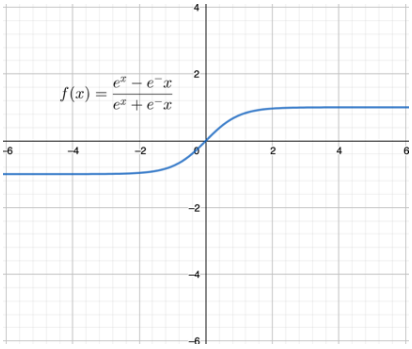

Figura 6. Función de activación tangente hiperbólica.

3.3.3 Función de salida (*Output Function*)

Por último, una neurona necesita la *"función de salida"*. El valor resultante de esta función es la salida de la neurona i (out_i); con este valor se determina si el resultado se transfiera a las neuronas vinculadas. Si la función de activación está por debajo de un umbral determinado, ningún valor de salida se pasa a las siguientes neuronas. Normalmente, los valores de salida están comprendidos en el rango $[0, 1]$ o $[-1, 1]$.

Dos de las funciones de salida más comunes son:

- Ninguna: La salida es la misma que la entrada, también se conoce como *"función de identidad"*.
- Binaria: $\begin{cases} 1, & si\ act_i \geq \xi_i \\ 0, & si\ act_i < \xi_i \end{cases}$, $donde\ \xi_i es\ el\ umbral.$

3.4 CAPAS DE UNA RED NEURONAL ARTIFICIAL

Como se mencionó en la sección anterior, dentro de una RNA, su distribución de neuronas se realiza formando niveles o capas, donde cada capa tiene un número determinado de neuronas y a partir de su posición dentro de la red, se pueden distinguir tres tipos de capas:

- **De entrada:** Es la capa inicial de la red, esta recibe la información proveniente de las fuentes externas a la red (datos de entrada).
- **Ocultas:** Son las capas internas dentro de la red, estas no tienen contacto con el entorno exterior. El número de capas ocultas puede estar entre cero y un número elevado. Las neuronas de las capas ocultas pueden estar interconectadas de distinta manera, lo que determina, las distintas topologías de redes neuronales artificiales.
- **De salida:** Es la última capa de la red, esta transfiere el resultado hacia el exterior.

En la Figura 2 se puede ver el ejemplo de la arquitectura de una posible red neuronal multicapa, en la que cada neurona o nodo está conectada con neuronas de un nivel superior y no entre neuronas de su misma capa. También es bueno mencionar que

una red es totalmente conectada si todas las salidas de un nivel llegan a todos los nodos del siguiente nivel.

3.5 MECANISMOS DE APRENDIZAJE

Una neurona debe de extraer y generalizar a partir de un determinado número de casos anteriores. Una RNA debe aprender a calcular la salida correcta para cada caso (arreglo o vector) de entrada en el conjunto de ejemplos. A este proceso de aprendizaje se denomina: “fase o proceso de entrenamiento”. Por ende, el conjunto de datos que se utiliza en este proceso es llamado: “conjunto de datos de entrenamiento”.

Si en la topología (arquitectura) de la red dentro de las diferentes funciones de cada neurona (entrada, activación y salida) no pueden cambiar durante el proceso de entrenamiento, mientras que los pesos de cada una de las conexiones si pueden hacerlo; esto significa que el aprendizaje de la RNA presenta una “adaptación de los pesos”.

De manera más compacta, la fase de entrenamiento de una RNA es el proceso por el cual una red modifica sus pesos en respuesta a los valores de entrada. En los sistemas biológicos existe una continua destrucción y creación de conexiones entre las neuronas, del mismo modo en los modelos de redes neuronales artificiales los cambios que se produces simulan la destrucción, modificación y creación de conexiones. Para esto, una nueva conexión implica que el peso de esta pasa a tener un valor diferente de cero y una conexión se destruye cuando su peso pasa a ser cero.

Con la modificación de los pesos de las conexiones de la red en la fase de entrenamiento, se puede decir que este proceso ha terminado (la red ha aprendido) cuando los valores de los pesos permaneces estables.

Es importante considerar como se modifican los valores de los pesos de una RNA, es decir, cuáles son los criterios que se siguen para cambiar el valor asignado a las conexiones para permitir el aprendizaje de un nuevo caso.

Existen dos métodos de aprendizaje que pueden distinguirse:

1) Aprendizaje supervisado.
2) Aprendizaje no supervisado.

Un criterio que se puede utilizar para diferenciar las reglas de aprendizaje se basa en considerar si la red puede aprender durante su funcionamiento habitual, o si para su aprendizaje es necesaria la desconexión de la red. En el primer caso, se trataría de un aprendizaje *ON LINE*, mientras que el segundo es lo que se conoce como *OFF LINE*.

Cuando el aprendizaje es *OFF LINE*, se distingue entre una fase de entrenamiento y una fase de operación o funcionamiento, existiendo un conjunto de datos de entrenamiento y un conjunto de datos de prueba. Además, los pesos de las conexiones permanecen fijos después de que termina la etapa de entrenamiento (Damián, 2001).

Una generalización de la formula o regla para decir los cambios en los pesos es:

Peso Nuevo = Peso Viejo + Cambio de Peso

Matemáticamente es:

$$w_{ij}(t+1) = w_{ij}(t) + \Delta w_{ij}(t)$$

Donde t hace referencia a la etapa de aprendizaje, $w_{ij}(t+1)$ al peso nuevo y $w_{ij}(t)$ al peso viejo.

3.5.1 Aprendizaje supervisado

El aprendizaje supervisado se caracteriza porque el proceso de aprendizaje se realiza mediante un entrenamiento controlado por un agente externo, que determina la respuesta que debería generar la red a partir de una entrada determinada. En caso de que la salida no coincida con la deseada, el supervisor procederá a modificar los pesos de las conexiones, con el fin de conseguir que la salida obtenida se aproxime a la deseada.

Para este tipo de aprendizaje se suelen considerar, tres formas de llevarlo a cabo:

1) Aprendizaje por corrección de error
2) Aprendizaje por refuerzo
3) Aprendizaje estocástico

3.5.2 Aprendizaje no supervisado

Las redes con aprendizaje no supervisado no requieren de una influencia externa para ajustar los pesos de las conexiones entre sus neuronas. La red no recibe ninguna información por parte del entorno que le indique si la salida generada es correcta o incorrecta.

El objetivo de estas redes es encontrar las características, regularidades, correlaciones o categorías que se puedan establecer entre los datos de entrada. Para la interpretación de la salida de estas redes existen varias posibilidades, que dependen de su arquitectura y del algoritmo de aprendizaje empleado.

En algunos casos, la salida de la red representa un grado de familiaridad o similitud entre la información de entrada y el conjunto de entradas que hasta entonces recibió. En otro caso, podría realizar una clusterización (*Clustering*) o establecimiento de categorías, indicando la red a la salida a qué categoría pertenece la información presentada a la entrada, siendo la propia red quien debe de encontrar las categorías apropiadas para agrupar la información a partir de las correlaciones entre los conjuntos de datos presentados.

Por lo general para el algoritmo de aprendizaje no supervisado se suelen considerar dos tipos:

1) Aprendizaje hebbiano:
2) Aprendizaje competitivo y comparativo:

3.6 DETENCIÓN DEL PROCESO DE APRENDIZAJE

Es necesario establecer una condición de detención, para determinar cuándo se detendrá el proceso de aprendizaje.

Normalmente el entrenamiento se detiene cuando el cálculo del error cuadrado de todos los casos de entrenamiento ha alcanzado un mínimo, o cuando el error está por debajo de un determinado umbral.

Otra condición para la detención del aprendizaje suele ser cuando se ha alcanzado un cierto número de ciclos y/o pasos de entrenamiento correctamente corridos.

Una vez alcanzada la condición de detención, los pesos no volverán a cambiar. Entonces podemos decir que la transformación de los datos de entrada en valores deseados de salida está resuelta.

Es importante considerar que los casos de entrenamiento se seleccionen de manera aleatoria, esto beneficia el aprendizaje de la red lo que le brinda un mayor rango de respuesta para futuros casos.

3.7 VALIDACIÓN DE LA RED NEURONAL ARTIFICIAL

Una vez finalizada la fase de entrenamiento, con los pesos de las conexiones en la red fijos. El siguiente paso es comprobar si la RNA puede resolver nuevos problemas, para los que ha sido entrenada. Por lo tanto, como se mencionó anteriormente, es necesario otro conjunto de datos, denominado “conjunto de validación o testeo”, con el propósito de validar la red.

Cada caso del conjunto de evaluación contiene los valores de entrada, con su correspondiente salida deseada; pero esta vez esta salida no le es otorgada a la RNA.

Al final se comparan las salidas calculadas por la red de cada uno de los casos con las salidas deseadas.

Solo queda por resolver de qué manera ha de considerarse cuando una salida de la red es correcta o no. Normalmente se establece un umbral con rangos de valores hacia arriba y hacia abajo.

3.8 REDES NEURONALES CON CONEXIÓN HACIA ADELANTE

3.8.1 Perceptrón simple

El perceptrón simple fue introducido por Rosenblatt y es un modelo unidireccional compuesto por dos capas de neuronas, una de entrada y otra de salida (Rosenblatt, 1962). La operación en un perceptrón simple que consta de n neuronas de entrada y m neuronas de salida, se puede expresar como:

$$y_i = f(\sum_{i=1}^{n} w_{ij}x_j - \theta_i)$$

Las neuronas de entrada son discretas y la función de activación de las neuronas de la capa de salida es de tipo escalón. Véase la Figura 7.

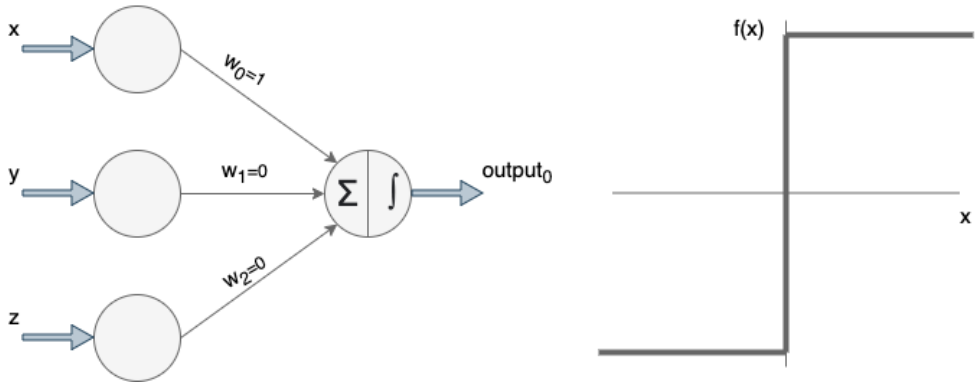

Figura 7. Arquitectura (izquierda) y función de activación (derecha) de un perceptrón simple.

El perceptrón simple puede utilizarse como clasificador, radicando su importancia histórica en su carácter de dispositivo con la capacidad de entrenarse, ya que el algoritmo de aprendizaje del modelo introducido por Rosenblatt en 1962 permite determinar automáticamente los pesos sinápticos que clasifican un conjunto de patrones a partir de un conjunto de ejemplos etiquetados (Rosenblatt, 1962).

El perceptrón simple tan sólo puede discriminar entre dos clases linealmente separables, es decir, clases cuyas regiones de decisión pueden ser separadas mediante una única condición lineal o hiperplano.
Si denotamos por x_1 *y* x_2a las dos neuronas de entrada, la operación efectuada por el perceptrón simple consiste en:

$$y = \begin{cases} 1 \; si \; w_1 x_1 + \; w_2 x_2 \geq \; \theta \\ 0 \; si \; w_1 x_1 + \; w_2 x_2 < \; \theta \end{cases}$$

Si consideramos x_1 *y* x_2 situadas sobre los ejes de abscisas y ordenadas respectivamente, la condición $w_1 x_1 + w_2 x_2 - \theta = 0$ es equivalente a:$x_2 = -\frac{w_1}{w_2} x_1 + \frac{\theta}{w_2}$ y representa una recta que define la región de decisión determinada por el perceptrón simple. Es por ello por lo que dicho

perceptrón simple representa un discriminador lineal, al implementar una condición lineal que separa dos regiones en el espacio que representan dos clases diferentes de patrones. Véase la Figura 8.

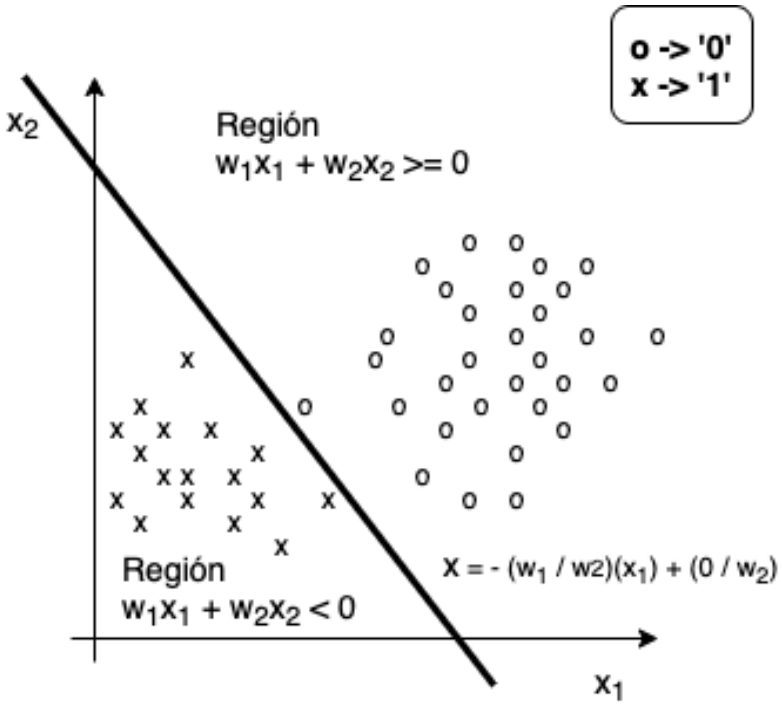

Figura 8. Región de decisión correspondiente a un perceptrón simple con dos neuronas de entrada.

3.8.2 Perceptrón multicapa

El perceptrón multicapa o MLP (*Multi-Layer Perceptron*) se suele entrenar por medio de un algoritmo de retropropagación de errores o *Backpropagation* de ahí que dicha arquitectura se conozca también bajo el nombre de red de retropropagación.

El desarrollo del algoritmo BP resulta una curiosa historia de redescubrimientos y olvidos. Si bien fue Werboz en 1974 quien en su tesis

doctoral lo introdujo por primera vez, el hecho no tuvo repercusión en su época hasta que en 1986 Rumelhart y col. lo redescubrieron de manera independiente y comenzaron a popularizarlo ayudados por los avances en computación existentes en la época, los cuales permitían satisfacer los costes computacionales que el algoritmo BP requiere (Werboz, 1974; Rumelhart & MacClelland, 1988).

La estructura de un MLP con una única capa oculta se muestra en la Figura 9.

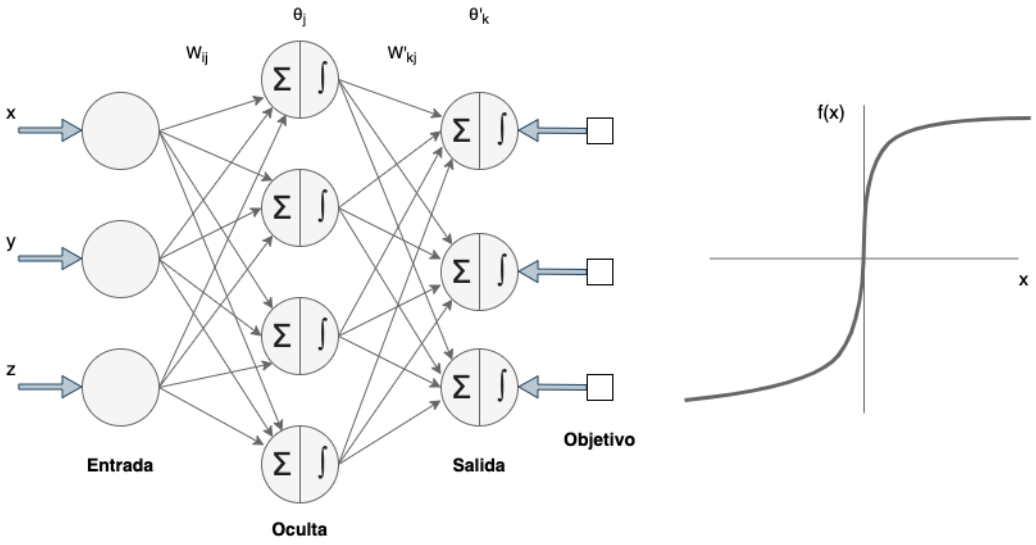

Figura 9. Arquitectura (izquierda) y función de activación (derecha) para el perceptrón multicapa.

$$\boldsymbol{x_i^\mu \rightarrow y_j^\mu \rightarrow z_k^\mu \rightarrow t_k^\mu}$$

Denotamos por x_i a las n entradas de la red, y_j a las θ salidas de la capa oculta y z_k a las s salidas de la capa final, por tanto, a las salidas de la red las cuales deben de ser comparadas con las salidas objetivo t_k. Además, w_{ij} representan los pesos de la capa oculta, θ_j sus umbrales correspondientes, w'_{kj} los pesos de la capa de salida y θ'_k sus umbrales respectivos.

Las operaciones efectuadas por un MLP con una única capa oculta y con funciones de activación para la capa oculta y capa final de tipo sigmoide y lineal respectivamente, son las siguientes:

$$z_k = \sum_{j=1}^{o} w'_{kj} y_j - \theta'_k = \sum_{j=1}^{o} w'_{kj} f\left(\sum_{i=1}^{n} w_{ji} x_i - \theta_j\right) - \theta'_k$$

Siendo $f(x)$ una función de tipo sigmoideo.

3.8.3 Regla de los mínimos cuadrados – *Least Mean Square*

La regla de aprendizaje de los mínimos cuadrados LMS (*Least Mean Square*) se fundamenta en considerar el aprendizaje de dichos pesos como un problema de optimización de una determinada función de coste. Dicha función de coste va a medir el rendimiento actual de la red y dependerá de los pesos sinápticos de la misma. Un método de optimización aplicado a dicha función de coste va a proporcionar una regla de actualización de los pesos, que a partir de los patrones de aprendizaje modifique iterativamente los pesos hasta alcanzar la configuración "óptima" de los mismos (Widrow & Hoff, 1988).

El método de optimización utilizado por la regla LMS es el denominado descenso por el gradiente, el cual puede ser visto como un optimizador local en un espacio de búsqueda continuo. Se parte de una función de coste E que proporciona el error actual cometido por la RNA. Dicha función de coste será una función de los pesos sinápticos. Es decir:

$$E: \mathbb{R}^{(nxm)+m} \rightarrow \mathbb{R}$$

$$(w_{11}, \dots, w_{1n}, \dots, w_{m1}, \dots, \theta_1, \dots, \theta_m) \rightarrow E(w_{11}, \dots, w_{1n}, \dots, w_{m1}, \dots, \theta_1, \dots, \theta_m)$$

El objetivo del aprendizaje, es decir, del proceso de actualización de pesos, consiste en encontrar la configuración de estos que corresponde al mínimo global de la función de error o función de coste definida. Con frecuencia en una RNA genérica nos vamos a tener que conformar con un mínimo local suficientemente bueno.

Para encontrar la configuración de pesos óptima mediante el método del descenso por el gradiente se opera del siguiente modo. Se parte en $t = 0$ de una cierta configuración de peso $\boldsymbol{w}^0$, y se calcula la dirección de la máxima variación de la función $E(\boldsymbol{w})$ en $\boldsymbol{w}^0$, la cual vendrá dada por su gradiente en $\boldsymbol{w}^0$. El sentido de la máxima variación apuntara hacia una colina en la hipersuperficie que representa a la función $E(\boldsymbol{w})$. Luego, se modifican los parámetros $\boldsymbol{w}$ siguiendo el sentido opuesto al indicado por el gradiente de la función de error. De esta manera se lleva a cabo un descenso por la hipersuperficie de error, aproximándose en una cierta cantidad al va mínimo local. El proceso anterior de descenso por el gradiente se itera hasta que dicho mínimo local es alcanzado.

Matemáticamente se expresaría: $\boldsymbol{w}(t+1) = \boldsymbol{w}(t) - \in \nabla E(\boldsymbol{w})$ donde $\in$ (que puede ser diferente para cada caso) indica el tamaño del paso tomado en cada iteración. Si bien idealmente $\in$ debería de ser infinitesimal, una elección de este tipo llevaría a un proceso de entrenamiento extremadamente lento, de ahí que $\in$ debe ser lo suficientemente grande como para que cumpla el compromiso de rápida actualización sin llevar a oscilaciones.

Teniendo en cuenta la función de coste a minimizar, si contemplamos el problema como un problema de optimización, podríamos obtener lo siguiente:

$$E(\boldsymbol{w}) = \frac{1}{2}\sum_{r=1}^{N}\sum_{i=1}^{m}(c_i^r - y_i^r)^2$$

Es decir, la función de error mide el error cuadrático correspondiente a las salidas actuales de la red respecto de los objetivos y calculando el gradiente de dicha función con respecto de cada variable (peso), y teniendo en cuenta que $y_i^r = \sum_{j=1}^{n} w_{ij}x_j^r - \theta_i$, se obtiene:

$$\frac{\partial E(w_{ij})}{\partial w_{ij}} = -\left(\frac{1}{2}\right)2\sum_{r=1}^{N}(c_i^r - y_i^r)\frac{dy_i^r}{dw_{ij}} = -\sum_{r=1}^{N}(c_i^r - y_i^r)x_j^r$$

De ahí que el incremento de los pesos, según la regla de adaptación *LMS* resulte ser:

$$\Delta w_{ij} = -\epsilon\frac{\partial E(w_{ij})}{\partial w_{ij}} = \epsilon\sum_{r=1}^{N}(c_i^r - y_i^r)\,x_j^r$$

La regla de LMS produce actualizaciones de tipo continuo, de manera que un mayor error produce una actualización mayor.

3.8.4 Entrenamiento de un perceptrón multicapa con *Backpropagation*

El algoritmo de retropropagación de errores (*Backpropagation*), podría verse como una extensión del algoritmo LMS a las redes multicapa. Para ello se plantea una función de error similar a la utilizada para obtener la regla de actualización de pesos LMS, y se obtendrán las fórmulas correspondientes al algoritmo BP tanto en función de los pesos de la capa de salida como de los pesos de la capa oculta. Se utiliza la regla de la cadena *(Chain rule)* y para ello se necesita que las funciones de transferencia de las neuronas sean derivables.

Tomando en cuenta un MLP de tres capas, es decir con una capa oculta, cuya arquitectura se presenta en la Figura 9, con las entradas, salidas, pesos y umbrales de las neuronas definidos anteriormente. Dado un patrón de entrada $x^r(r = 1, \dots, N)$ la operación global de esta arquitectura se representa del siguiente modo, para cada una de las k con $(k = 1, \dots, s)$ neuronas de salida:

$$z_k^r = \sum_{j=1}^{o} w'_{kj} y_j^r - \theta'_k = \sum_{j=1}^{o} w'_{kj} f\left(\sum_{i=1}^{n} w_{ji} x_i^r - \theta_j\right) - \theta'_k$$

La función de costo de la que se parte es el error cuadrático medio, $E(\boldsymbol{w}, \boldsymbol{w'}, \boldsymbol{\theta}, \boldsymbol{\theta'}) = \frac{1}{2}\sum_{r=1}^{N}\sum_{k=1}^{m}(c_k^r - z_k^r)^2$, siendo la función $E(\boldsymbol{w}, \boldsymbol{w'}, \boldsymbol{\theta}, \boldsymbol{\theta'})$.

$$E: \mathbb{R}^{n\times o+o\times s+o+s} \to \mathbb{R}$$

$$(w_{11}, \dots, w_{on}, w'_{1s}, \dots, w'_{os}, \theta_1, \dots, \theta_o, \theta'_1, \dots, \theta'_s) \to E(w_{11}, \dots, w_{on}, w'_{1s}, \dots, w'_{os}, \theta_1, \dots, \theta_o, \theta'_1, \dots, \theta'_s)$$

La minimización se lleva a cabo por el descenso del gradiente, existiendo en este caso un gradiente respecto de los pesos de la capa de salida $\Delta w'_{kj} = -\epsilon \frac{\partial E}{\partial w'_{kj}}$ y otro respecto de los pesos de la capa oculta $\Delta w_{ji} = -\epsilon \frac{\partial E}{\partial w_{ji}}$.

Las expresiones de actualización de los pesos se obtienen derivando, teniendo en cuenta las dependencias funcionales y aplicando la regla de la cadena.

$$\Delta w'_{kj} = \epsilon \sum_{r=1}^{N}\left(c_k^r - \left(\sum_{j=1}^{o} w'_{kj} y_j^r - \theta'_k\right)\right)^2 y_j^r$$

$$\Delta w_{ji} = \epsilon \sum_{r=1}^{N} \Delta_j^r \, x_i^r$$

con $\Delta_j^r = \left(\sum_{k=1}^{s}\left(\sum_{j=1}^{o} w'_{kj} y_j^r - \theta'_k\right) w'_{kj}\right) \frac{\partial f\left(\sum_{i=1}^{n} w_{ji} x_i^r - \theta_j\right)}{\partial\left(\sum_{i=1}^{n} w_{ji} x_i^r - \theta_j\right)}$

La actualización de los umbrales (*Bias*) se realiza por medio de las anteriores expresiones, considerando que el umbral es un caso particular de un peso sináptico, cuya entrada es una constante igual a −1.

En las expresiones anteriores está implícito el concepto de propagación hacia atrás de los errores que da el nombre al algoritmo. En primer lugar, se calcula la expresión $\left(c_k^r - \left(\sum_{j=1}^{o} w'_{kj} y_j^r - \theta'_k\right)\right)$ que se denomina *señal de error*, por ser proporcional al error de la salida actual de la red, con el que se calcula la actualización $\Delta w'_{kj}$ de los pesos de la capa de salida. Después, se propagan hacia atrás los errores $\left(c_k^r - \left(\sum_{j=1}^{o} w'_{kj} y_j^r - \theta'_k\right)\right)$ a través de la sinapsis, obteniendo las señales de error $\left(\sum_{k=1}^{s}\left(\sum_{j=1}^{o} w'_{kj} y_j^r - \theta'_k\right) w'_{kj}\right) \frac{\partial f\left(\sum_{i=1}^{n} w_{ji} x_i^r - \theta_j\right)}{\partial\left(\sum_{i=1}^{n} w_{ji} x_i^r - \theta_j\right)}$ correspondientes a las sinapsis de la capa oculta.

Con estas señales de error se calcula la actualización de Δw_{ji} de las sinapsis ocultas. El algoritmo puede además ser extendido a arquitecturas con más de una capa oculta siguiendo este esquema de retropropagación del error.

Ahora se puede definir el procedimiento para entrenar mediante el algoritmo BP una arquitectura MLP:

Paso 1.

Establecer aleatoriamente los *pesos* y *umbrales* iniciales $(t := 0)$.

Paso 2.

Para cada patrón *r* del conjunto de entrenamiento: llevar a cabo una fase de ejecución para obtener la respuesta de la red frente al patrón *r-ésimo*.

Calcular las señales de error asociadas $\left(c_k^r - \left(\sum_{j=1}^{o} w'_{kj} y_j^r - \theta'_k\right)\right)$ y $\left(\sum_{k=1}^{s}\left(\sum_{j=1}^{o} w'_{kj} y_j^r - \theta'_k\right) w'_{kj}\right) \frac{\partial f\left(\sum_{i=1}^{n} w_{ji} x_i^r - \theta_j\right)}{\partial\left(\sum_{i=1}^{n} w_{ji} x_i^r - \theta_j\right)}$. Calcular el incremento parcial de los *pesos* y *umbrales* debidos a cada patrón *r*.

Paso 3.

Calcular el incremento total actual, extendido a todos los patrones, de los pesos $\Delta w'_{kj}$ y Δw_{ji}. Hacer lo mismo con los *umbrales*.

Paso 4.

Actualizar pesos y umbrales.

Paso 5.

Calcular el error total. Hacer $t := t + 1$ y volver al Paso 2 si todavía el error total no es satisfactorio.

Es importante tener en cuenta que se debe comenzar siempre con pesos iniciales aleatorios pequeños, tanto positivos como negativos. Este esquema de entrenamiento se acostumbra a denominar "*aprendizaje por lotes o en Batch", ya que* se lleva a cabo para todos y cada uno de los casos del conjunto de entrenamiento, primero se calcula la variación en los pesos debida a cada patrón, se acumulan y se efectúa la actualización de los pesos (Larrañaga, et al., 2015).

CAPÍTULO IV NUEVO ENFOQUE DE REDES NEURONALES ARTIFICIALES

En la presente sección se muestra la deducción matemática con respecto al procedimiento para entrenar una Red Neuronal Artificial (RNA) mediante el algoritmo de *Backpropagation*. Esta está dividida en tres apartados, en la primera se muestran las ecuaciones generales de un perceptrón simple, uno multicapa, la propagación hacia adelante y hacia atrás, en el segundo apartado se encuentra la propuesta de este proyecto y en el tercer apartado se muestran las ecuaciones del algoritmo de BP bajo un contexto matricial, estas permiten la reducción del coste computacional, con el fin de implementar y testear estas ecuaciones en una sección posterior.

4.1 EXPLICACIÓN DE UNA RED NEURONAL ARTIFICIAL

4.1.1 Perceptrón simple

El perceptrón simple se puede modelar a través de la siguiente ecuación:

$$y = \varphi(z)$$

donde y es la salida de un perceptrón simple al evaluar una pre-activación z en una función de activación $\varphi(z)$, usualmente la función Sigmoide:

$$\varphi(z) = \frac{1}{1 + e^{-z}} \in (0, 1)$$

En este contexto z se calcula a partir de la suma ponderada entre las entradas x_i y los pesos w_i hasta n entradas, como se muestra en la siguiente ecuación:

$$z = \sum_{i=1}^{n} w_i x_i$$

Sin embargo, para una aplicación computacional, la preactivación z se puede calcular a partir de un producto punto entre dos vectores, por lo que la siguiente ecuación se puede representar de la siguiente manera:

$$z = W \cdot X$$

donde $W = [w_1, w_2, w_3, \dots, w_4]$ es un vector fila conformado por los pesos del perceptrón y X es un vector columna con los datos de entrada:

$$X = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_n \end{bmatrix}$$

por lo que:

$$z = W \cdot X = [w_1, w_2, w_3, \dots, w_n] \cdot \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_n \end{bmatrix} = [w_1 \cdot x_1 + w_2 \cdot x_2 + w_3 \cdot x_3 + \cdots + w_n \cdot x_n]$$

De esta manera, el perceptrón simple es igual que:

$$y = \varphi(z) = \varphi(W \cdot X)$$

4.1.2 Perceptrón multicapa

La operación matemática básica que realiza un perceptrón multicapa es la siguiente:

$$y = \varphi(Z)$$

A simple vista podría ser vista como la ecuación del perceptrón simple, sin embargo, en este contexto se extiende el concepto, ya que Z es un vector (columna) de pre-activaciones z_i que se calcula como sigue:

$$Z = W \cdot X + b$$

donde b es el bias (sesgo), usualmente un vector de 1's, X es un vector columna de n entradas y W es una matriz de pesos, en donde cada fila m de la matriz, corresponde a un vector de n pesos de un perceptrón simple o neurona:

$$W = \begin{bmatrix} w_{1,1} & w_{1,2} & w_{1,3} & \cdots & w_{1,n} \\ w_{2,1} & w_{2,2} & w_{2,3} & \cdots & w_{2,n} \\ w_{3,1} & w_{3,2} & w_{3,3} & \cdots & w_{3,n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ w_{m,1} & w_{m,2} & w_{m,3} & \cdots & w_{m,n} \end{bmatrix}$$

Por lo que la operación de pre-activación de la siguiente ecuación, se puede generalizar de la siguiente manera:

$$Z = \begin{bmatrix} z_1 \\ z_2 \\ z_3 \\ \vdots \\ z_n \end{bmatrix} = \begin{bmatrix} w_{1,1} & w_{1,2} & w_{1,3} & \cdots & w_{1,n} \\ w_{2,1} & w_{2,2} & w_{2,3} & \cdots & w_{2,n} \\ w_{3,1} & w_{3,2} & w_{3,3} & \cdots & w_{3,n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ w_{m,1} & w_{m,2} & w_{m,3} & \cdots & w_{m,n} \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_n \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \\ 1 \\ \vdots \\ 1 \end{bmatrix}$$

4.1.3 Propagación hacia adelante

Para una capa $l \in \{1, \dots, L\}$, donde L es la capa de salida, inicializar una matriz de pesos $W^l \in [-1,1]$ con valores aleatorios. En la capa de entrada $l = 1$ (ubicado como superíndice), tenemos lo siguiente:

$$\begin{cases} Z^1 = W^1 \cdot X \\ a^1 = \varphi(Z^1) \end{cases}$$

donde Z^1 es un vector de pre-activación construido a partir del producto punto entre la matriz de pesos W^1 y el vector de datos de entrada X. Finalmente a^1 es un vector con las salidas de las neuronas en la capa de entrada, el cual se convertirá en los datos de entrada de la siguiente capa.

Para $l \in \{2, \dots, L\}$, se realiza como se muestra a continuación:

$$\begin{cases} Z^l = W^l \cdot a^{(l-1)} \\ a^l = \varphi(Z^l) \end{cases}$$

Es importante notar que $a^L = y$ es la salida de la RNA.

4.1.4 Propagación hacia atrás (*Backpropagation*)

El error de salida de la red se calcula de la siguiente manera:

$$\begin{cases} e^L = d - a^L \\ \delta^L = \varphi'(Z^L) \odot e^L \end{cases}$$

Donde $\odot$ es el producto de *Hadamard*, en éste se realizan multiplicaciones elemento a elemento, las matrices deben ser de la misma forma y como resultado se obtiene otra matriz con la misma forma. Para $l \in \{L-1, L-2, \dots, 3, 2, 1\}$:

Ejemplo del producto de *Hadamard*:

$$\begin{bmatrix} 1 \\ 3 \\ 2 \end{bmatrix} \odot \begin{bmatrix} 4 \\ 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 \cdot 4 \\ 3 \cdot 1 \\ 2 \cdot 1 \end{bmatrix} = \begin{bmatrix} 4 \\ 3 \\ 2 \end{bmatrix}$$

4.1.4.1 Actualización de pesos

Para la entrada $l = 1$:

$$\begin{cases} \Delta W^1 = \eta \cdot (\delta^1 \otimes X) \\ W^1 = W^1 + \Delta W^1 \end{cases}$$

donde $\eta \in (0,1)$ es conocido como el factor de aprendizaje, $\otimes$ es el operador producto exterior, de tal manera que, $\delta^1 \otimes X \coloneqq \delta^1 \cdot (X)^T$. Finalmente, para $l \in \{2,3, \dots, L\}$:

$$\begin{cases} \Delta W^l = \eta \cdot (\delta^l \otimes a^{(l-1)}) \\ W^l = W^l + \Delta W^l \end{cases}$$

Ejemplo de producto exterior ⊗:

$$\delta = \begin{bmatrix} 1 \\ 3 \\ 2 \end{bmatrix}; X = \begin{bmatrix} 4 \\ 1 \\ 1 \end{bmatrix}; X^T = [4,1,1]; \delta \otimes X \coloneqq \delta \cdot (X)^T; \begin{bmatrix} 1 \\ 3 \\ 2 \end{bmatrix} ([4,1,1]) = [1 \cdot 4 + 3 \cdot 1 + 2 \cdot 1] = 18$$

4.2 PERCEPTRÓN MULTICAPA CON FUNCIÓN GENERALIZADA

4.2.1 Propuesta

El cálculo del error de un perceptrón multicapa es el siguiente:

$$e = d - a$$

donde e es el error, d es el valor deseado, y $a = \varphi(Z)$ es el valor calculado en la salida de la red. En este contexto, $\varphi(Z)$ es la función de activación, comúnmente la función Sigmoide, y Z es la pre-activación de la neurona. Cabe señalarse que en un contexto normal, la pre-activación Z se calcula mediante la siguiente ecuación:

$$Z = \sum wx$$

donde w es el peso y x el dato de entrada de una neurona artificial, lo cual, a grandes rasgos, permite que su comportamiento sea como el de un separador lineal.

Sin embargo, nosotros proponemos la siguiente ecuación:

$$z = \sum w\psi(x)$$

donde $\psi(x)$ es una función que permite modificar los datos de entrada x, permitiendo que la neurona deje de comportarse como un separador lineal. Sin embargo, es importante notar que si $\psi(x) = x$, su comportamiento sería el de una neurona tradicional.

Finalmente, se calcula el error cuadrático medio ξ, el cual permitirá aplicar las técnicas de descenso de gradiente para poder realizar el entrenamiento de la red neuronal artificial:

$$\xi = \frac{1}{2}e^2$$

Debido a la incorporación de la función $\psi(\cdot)$ en la ecuación $z = \sum w\psi(x)$, para realizar la actualización de una red neuronal artificial, se debe de establecer una nueva forma para actualizar los pesos entre ξ y w. Por lo que se deduce la siguiente ecuación:

$$\frac{\partial \xi}{\partial w} = \frac{\partial \xi}{\partial e} \cdot \frac{\partial e}{\partial a} \cdot \frac{\partial a}{\partial Z} \cdot \frac{\partial Z}{\partial w}$$

donde

$$\frac{\partial \xi}{\partial e} = \frac{\partial}{\partial e}\left(\frac{1}{2}e^2\right) = e$$

$$\frac{\partial e}{\partial a} = \frac{\partial}{\partial a}(d - a) = -1$$

$$\frac{\partial a}{\partial Z} = \frac{\partial}{\partial Z}\varphi(Z) = \varphi'(Z)$$

$$\frac{\partial Z}{\partial w} = \frac{\partial}{\partial w}\sum w\varphi(x) = \varphi(x)$$

Finalmente, sustituyendo se obtiene que:

$$\frac{\partial \xi}{\partial w} = -e\varphi'(Z)\psi(x)$$

Con el fin de reducir algunos términos y simplificar la matemática. Si $\delta = e\varphi'(Z)$, entonces la ecuación anterior, se transforma en la siguiente:

$$\frac{\partial \xi}{\partial w} = -\delta\varphi(x)$$

Ahora pasamos a la regla de actualización de pesos, donde:

$$\Delta w = w_{i+1} - w_1$$

Sin embargo, de acuerdo al descenso de gradiente, Δw también es:

$$\Delta w = -\eta \frac{\partial \xi}{\partial w} = -\eta\big(-\delta\psi(x)\big) = \eta\delta\psi(x)$$

donde η, en el contexto de RNAs es el llamado coeficiente de aprendizaje. Igualando las últimas dos ecuaciones, es decir:

$$w_{i+1} - w_1 = \eta\delta\psi(x)$$

obtenemos la regla de actualización de pesos de una neurona, en el contexto del descenso de gradiente:

$$w_{i+1} = w_1 + \eta\delta\psi(x)$$

4.3 MODIFICACIÓN AL ALGORITMO BACKPROPAGATION

En la presente sección de muestra el algoritmo *Backpropagation* bajo un contexto matricial, con la finalidad de reducir coste computacional.

4.3.1 Propagación hacia adelante

Para una capa $l \in \{1, \dots, L\}$, donde L es la capa de salida, inicializar una matriz de pesos W^l con valores aleatorios entre $[-1,1]$, es decir:

$$W \in [-1, 1]$$

En la capa de entrada $l = 1$, tenemos lo siguiente:

$$\begin{cases} Z^1 = W^1 X \\ a^1 = \varphi(Z^1) \end{cases}$$

donde Z^l es un vector de pre-activación construido a partir del producto punto entre la matriz de pesos W^1 y el vector de datos de entrada X. Finalmente a^1 es un vector con las salidas de las neuronas en la capa de entrada, el cual se convertirá en los datos de entrada de la siguiente capa.

Para $l \in \{2, \dots, L\}$, y de acuerdo con la ecuación $z = \sum w\psi(x)$, también incorporamos la función $\psi(\cdot)$ como se muestra a continuación:

$$\begin{cases} Z^l = W^l \psi\left(a^{(l-1)}\right) \\ a^l = \varphi\left(Z^l\right) \end{cases}$$

Es importante notar que $a^L = y$ es la salida de la RNA.

4.3.2 Propagación hacia atrás

El error de salida de la red se calcula de la siguiente manera, se comienza con la última capa L, por eso se le conoce como propagación hacia atrás, esto se puede resumir en la siguiente ecuación:

$$\begin{cases} e^L = d - a^L \\ \delta^L = \varphi'(Z^L) \odot e^L \end{cases}$$

Para las siguientes capas $l \in \{L-1, L-2, \dots, 3, 2, 1\}$, se aplica la siguiente ecuación:

$$\begin{cases} e^l = \left(W^{(l+1)}\right)^T \delta^{(l+1)} \\ \delta^l = \varphi'(Z^l) \odot e^l \end{cases}$$

4.3.3 Actualización de pesos

El último paso para la deducción es la actualización de pesos y con esto se completan las ecuaciones necesarias para realizar la implementación y testeo de estas.

En la primera entrada $l = 1$ la ecuación es la siguiente:

$$\begin{cases} \Delta W^1 = \eta(\delta^1 \otimes X) \\ W^1 = W^1 + \Delta W^1 \end{cases}$$

Donde $\otimes$ es el operador producto exterior, de tal manera que, $\delta^1 \otimes X \coloneqq \delta^1 (X)^T$. Para las siguientes capas $l \in \{2, 3, \dots, L\}$:

$$\begin{cases} \Delta W^l = \eta\left(\delta^l \otimes \psi\left(a^{(l-1)}\right)\right) \\ W^l = W^l + \Delta W^l \end{cases}$$

CAPÍTULO V IMPLEMENTACIÓN COMPUTACIONAL (DIAGRAMAS Y ESTRUCTURA)

En este capítulo se muestran una serie de diagramas que describen las actividades que permiten el desarrollo de este proyecto y brindan la posibilidad de poner a prueba el enfoque propuesto.

Se inicia con los casos de usos para la implementación del nuevo enfoque de la Red Neuronal Artificial (RNA) que se muestran en el diagrama de la Figura 10.

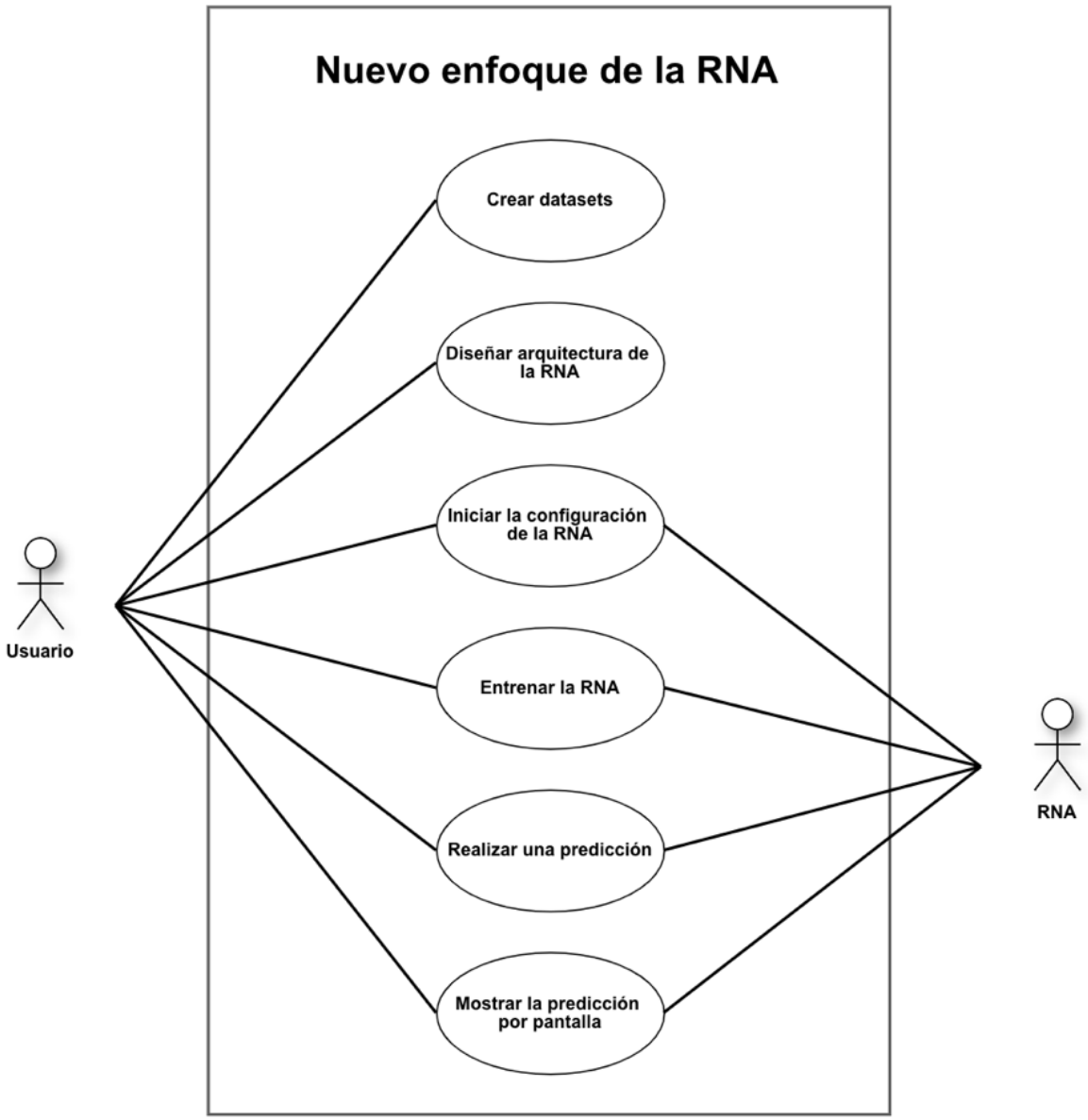

Figura 10. Casos de uso del nuevo enfoque para la RNA.

En este diagrama se observan seis casos de uso, para ampliar su comprensión se describe cada uno a continuación:

1. **Crear *Datasets*:** En esta actividad es necesario generar algunos *Datasets* sintéticos.
2. **Diseñar arquitectura de la RNA:** Es necesario generar un diseño general para brindarle a la RNA una arquitectura con la cual trabajar.
3. **Iniciar la configuración de la RNA:** A partir de la arquitectura, la RNA tiene que ser capaz de realizar su configuración inicial para su posterior entrenamiento.
4. **Entrenar la RNA:** Se debe indicar a la RNA, que realice su entrenamiento donde el usuario indicara parámetros para cumplirse la actividad.
5. **Realizar una predicción:** Una vez que la RNA termino su entrenamiento, se debe de probar si puede realizar una clasificación correcta de los datos de entrada.
6. **Mostrar la predicción por pantalla:** Con los resultados de la predicción, se deben de realizar una serie de graficas que muestre más claramente los resultados.

Para describir cada actividad a detalle, se realizaron los diagramas de actividad de estos casos, exceptuando la de diseñar arquitectura de la RNA; esto porque para esta actividad solo será necesario explicar los parámetros que permiten estructurar la arquitectura y que esta sea proporcionada a la RNA.

Comenzando por el diagrama para crear un *Dataset*, se puede destacar que este consta de siete pasos lineales, lo cual muestra una secuencia simple para crear los *Datasets* sintéticos necesarios para la experimentación y el testeo de la RNA. Ver Figura 11.

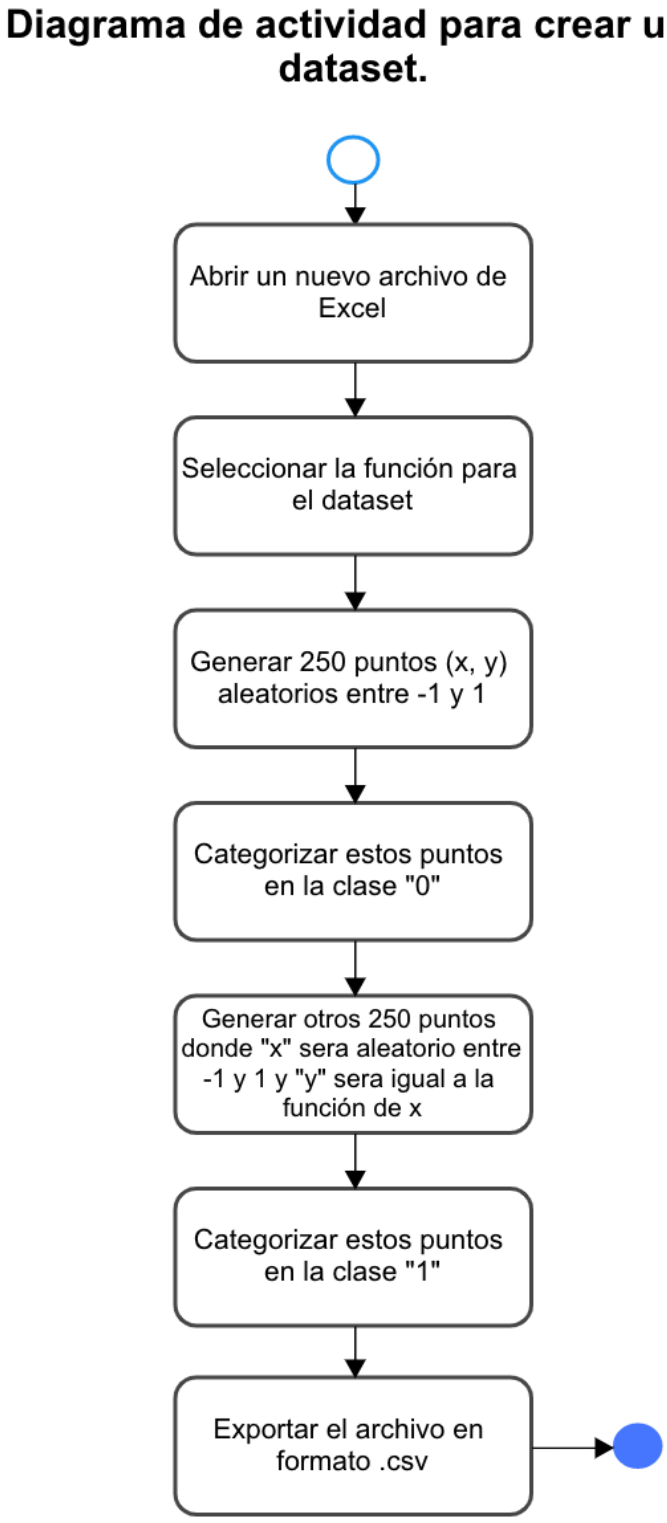

Figura 11. Diagrama de actividad para crear un Dataset.

Para el diseño de la arquitectura, se tuvo en cuenta que esta se le cargaría a la RNA, mediante un archivo JSON, esto porque este tipo de archivos brinda un formato sencillo y entendible a la vista lo que facilita el entendimiento del usuario y la RNA.

Teniendo esto en cuenta y conociendo que los archivos JSON utilizan objetos (nombre/valor). Se diseño la arquitectura de la RNA donde el primer objeto nombrado “0” será la capa de entrada, el ultimo objeto nombrado “N” será la capa de salida, y los objetos intermedios serán las capas ocultas nombradas desde “1”, “2”, “3”, …, “N” como se ve en la Figura 12.

Figura 12. Ejemplo de arquitectura de la RNA en formato JSON.

En este ejemplo podríamos decir que la arquitectura de la red tiene la capa de entrada, tres capas ocultas y la capa de salida.

El objeto “0” o capa de entrada solo tiene un atributo “*number_of_attributes*” que como su nombre lo indica, es el número de atributos de los datos de entrada como se puede ver en la Figura 13.

```
{
    "0": {
        "number_of_attributes": 2
    },
```

Figura 13. Ejemplo de capa de entrada en la arquitectura de la RNA en formato JSON.

En los objetos denominados capas ocultas, se encuentran 4 atributos, “number_of_attributes”, “activation_function”, “derivative_activation_function”, “psi_function”, que sirven para indicar el número de atributos, la función de activación, la derivada de dicha función y la función psi de cada capa oculta. Véase la Figura 14.

```
    "1": {
        "number_of_attributes": 1,
        "activation_function": "1 / (1 + np.e**-x)",
        "derivative_activation_function": "x * (1 - x)",
        "psi_function": "x"
    },
```

Figura 14. Ejemplo de capas ocultas en la arquitectura de la RNA en formato JSON.

El ultimo objeto o capa de salida tiene los mismos atributos que las capas ocultas, sin embargo, es importante resaltar que en el atributo “number_of_attributes” el número que se indique, serán los resultados que arrojara la RNA, un ejemplo de esto se puede ver en la Figura 15.

```
"2": {
    "number_of_attributes": 1,
    "activation_function": "1 / (1 + np.e**-x)",
    "derivative_activation_function": "x * (1 - x)",
    "psi_function": "x"
}
```

Figura 15. Ejemplo de capa de salida en la arquitectura de la RNA en formato JSON.

A continuación, se describen los pasos para realizar la configuración inicial de la RNA, en primer lugar, la RNA recibirá la arquitectura por medio de un archivo JSON y realizara las operaciones que se muestran en el diagrama de la Figura 16. En este diagrama se puede destacar un proceso cíclico que finalizara hasta que la descripción de todas las capas sea vaciada en la RNA.

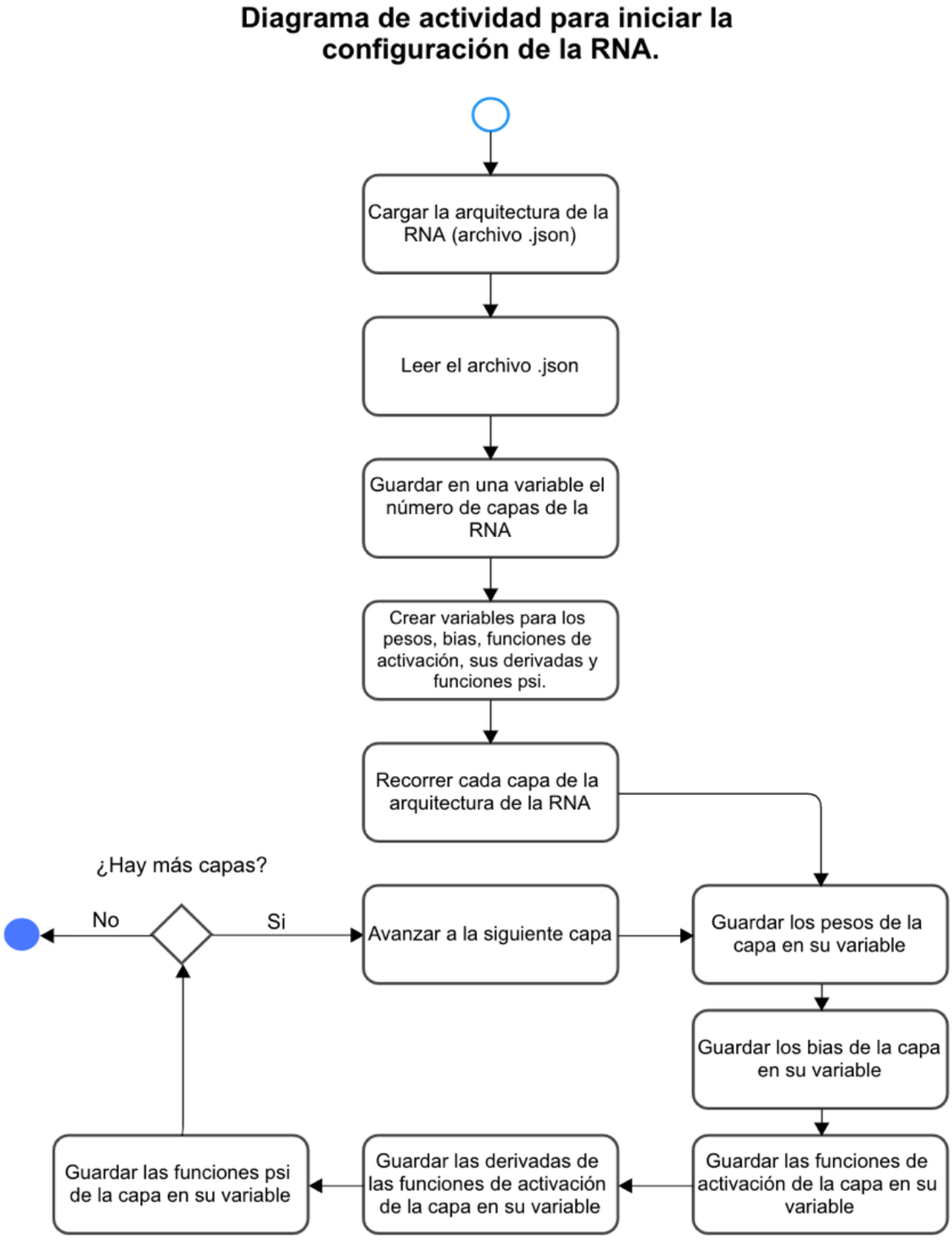

Figura 16. Diagrama de actividad para iniciar la configuración de la RNA.

Aunque el proceso de entrenamiento es primero que el de la predicción, se realizó en orden inverso, esto porque en los pasos para el entrenamiento se lleva a cabo el proceso de predicción. Reciclar este proceso ahorra algunos pasos repetitivos en el siguiente proceso.

Los pasos para realizar una predicción o clasificación de los datos de entrada se muestran en el diagrama de la Figura 17, donde también hay un proceso cíclico que recorre todas las capas de la RNA y al término de la última capa, finaliza el proceso y retorna los resultados de la predicción.

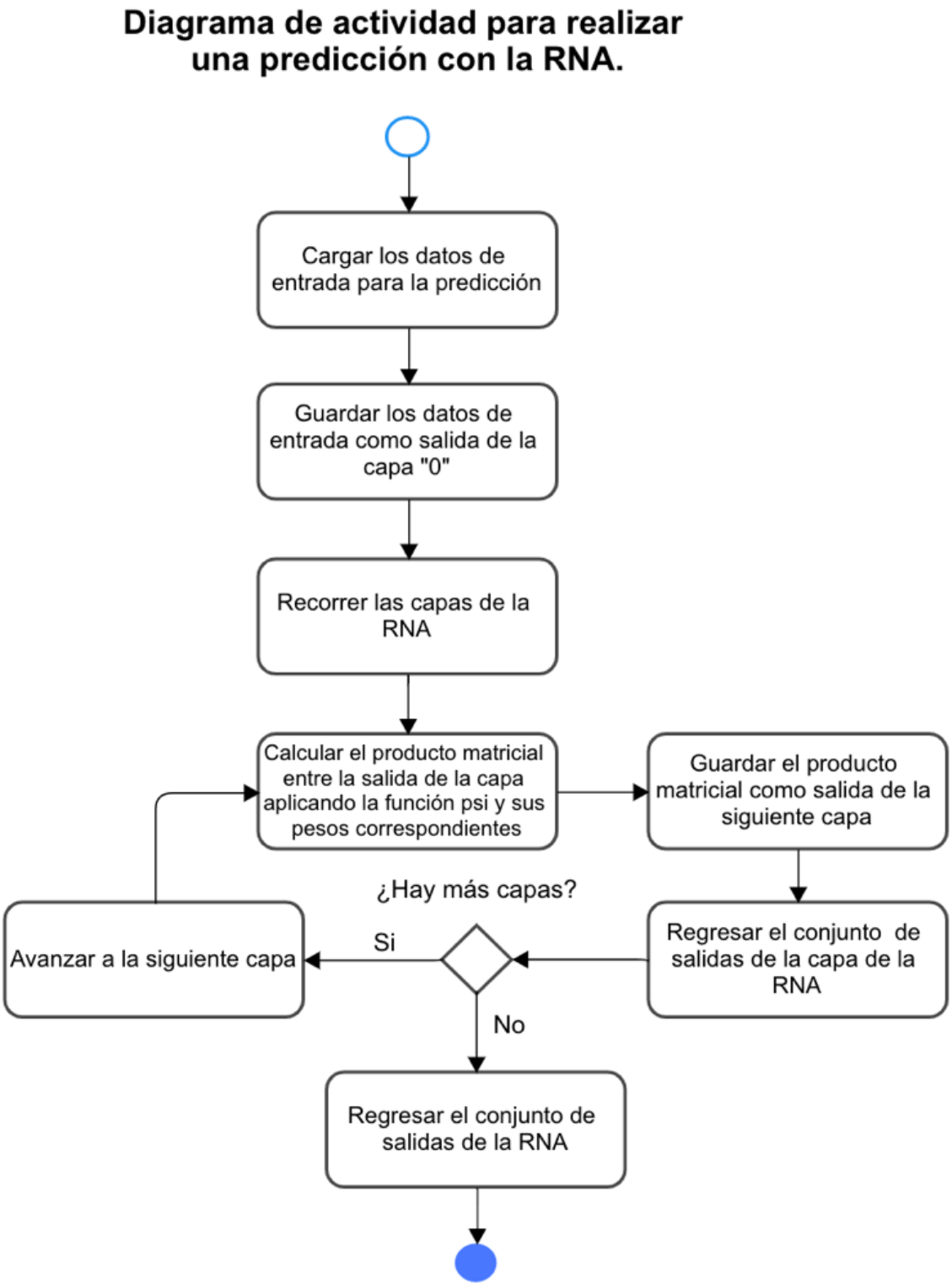

Figura 17. Diagrama de actividad para realizar una predicción con la RNA.

El entrenamiento de la RNA es el proceso más largo y tardado, en éste se tiene un subproceso que es el de la predicción, y este junto con los demás pasos se realiza de forma cíclica hasta que se terminen las iteraciones necesarias para el entrenamiento, dentro de este proceso hay otros dos que también recorren un cierto número de repeticiones en este caso todas las capas de la RNA formando un ciclo que no termina hasta que se llegue a la última capa. Ver diagrama de la Figura 18.

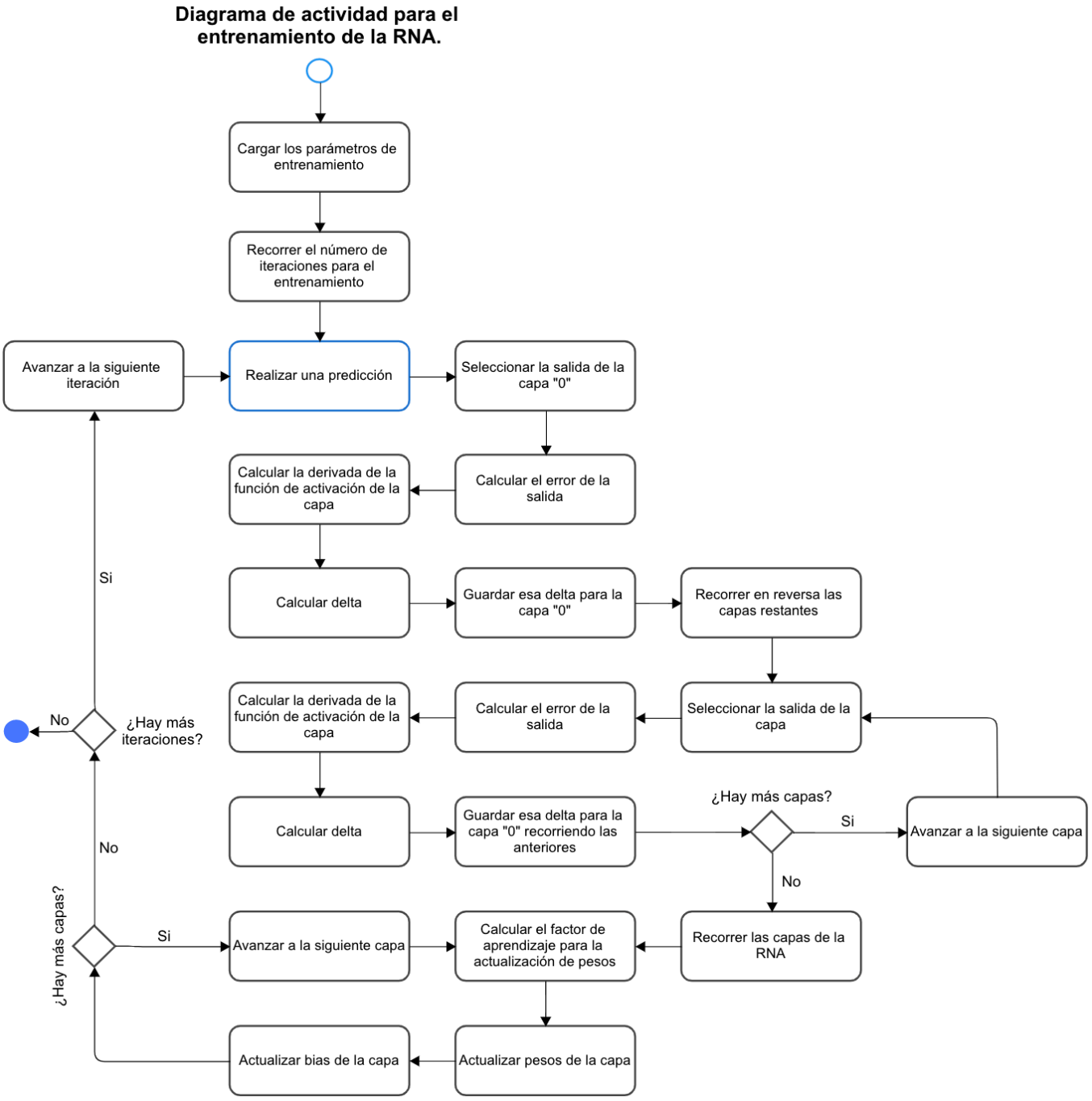

Figura 18. Diagrama de actividad para el entrenamiento de la RNA.

Como se puede observar en este diagrama, un bloque que se resalta con la línea de color azul es el de “Realizar una predicción” que es un subproceso interno y los pasos de esto se observan en el diagrama de la Figura 17.

Por último, se observan una serie de pasos lineales en el diagrama de la Figura 19, estas actividades se realizan para lograr mostrar los resultados de las predicciones en un plano con tres graficas de puntos.

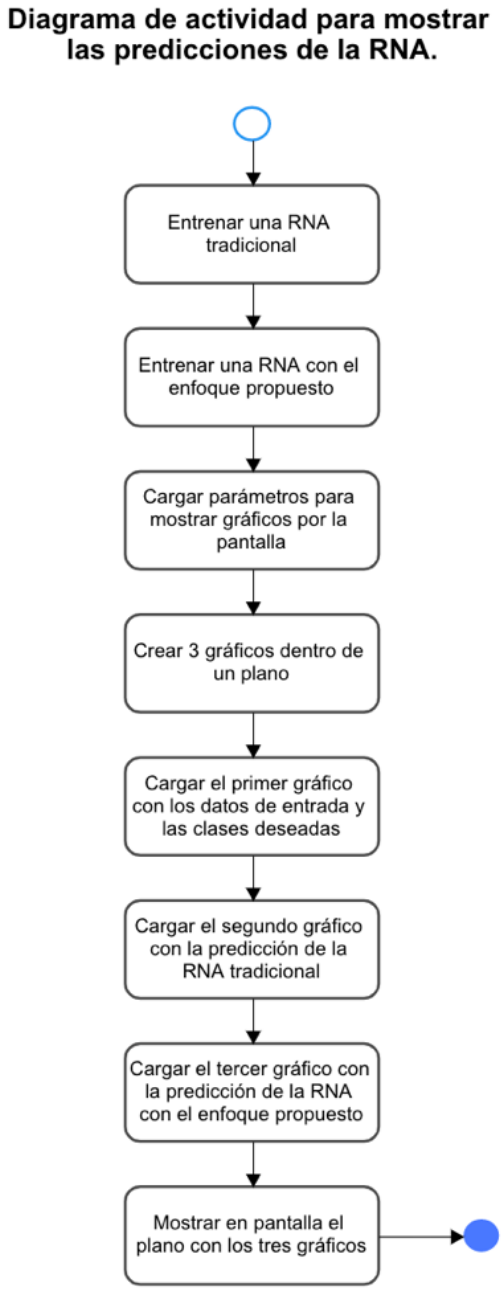

Figura 19. Diagrama de actividad para mostrar las predicciones de la RNA.

Con todos estos diagramas ya se puede visualizar la implementación de la RNA con el enfoque propuesto. Para realizar la experimentación y testeo de la RNA estos diagramas se codificaron en el lenguaje de programación Python que se pueden ver en el Anexo 1.

CAPÍTULO VI ESQUEMA TRADICIONAL VS ESQUEMA PROPUESTO

Este es el último apartado del documento, y en él se encuentra el proceso de desarrollo del proyecto y las pruebas realizadas. Este capítulo está dividido en cuatro secciones que permiten entender cómo se realizó la experimentación del nuevo enfoque propuesto en la Red Neuronal Artificial (RNA). La primera sección describe la creación de los *Datasets* necesarios para la experimentación, en la segunda sección se describen los parámetros de los experimentos, los resultados de éstos se muestran en la tercera sección, por último, el análisis de los resultados se puede observar en la sección cuatro de este apartado.

6.1 CREACIÓN DE *DATASETS* SINTÉTICOS

Para el testeo de la RNA y verificar si el enfoque propuesto muestra resultados satisfactorios, que permitan reducir el número de capas o de neuronas por capa, se crearon de manera manual cuatro *Datasets* sintéticos, cada uno con un comportamiento diferente dependiendo una función que se utilizara dentro de la RNA propuesta. Los cuatro *Datasets* constan de 500 pares de elementos (*X* - *Y*) que especifican un punto en un plano cartesiano, los primeros 250 puntos están etiquetados con el "*0*" y los otros 250 puntos con el "*1*", teniendo así dos clases que tendrá que clasificar la RNA.

Los *Datasets* creados son descritos a continuación:

1. ***Dataset* lineal:**

   En este *Dataset* los valores de *X* se encuentran en un rango de -50 y 50 y los valores de *Y* son el resultado de aplicar la función $f = 2x$. Es importante mencionar que para lograr una separación entre las clases "*0*" y "*1*", a la

función de los puntos de la última clase se sumaron 10 obteniendo la función $f = 2x + 10$.

Tabla 1. Configuración de Dataset número 1 (lineal).

Clase	Cantidad de puntos	Valores en X	Valores en Y
0	250	Número aleatorio en un rango de -50 y 50	$f = 2x$
1	250	Número aleatorio en un rango de -50 y 50	$f = 2x + 10$

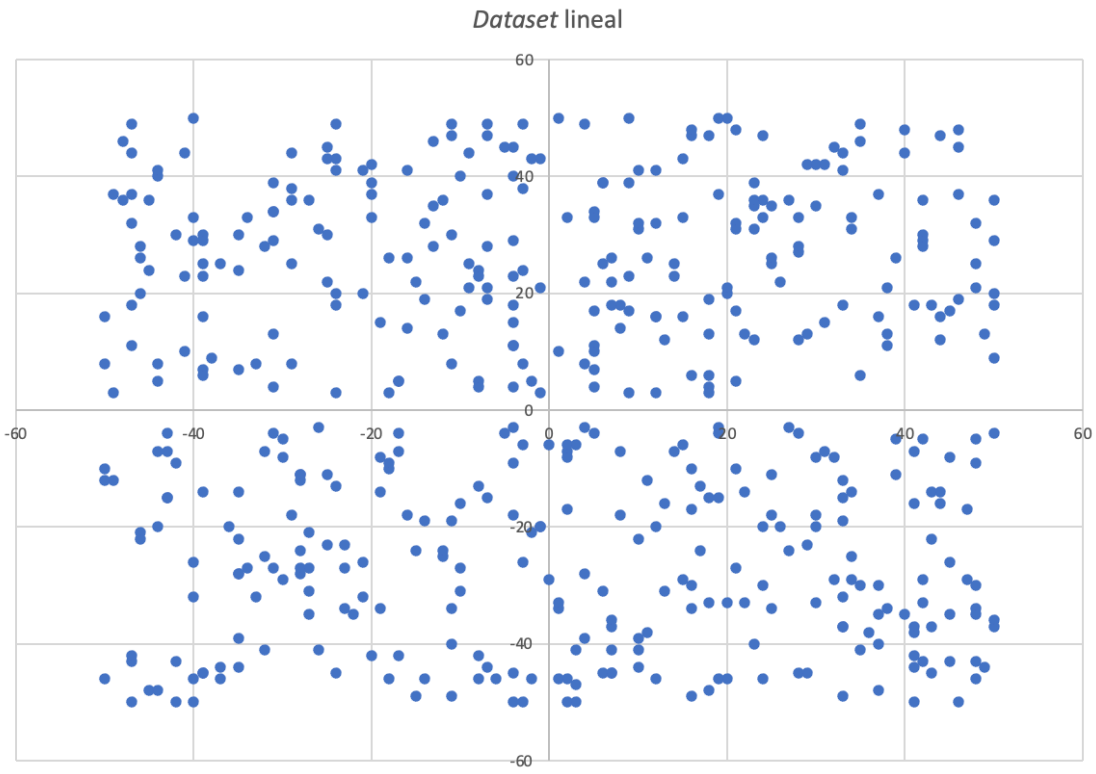

Figura 20. Gráfica de distribución de puntos del Dataset número 1 (lineal).

2. *Dataset* cuadrático:

Para este *Dataset* la función utilizada es una cuadrática $f = x^2$ donde en la clase "*1*" muestra una separación con respecto a la "*0*" de 5 puntos esto se representa en la función $f = x^2 + 5$.

Tabla 2. Configuración de Dataset número 2 (cuadrática).

Clase	Cantidad de puntos	Valores en X	Valores en Y
0	250	Número aleatorio en un rango de -3 y 3	$f = x^2$
1	250	Número aleatorio en un rango de -3 y 3	$f = x^2 + 5$

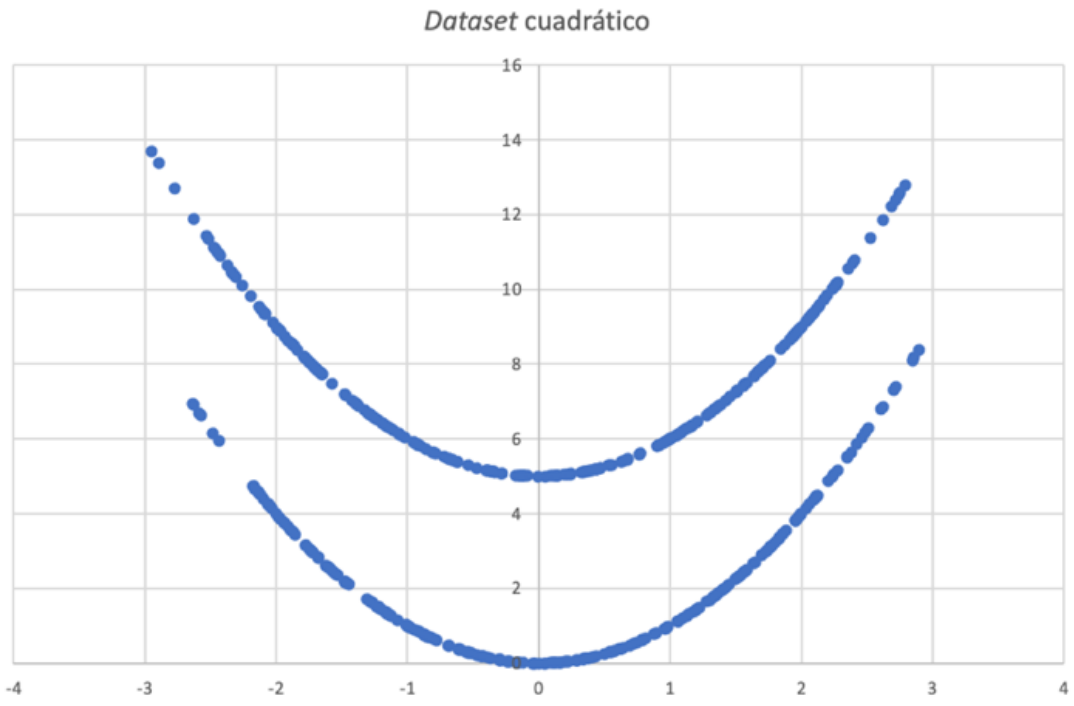

Figura 21. Gráfica de distribución de puntos del Dataset número 2 (cuadrática).

3. *Dataset* coseno:

La mitad de los puntos de la clase *Y* de este *Dataset* son afectados por la función $f = \cos(x)$, mientras que la otra mitad es afectada por $f = \cos(x) + 1$ logrando así una separación entre las dos clases "*0*" y "*1*".

Tabla 3. Configuración de Dataset número 3 (coseno).

Clase	Cantidad de puntos	Valores en X	Valores en Y
0	250	Número aleatorio en un rango de -4 y 1	$f = \cos(x)$
1	250	Número aleatorio en un rango de -4 y 1	$f = \cos(x) + 1$

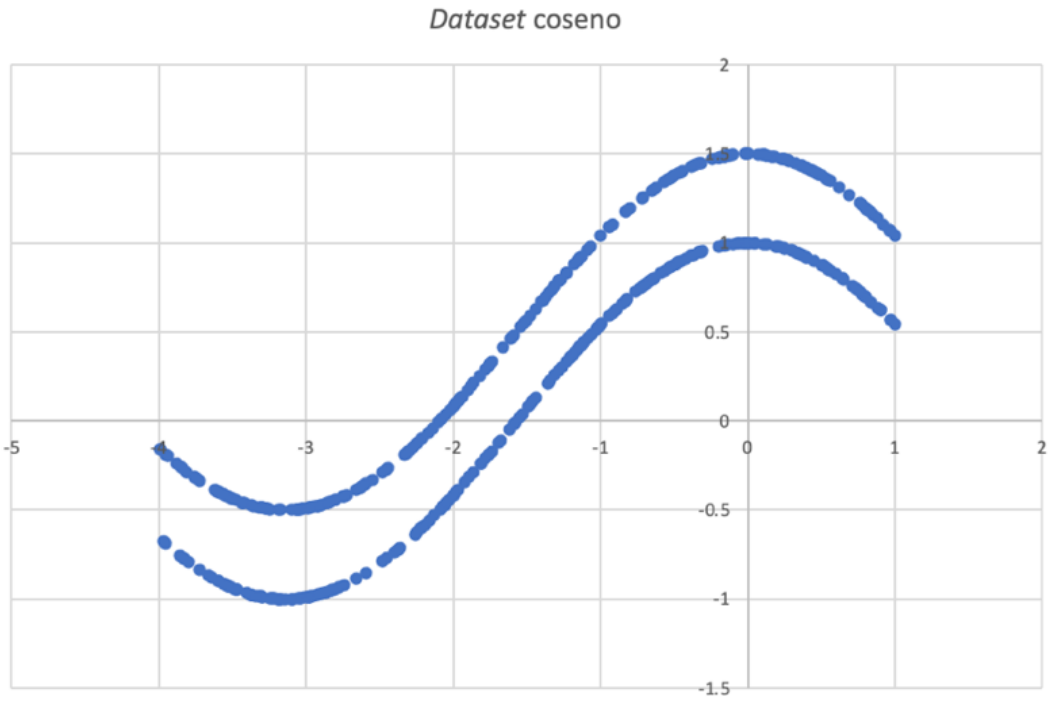

Figura 22. Gráfica de distribución de puntos del Dataset número 3 (coseno).

4. *Dataset* tangente:

Para el ultimo *Dataset* se utilizó la función $f = \tan(x)$ para el comportamiento de los puntos en el eje de *Y*. Para lograr una separación entre las clases "*0*" y "*1*" a los últimos 250 puntos del eje *Y* se agregó un punto de separación obteniendo la función $f = \tan(x) + 1$.

Tabla 4. Configuración de Dataset número 4 (tangente).

Clase	Cantidad de puntos	Valores en X	Valores en Y
0	250	Número aleatorio en un rango de -4 y 1	$f = \tan(x)$
1	250	Número aleatorio en un rango de -4 y 1	$f = \tan(x) + 1$

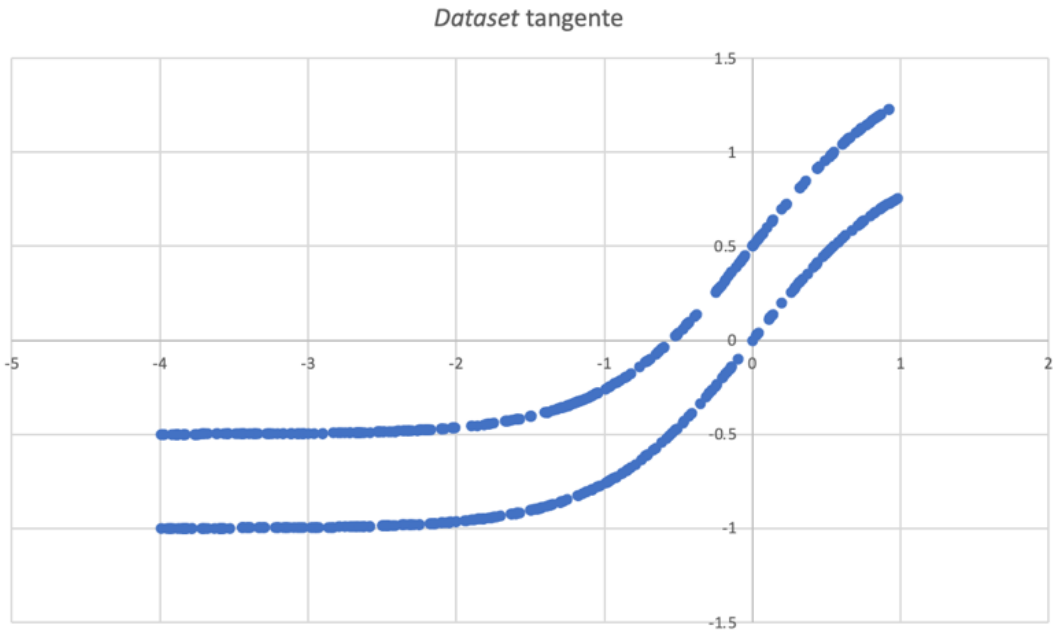

Figura 23. Gráfica de distribución de puntos del Dataset número 4 (tangente).

También se utilizó un *Dataset* proporcionado por scikit-learn, este también tiene dos clases que están etiquetadas con "*0*" y "*1*", lo que se adecua a los *Datasets* sintéticos, en este caso los puntos de una clase forman un círculo que se encuentra dentro de otro circulo formado por los puntos de la otra clase. Este Dataset se nombró como **"*Dataset* circular"**.

Tabla 5. Configuración de Dataset número 5 (circular).

Clase	Cantidad de puntos
0	250
1	250

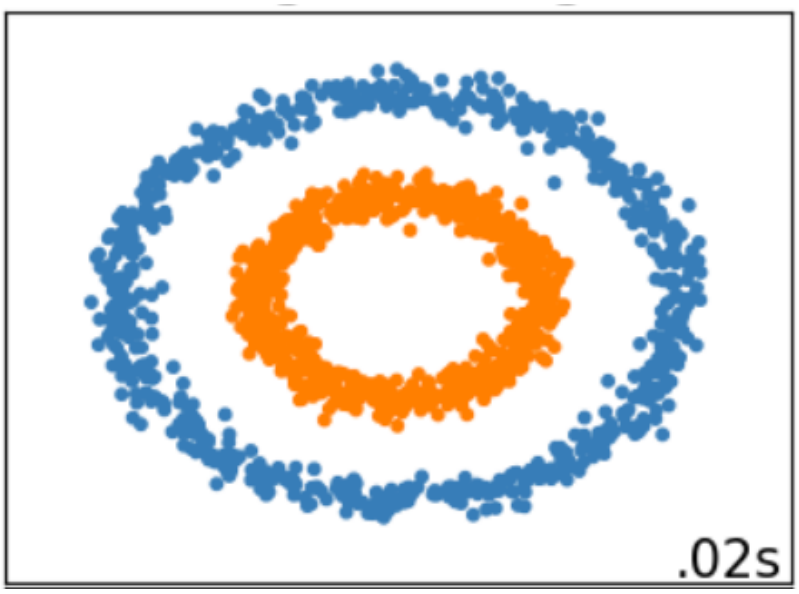

Figura 24. Gráfica de distribución de puntos del Dataset número 5 (circular).

En la Tabla 6 se muestra una guía de los *Datasets*, esta información proporciona una referencia para el diseño experimental.

Tabla 6. Lista de Datasets de referencia para el diseño experimental.

Número	Nombre	Cantidad de puntos	Origen
1	*Dataset* lineal	500	sintético
2	*Dataset* cuadrático	500	sintético
3	*Dataset* coseno	500	sintético
4	*Dataset* tangente	500	sintético
5	*Dataset* circular	500	scikit-learn

6.2 DISEÑO EXPERIMENTAL

Para poner a prueba la RNA con el enfoque propuesto y ver su rendimiento en comparación de una RNA tradicional se proponen un total de 5 experimentos, 1 por cada *Dataset* que se muestra en la Tabla 6, estos experimentos buscan comparar el rendimiento de las RNAs centrándose en el porcentaje de clasificación que arrojen.

Es importante mencionar que se realizó un estudio empírico experimental exhaustivo para determinar la mejor configuración para cada experimento, para esto se tomó en cuenta el número de capas, el número de neuronas y el número de iteraciones.

A continuación, se describen los cinco experimentos y sus configuraciones:

Diseño experimental número 1:

Para el primer experimento se utiliza el *Dataset* lineal, éste es el más sencillo de clasificar por lo que la RNA tradicional y la propuesta no necesitan capas ocultas, se pretende que este experimento demuestre que el enfoque propuesto para la RNA puede funcionar exactamente igual a la tradicional si como función ψ que se encuentra en la sección (4.3 MODIFICACIÓN AL ALGORITMO BACKPROPAGATION) se utiliza $f = x$. Para ver el diseño completo del experimento véase la Tabla 7.

Tabla 7. Diseño experimental número 1.

	RNA Tradicional				RNA Propuesta				
Dataset	**Número de iteraciones**	**Learning rate**	**Capas**	**Neuronas por capa**	**Número de iteraciones**	**Learning rate**	**Función psi**	**Capas**	**Neuronas por capa**
Dataset lineal	100	0.05	Entrada	2	100	0.05	$f = x$	Entrada	2
			Salida	1				Salida	1

Diseño experimental número 2:

Para el segundo experimento el *Dataset* a utilizar es el cuadrático, en este la función ψ ($f = x^2$) permite ver el objetivo del enfoque propuesto para la RNA, donde se puede observar que la RNA tradicional tiene una capa oculta con dos neuronas mientras que la RNA propuesta tiene una sola neurona en su capa oculta. Ver Tabla 8.

Tabla 8. Diseño experimental número 2.

	RNA Tradicional				RNA Propuesta				
Dataset	**Número de iteraciones**	**Learning rate**	**Capas**	**Neuronas por capa**	**Número de iteraciones**	**Learning rate**	**Función psi**	**Capas**	**Neuronas por capa**
Dataset cuadrático	1000	0.05	Entrada	2	1000	0.05	$f = x^2$	Entrada	2
			Oculta	2				Oculta	1
			Salida	1				Salida	1

Diseño experimental número 3:

En la Tabla 9 se observa el diseño experimental número 3, nuevamente se incrementa el número de neuronas en la capa oculta en la RNA tradicional con respecto a la propuesta, en este experimento se decidió utilizar la función $f = \tan(x)$ en la RNA propuesta para intentar clasificar los puntos del *Dataset* tangente.

Tabla 9. Diseño experimental número 3.

	RNA Tradicional				RNA Propuesta				
Dataset	**Número de iteraciones**	**Learning rate**	**Capas**	**Neuronas por capa**	**Número de iteraciones**	**Learning rate**	**Función psi**	**Capas**	**Neuronas por capa**
Dataset tangente	1000	0.05	Entrada	2	1000	0.05	$f = \tan(x)$	Entrada	2
			Oculta	4				Oculta	1
			Salida	1				Salida	1

Diseño experimental número 4:

En la configuración de este experimento se puede observar que en la RNA tradicional hay un incremento no solo en el número de neuronas sino, en el número de capas ocultas con respecto a la RNA con el enfoque propuesto, esto porque el *Dataset* circular requiere más conexiones de nodos para la clasificación correcta, y estas conexiones se pueden remplazar utilizando la función $f = x^2$ cómo se observa en la Tabla 10.

Tabla 10. Diseño experimental número 4.

	RNA Tradicional				RNA Propuesta				
Dataset	**Número de iteraciones**	**Learning rate**	**Capas**	**Neuronas por capa**	**Número de iteraciones**	**Learning rate**	**Función psi**	**Capas**	**Neuronas por capa**
Dataset circular	2000	0.05	Entrada	2	2000	0.05	$f = x^2$	Entrada	2
			Oculta	4				Oculta	1
			Oculta	2				Salida	1
			Salida	1					

Diseño experimental número 5:

La configuración del último experimento se puede observar en la Tabla 11, en este podemos destacar que tanto la RNA tradicional como la RNA propuesta tienen dos capas ocultas esto se debe a el Dataset coseno, la clasificación de este es un poco complicada y requiere de bastantes conexiones de nodos, sin embargo, podemos ver que la RNA propuesta sigue logrando su objetivo reduciendo el número de neuronas en las capas ocultas, para lograrlo es necesario utilizar la función psi, $f = sen(x)$ aunque para la creación del *Dataset* se utilizó la función $f = \cos(x)$.

Tabla 11. Diseño experimental número 5.

	RNA Tradicional				RNA Propuesta				
Dataset	**Número de iteraciones**	**Learning rate**	**Capas**	**Neuronas por capa**	**Número de iteraciones**	**Learning rate**	**Función psi**	**Capas**	**Neuronas por capa**
Dataset coseno	10000	0.05	Entrada	2	10000	0.05	$f = sen(x)$	Entrada	2
			Oculta	6				Oculta	2
			Oculta	2					
			Salida	1				Salida	1

6.3 RESULTADOS

El objetivo de la RNA con el enfoque propuesto es reducir el número de capas y de neuronas necesarias para la clasificación en comparación con la RNA tradicional, con el fin de mostrar experimentalmente esta capacidad en la RNA se realizó la ejecución de los cinco experimentos con sus configuraciones correspondientes. En cada experimento se mostrará lo siguiente:

- Una figura con un plano que contendrá tres 3 gráficos, el primero mostrará la clasificación real de los puntos del *Dataset*, en el segundo grafico la clasificación realizada por la RNA tradicional y en el ultimo la clasificación realizada por la RNA propuesta.
- Una tabla con 35 resultados de la ejecución del experimento, esta tabla contendrá los puntos clasificados por clase, el accuracy y el tiempo de ejecución comparando la RNA tradicional y la RNA propuesta.

Resultado del experimento número 1:

Como se muestra en la Tabla 7 las configuraciones en este experimento son muy simples, cabe mencionar que la función ψ no altera los datos de entrada. Las clases tienen una separación lineal y es sencillo para la RNA tradicional y para la RNA propuesta clasificarlas perfectamente como se muestra en la Figura 25.

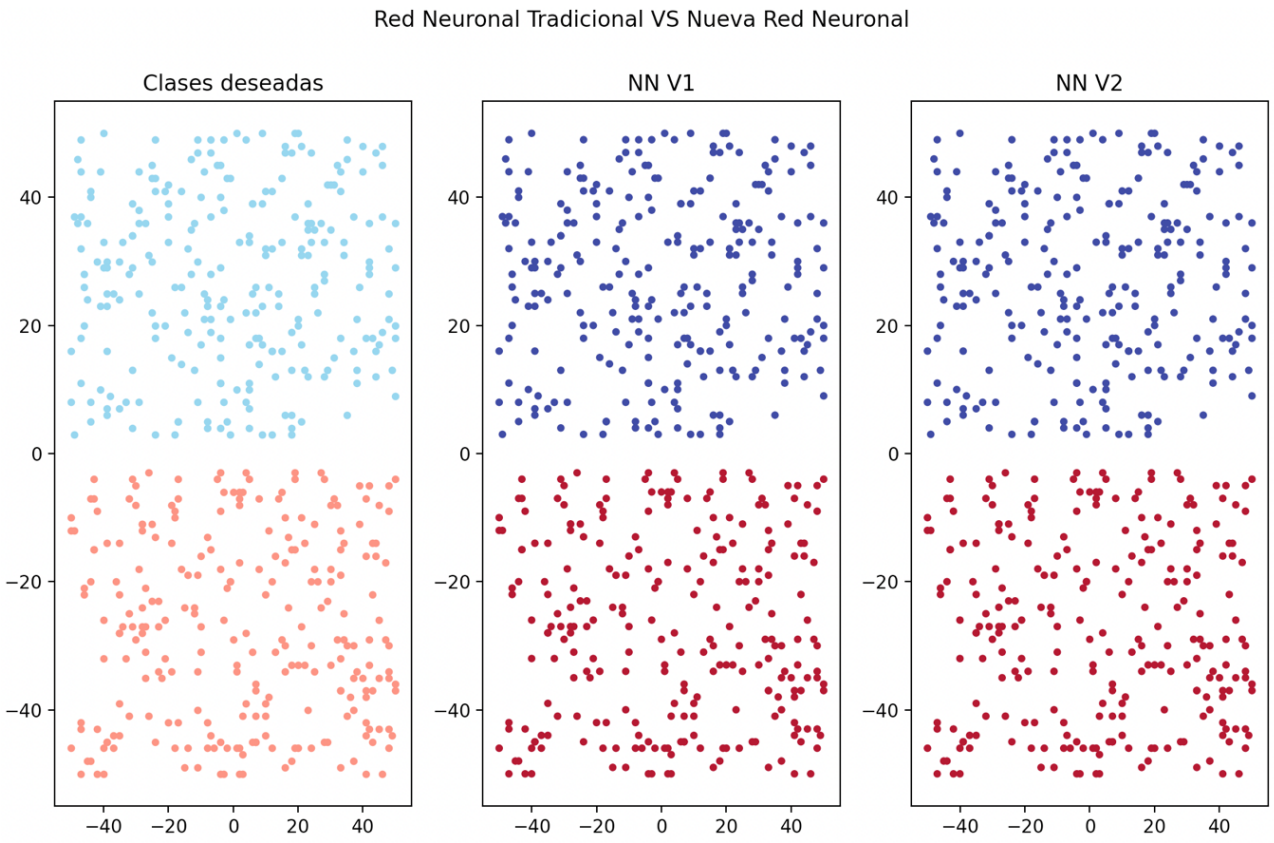

Figura 25. Ejemplo de predicción de la RNA tradicional y la RNA propuesta (Experimento número 1).

En la Tabla 12 se muestran los 35 resultados de la ejecución del primer experimento. De cada ejecución se captura los puntos clasificados, el accuracy y el tiempo de ejecución.

Tabla 12. Resultados de la ejecución del experimento número 1.

		RNA Tradicional				RNA Propuesta			
	Número de ejecución	Puntos clasificados de la clase 0 (250)	Puntos clasificados de la clase 1 (250)	Accuracy	Tiempo de ejecución (s)	Puntos clasificados de la clase 0 (250)	Puntos clasificados de la clase 1 (250)	Accuracy	Tiempo de ejecución (s)
Experimento número 1	1	250	250	100%	0.008	250	250	100%	0.016
	2	250	250	100%	0.006	250	250	100%	0.012
	3	250	250	100%	0.005	250	250	100%	0.014
	4	250	250	100%	0.005	250	250	100%	0.01
	5	250	250	100%	0.006	250	250	100%	0.013
	6	250	250	100%	0.005	250	250	100%	0.012
	7	250	250	100%	0.005	250	250	100%	0.01
	8	250	250	100%	0.006	250	250	100%	0.012
	9	250	250	100%	0.007	250	250	100%	0.015
	10	250	250	100%	0.007	250	250	100%	0.012
	11	250	250	100%	0.006	250	250	100%	0.013
	12	250	250	100%	0.005	250	250	100%	0.013
	13	250	250	100%	0.006	250	250	100%	0.011
	14	250	250	100%	0.005	250	250	100%	0.013
	15	250	250	100%	0.005	250	250	100%	0.011
	16	250	250	100%	0.007	250	250	100%	0.014
	17	250	250	100%	0.006	250	250	100%	0.012
	18	250	250	100%	0.005	250	250	100%	0.012
	19	250	250	100%	0.007	250	250	100%	0.013
	20	250	250	100%	0.005	250	250	100%	0.012
	21	250	250	100%	0.005	250	250	100%	0.012
	22	250	250	100%	0.006	250	250	100%	0.014
	23	250	250	100%	0.007	250	250	100%	0.014
	24	250	250	100%	0.007	250	250	100%	0.011
	25	250	250	100%	0.005	250	250	100%	0.012
	26	250	250	100%	0.005	250	250	100%	0.011
	27	250	250	100%	0.007	250	250	100%	0.015
	28	250	250	100%	0.005	250	250	100%	0.01
	29	250	250	100%	0.006	250	250	100%	0.013
	30	250	250	100%	0.006	250	250	100%	0.014
	31	250	250	100%	0.006	250	250	100%	0.013
	32	250	250	100%	0.007	250	250	100%	0.013
	33	250	250	100%	0.006	250	250	100%	0.012
	34	250	250	100%	0.005	250	250	100%	0.012
	35	250	250	100%	0.007	250	250	100%	0.013
Media		250.00	250.00	100%	0.006	250.00	250.00	100%	0.013
Mediana		250	250	100%	0.006	250	250	100%	0.012
Desviación estándar		0	0	0.000	0.001	0	0	0.000	0.001

Resultado del experimento número 2:

En el experimento 2 el *Dataset* que se utiliza es el cuadrático, en este ya se puede observar los beneficios de utilizar una función psi, esta función altera los datos de entrada y permite imitar el comportamiento de la RNA tradicional con menos neuronas en la capa oculta. Un ejemplo de esta predicción se puede observar en la Figura 26.

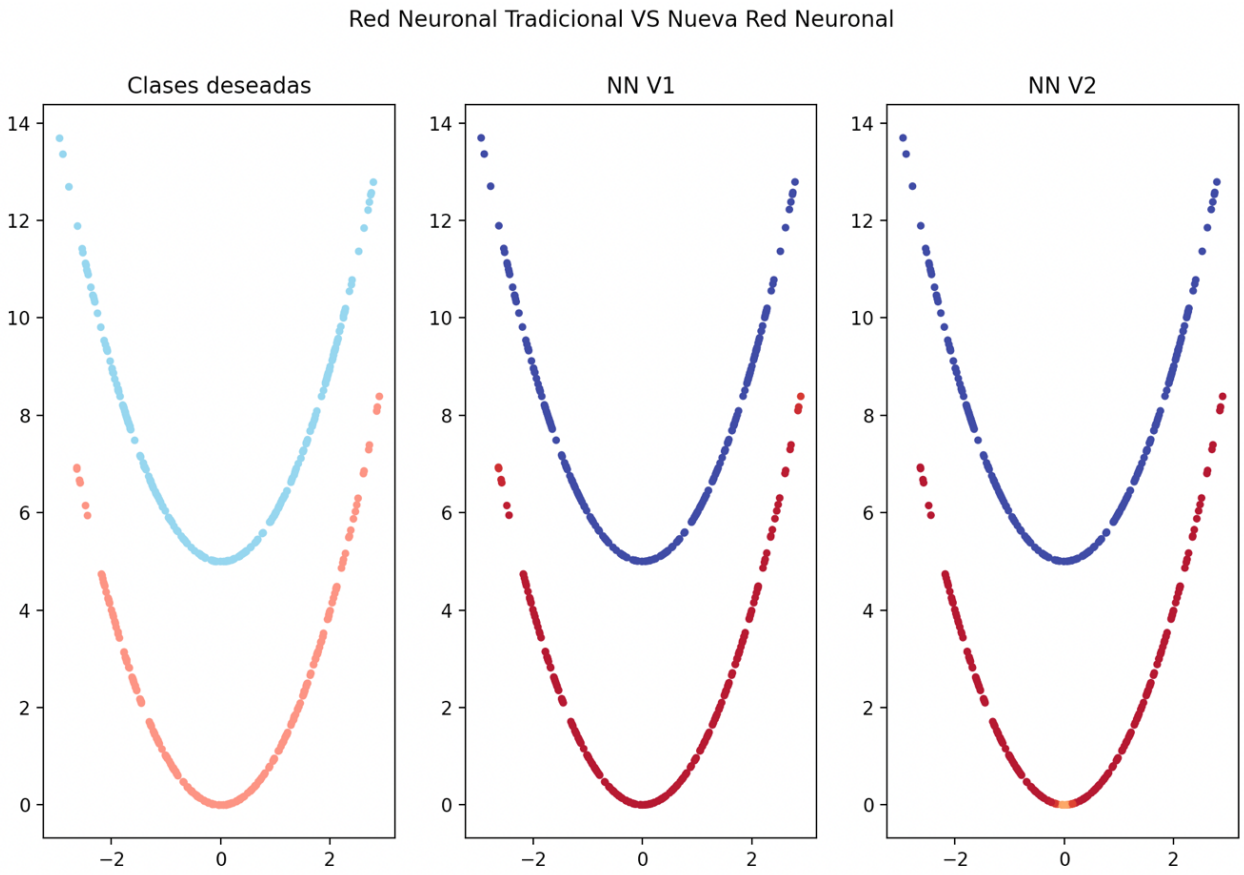

Figura 26. Ejemplo de predicción de la RNA tradicional y la RNA propuesta (Experimento número 2).

Los resultados de las 35 ejecuciones del segundo experimento se muestran en la Tabla 13.

Tabla 13. Resultados de la ejecución del experimento número 2.

		RNA Tradicional				RNA Propuesta			
	Número de ejecución	Puntos clasificados de la clase 0 (250)	Puntos clasificados de la clase 1 (250)	Accuracy	Tiempo de ejecución (s)	Puntos clasificados de la clase 0 (250)	Puntos clasificados de la clase 1 (250)	Accuracy	Tiempo de ejecución (s)
Experimento número 2	1	250	250	100%	0.149	250	250	100%	0.294
	2	250	250	100%	0.129	250	250	100%	0.268
	3	170	250	84%	0.149	250	250	100%	0.292
	4	79	250	66%	0.148	250	250	100%	0.259
	5	72	250	64%	0.148	243	250	99%	0.258
	6	129	250	76%	0.159	90	250	68%	0.339
	7	250	250	100%	0.147	250	250	100%	0.253
	8	250	250	100%	0.171	250	250	100%	0.262
	9	250	250	100%	0.149	250	250	100%	0.322
	10	250	250	100%	0.147	250	250	100%	0.29
	11	250	0	50%	0.139	250	250	100%	0.279
	12	250	0	50%	0.155	250	250	100%	0.275
	13	250	250	100%	0.139	250	250	100%	0.268
	14	250	0	50%	0.133	250	250	100%	0.261
	15	250	250	100%	0.149	250	250	100%	0.294
	16	250	250	100%	0.14	250	250	100%	0.28
	17	211	189	80%	0.173	250	250	100%	0.275
	18	250	250	100%	0.144	250	250	100%	0.305
	19	250	250	100%	0.144	250	250	100%	0.269
	20	250	250	100%	0.137	250	250	100%	0.274
	21	250	250	100%	0.159	250	250	100%	0.286
	22	177	250	85%	0.137	88	250	68%	0.252
	23	250	250	100%	0.141	250	250	100%	0.288
	24	250	250	100%	0.166	250	250	100%	0.323
	25	145	250	79%	0.144	250	250	100%	0.275
	26	250	0	50%	0.144	250	250	100%	0.271
	27	250	250	100%	0.14	73	250	65%	0.288
	28	250	250	100%	0.16	250	250	100%	0.289
	29	70	250	64%	0.139	250	250	100%	0.272
	30	250	250	100%	0.156	250	250	100%	0.29
	31	250	250	100%	0.14	250	250	100%	0.274
	32	174	250	85%	0.353	250	250	100%	0.286
	33	250	250	100%	0.157	250	250	100%	0.324
	34	250	250	100%	0.145	250	0	50%	0.278
	35	124	250	75%	0.142	250	250	100%	0.29
Media		217.17	219.69	87%	0.153	235.54	242.86	96%	0.283
Mediana		250	250	100%	0.147	250	250	100%	0.279
Desviación estándar		58	81	0.179	0.036	47	42	0.123	0.021

Resultados del experimento número 3:

Nuevamente en este experimento la RNA propuesta disminuye el número de neuronas en la capa oculta, pero esta logra clasificar los puntos del *Dataset* de manera exitosa gracias a la función psi. Esto se puede observar en la Figura 27.

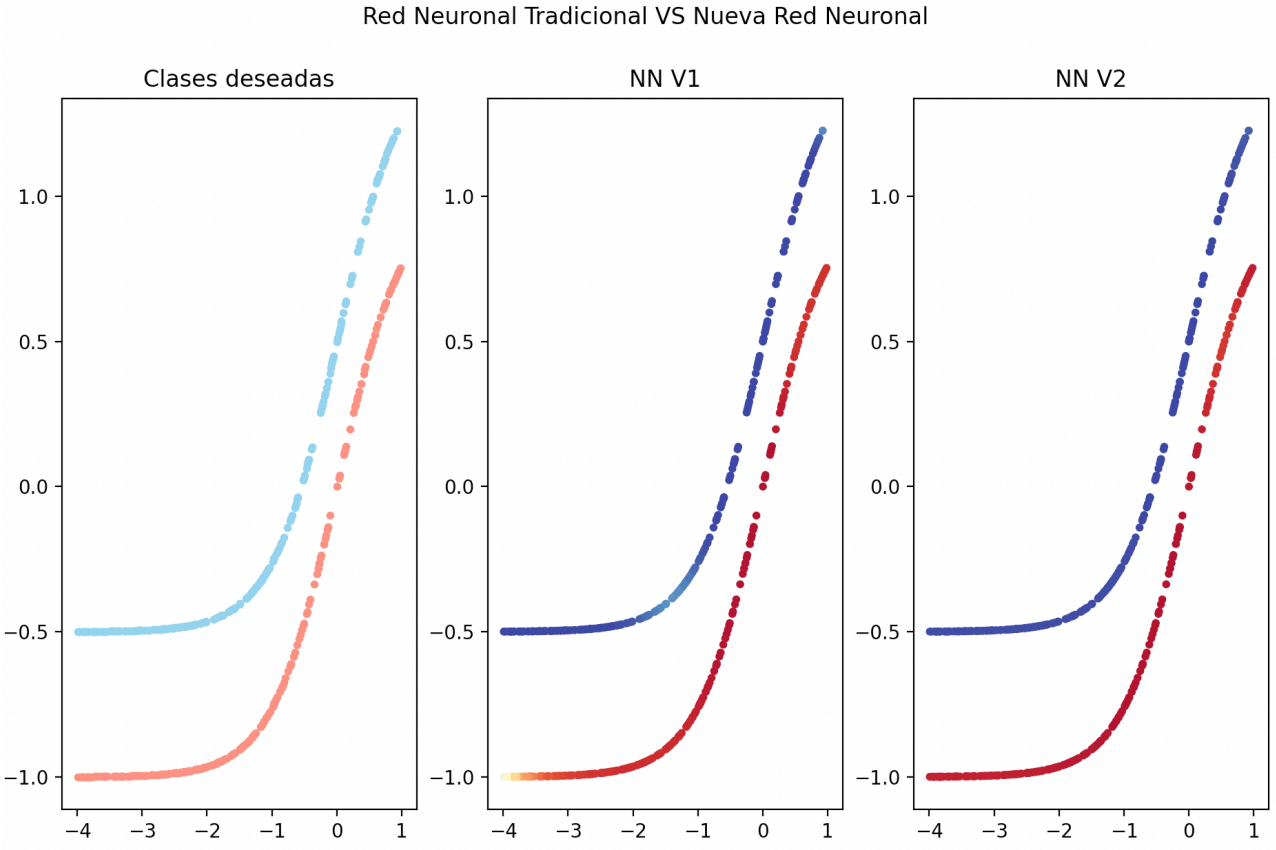

Figura 27. . Ejemplo de predicción de la RNA tradicional y la RNA propuesta (Experimento número 3).

En la Tabla 14 se muestran los resultados de las 35 ejecuciones del tercer experimento.

Tabla 14. Resultados de la ejecución del experimento número 3.

		RNA Tradicional				RNA Propuesta			
	Número de ejecución	Puntos clasificados de la clase 0 (250)	Puntos clasificados de la clase 1 (250)	Accuracy	Tiempo de ejecución (s)	Puntos clasificados de la clase 0 (250)	Puntos clasificados de la clase 1 (250)	Accuracy	Tiempo de ejecución (s)
Experimento número 3	1	248	250	100%	0.172	250	250	100%	0.263
	2	250	250	100%	0.156	217	250	93%	0.241
	3	250	250	100%	0.228	250	250	100%	0.316
	4	250	250	100%	0.285	250	250	100%	0.3
	5	250	250	100%	0.17	221	250	94%	0.263
	6	250	250	100%	0.178	250	250	100%	0.282
	7	250	250	100%	0.177	250	250	100%	0.258
	8	250	250	100%	0.166	250	250	100%	0.255
	9	250	250	100%	0.167	217	250	93%	0.259
	10	250	250	100%	0.171	250	250	100%	0.237
	11	250	250	100%	0.163	198	250	90%	0.271
	12	250	250	100%	0.173	250	250	100%	0.259
	13	250	250	100%	0.162	198	250	90%	0.254
	14	250	250	100%	0.163	228	250	96%	0.256
	15	250	250	100%	0.166	250	250	100%	0.257
	16	250	250	100%	0.154	250	250	100%	0.249
	17	250	250	100%	0.155	198	250	90%	0.24
	18	250	250	100%	0.164	199	250	90%	0.266
	19	250	250	100%	0.165	199	250	90%	0.249
	20	250	250	100%	0.165	250	250	100%	0.25
	21	250	250	100%	0.164	250	250	100%	0.252
	22	250	250	100%	0.169	250	250	100%	0.253
	23	246	250	99%	0.214	250	250	100%	0.302
	24	249	250	100%	0.174	250	250	100%	0.254
	25	250	250	100%	0.192	199	250	90%	0.294
	26	250	250	100%	0.175	250	250	100%	0.266
	27	250	250	100%	0.205	250	250	100%	0.29
	28	250	250	100%	0.197	250	250	100%	0.258
	29	250	250	100%	0.166	198	250	90%	0.28
	30	250	250	100%	0.171	235	250	97%	0.267
	31	250	250	100%	0.193	198	250	90%	0.305
	32	250	250	100%	0.197	199	250	90%	0.266
	33	250	250	100%	0.188	250	250	100%	0.274
	34	250	250	100%	0.176	199	250	90%	0.256
	35	250	250	100%	0.17	250	250	100%	0.261
Media		249.80	250.00	100%	0.179	231.51	250.00	96%	0.266
Mediana		250	250	100%	0.171	250	250	100%	0.259
Desviación estándar		1	0	0.002	0.025	23	0	0.046	0.019

Resultados del experimento número 4:

En el cuarto experimento se utiliza el único *Dataset* que no es sintético y el primero donde la RNA propuesta no solo reduce el número de neuronas en la capa oculta, sino que también reduce el número de capas, esta configuración se muestra en la Tabla 10. Una predicción de este experimento se puede observar en la Figura 28.

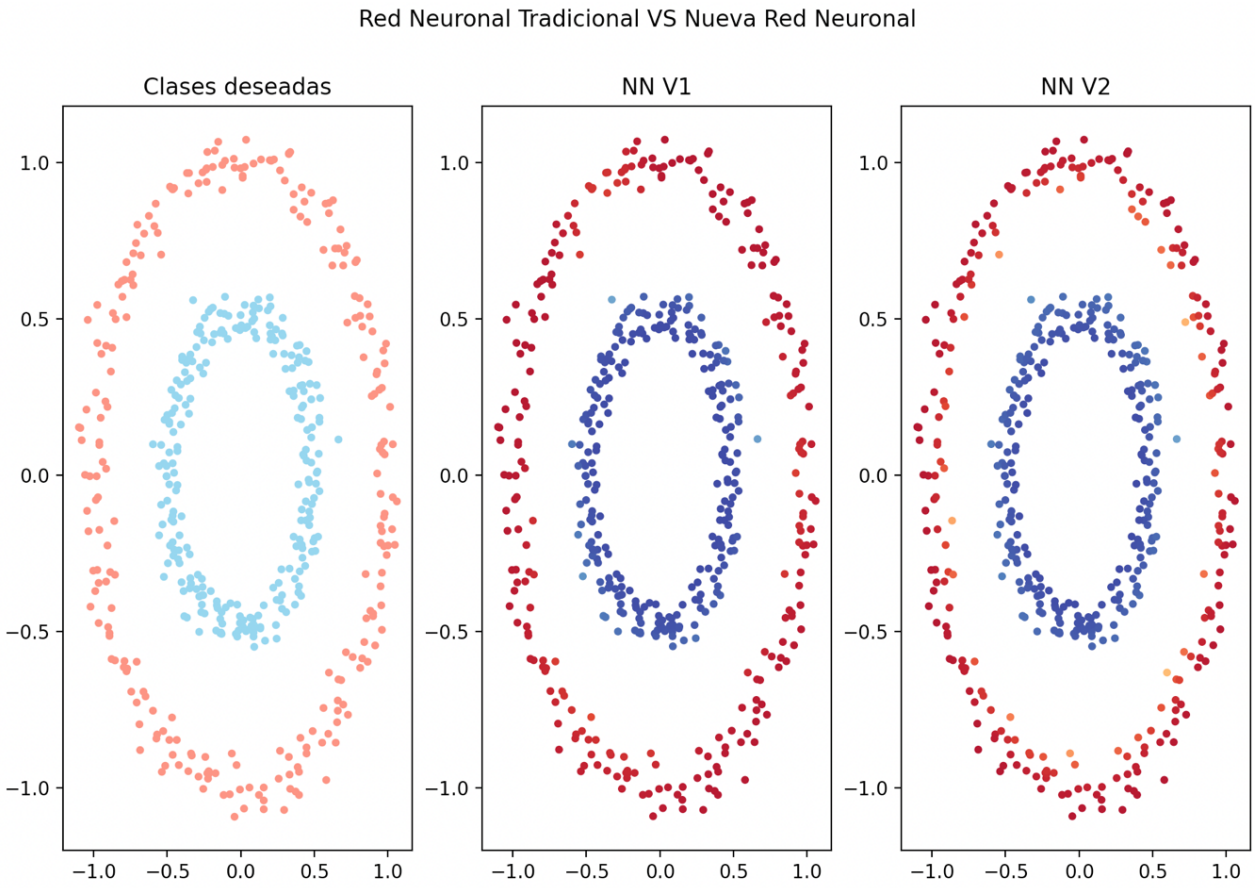

Figura 28. Ejemplo de predicción de la RNA tradicional y la RNA propuesta (Experimento número 4).

Para ver el rendimiento de la RNA con el enfoque propuesto en comparación de la RNA tradicional se puede observar la Tabla 15, en esta están las 35 ejecuciones del cuarto experimento.

Tabla 15. Resultados de la ejecución del experimento número 4.

		RNA Tradicional				RNA Propuesta			
	Número de ejecución	Puntos clasificados de la clase 0 (250)	Puntos clasificados de la clase 1 (250)	Accuracy	Tiempo de ejecución (s)	Puntos clasificados de la clase 0 (250)	Puntos clasificados de la clase 1 (250)	Accuracy	Tiempo de ejecución (s)
Experimento número 4	1	250	250	100%	0.516	250	250	100%	0.456
	2	250	250	100%	0.601	250	249	100%	0.498
	3	193	248	88%	0.493	250	250	100%	0.444
	4	250	250	100%	0.517	250	250	100%	0.515
	5	212	249	92%	0.601	250	250	100%	0.58
	6	234	250	97%	0.572	250	250	100%	0.518
	7	175	250	85%	0.561	250	250	100%	0.465
	8	250	250	100%	0.561	250	250	100%	0.449
	9	250	250	100%	0.536	250	249	100%	0.455
	10	250	250	100%	0.49	250	250	100%	0.45
	11	158	246	81%	0.507	250	249	100%	0.452
	12	250	250	100%	0.525	250	250	100%	0.671
	13	161	240	80%	0.521	250	250	100%	0.448
	14	167	250	83%	0.564	250	250	100%	0.477
	15	250	250	100%	0.533	250	250	100%	0.488
	16	184	244	86%	1.031	250	250	100%	0.625
	17	250	250	100%	1.524	250	250	100%	0.813
	18	250	250	100%	0.54	250	250	100%	0.457
	19	190	248	88%	0.528	250	250	100%	0.506
	20	250	250	100%	0.539	250	250	100%	0.496
	21	250	250	100%	0.617	250	250	100%	0.559
	22	154	241	79%	0.506	250	250	100%	0.445
	23	250	250	100%	0.545	250	250	100%	0.453
	24	161	244	81%	0.537	250	250	100%	0.473
	25	250	250	100%	0.566	250	250	100%	0.506
	26	192	245	87%	0.79	250	250	100%	0.615
	27	158	246	81%	0.695	250	249	100%	0.914
	28	250	250	100%	0.547	250	250	100%	0.523
	29	158	245	81%	0.502	250	249	100%	0.449
	30	250	250	100%	0.517	250	250	100%	0.576
	31	250	250	100%	0.536	250	250	100%	0.53
	32	158	246	81%	0.541	250	249	100%	0.579
	33	250	190	88%	0.518	250	250	100%	0.456
	34	250	250	100%	0.566	250	250	100%	0.502
	35	250	250	100%	0.732	250	250	100%	0.494
Media		218.71	246.63	93%	0.599	250.00	249.83	100%	0.524
Mediana		250	250	100%	0.54	250	250	100%	0.496
Desviación estándar		40	10	0.083	0.191	0	0	0.000	0.103

Resultados del experimento número 5:

Para el ultimo experimento las predicciones de las RNAs fueron las más complicadas, como se muestra en la Figura 29. Incluso se puede observar que la RNA con el enfoque propuesto, logra una clasificación menos precisa en comparación con la tradicional, sin embargo, hay que tomar en cuenta que utiliza menos capas ocultas y menos neuronas.

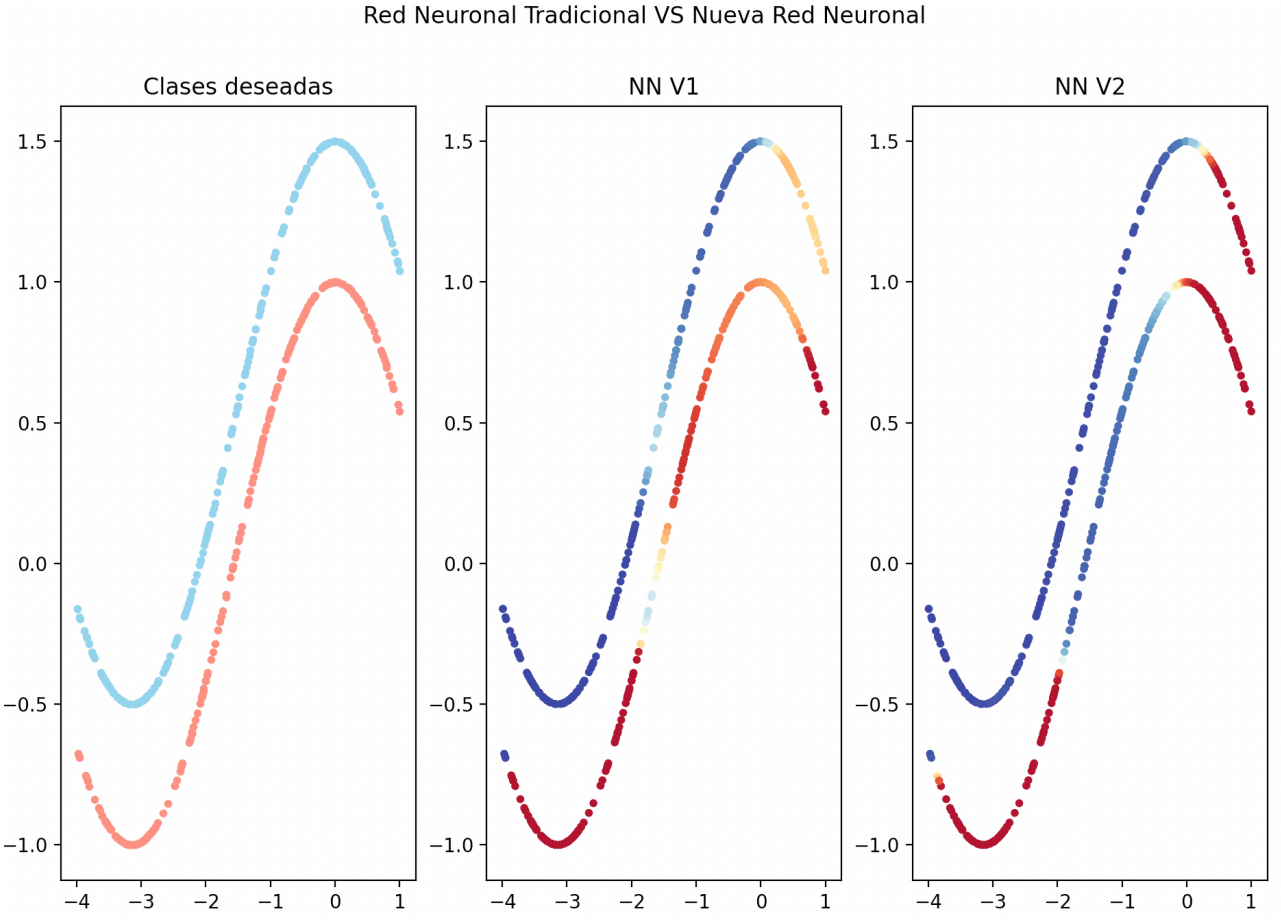

Figura 29. Ejemplo de predicción de la RNA tradicional y la RNA propuesta (Experimento número 5).

La comparación de las RNAs se puede observar en la Tabla 16, en esta se encuentran las 35 ejecuciones del quinto experimento.

Tabla 16. Resultados de la ejecución del experimento número 5.

		RNA Tradicional				RNA Propuesta			
	Número de ejecución	Puntos clasificados de la clase 0 (250)	Puntos clasificados de la clase 1 (250)	Accuracy	Tiempo de ejecución (s)	Puntos clasificados de la clase 0 (250)	Puntos clasificados de la clase 1 (250)	Accuracy	Tiempo de ejecución (s)
Experimento número 5	1	238	210	90%	2.839	162	210	74%	2.671
	2	247	163	82%	3.68	164	217	76%	3.596
	3	216	231	89%	4.9	170	217	77%	4.03
	4	247	134	76%	3.116	156	213	74%	2.868
	5	236	213	90%	3.092	159	213	74%	2.534
	6	249	210	92%	2.978	191	222	83%	2.789
	7	204	233	87%	3.077	159	213	74%	2.72
	8	219	237	91%	2.61	159	215	75%	2.506
	9	247	205	90%	2.766	163	220	77%	2.805
	10	248	208	91%	3.738	157	213	74%	3.443
	11	247	226	95%	4.557	190	231	84%	3.634
	12	247	250	99%	4.374	160	220	76%	3.315
	13	247	242	98%	3.618	164	216	76%	3.727
	14	250	156	81%	3.514	159	213	74%	3.405
	15	247	244	98%	3.601	169	217	77%	4.227
	16	247	234	96%	3.992	162	211	75%	3.216
	17	156	213	74%	4.135	247	171	84%	4.106
	18	215	217	86%	2.846	161	216	75%	2.93
	19	247	250	99%	3.972	162	213	75%	3.498
	20	227	224	90%	3.572	103	250	71%	3.202
	21	157	213	74%	4.04	161	208	74%	3.398
	22	230	207	87%	2.715	229	179	82%	3.074
	23	248	250	100%	3.493	161	213	75%	3.792
	24	247	236	97%	2.805	247	181	86%	3.941
	25	247	228	95%	4.368	157	217	75%	3.605
	26	250	192	88%	5.255	221	223	89%	3.122
	27	249	225	95%	4.878	162	216	76%	3.238
	28	183	207	78%	3.563	164	220	81%	3.522
	29	247	250	99%	3.51	156	230	77%	3.983
	30	217	217	87%	3.935	162	210	74%	3.686
	31	192	234	85%	4.494	158	234	75%	3.642
	32	247	250	99%	4.834	247	169	83%	3.673
	33	250	0	50%	3.469	250	100	70%	3.343
	34	194	235	86%	4.183	159	225	77%	3.241
	35	230	226	91%	4.014	247	155	80%	3.382
Media		230.54	213.43	89%	3.730	177.09	208.31	77%	3.368
Mediana		247	225	90%	3.618	162	215	76%	3.398
Desviación estándar		26	46	0.099	0.702	35	27	0.043	0.451

6.4 ANÁLISIS DE RESULTADOS

Con los resultados obtenidos tras la ejecución de los experimentos, se observa el desempeño de la RNA propuesta en comparación de la RNA tradicional, con estas observaciones se obtienen las siguientes conclusiones por experimento.

- **Experimento 1:**

  En el primer experimento la función ψ de la RNA propuesta no altero los datos de entrada por lo que esta funciono de la misma manera que la RNA tradicional y ambas obtuvieron un 100% de precisión al clasificar los puntos del primer *Dataset*, el tiempo de ejecución de la RNA tradicional fue un 0.0058s mientras que en la RNA propuesta obtuvo casi el doble, este fue de 0.012s.

- **Experimento 2:**

  La RNA tradicional de este experimento clasifico en promedio 217 puntos de los 250 de la clase “0” y 220 de la clase “1” obteniendo así un 87% de precisión, esta RNA se vio superada por el enfoque propuesto que implementa la función psi, obteniendo un 96% de precisión, donde en la clase “0” de los 250 puntos, clasifico correctamente 236 y en la clase “1” 242 puntos fueron clasificados correctamente. Aunque en precisión la RNA propuesta fue mejor, en cuestión de tiempo de ejecución la RNA tradicional realizaba las ejecuciones casi en la mitad del tiempo, sus tiempos promedio fueron 0.283s y 0.153s respectivamente.

- **Experimento 3:**

  Los resultados que arrojo el experimento número 3, muestran un mejor rendimiento de la RNA tradicional obteniendo en promedio un 100% de precisión en comparación de la RNA propuesta que obtuvo un 96%, pero si tomemos que en la RNA propuesta se utilizaron 3 neuronas menos que en la

RNA tradicional los resultados son bastante óptimos y muestran que utilizando una función ψ que refleje la geometría del problema de clasificación se pueden obtener resultados cercanos a los que se obtendrían al usar más neuronas en la red.

- **Experimento 4:**

  En el cuarto experimento se puede observar que la RNA propuesta obtuvo mejores resultados en la precisión y en el tiempo de ejecución, tratándose de la clasificación del *Dataset* la RNA tradicional obtuvo un 93% de precisión, esto porque de los 250 puntos de la clase "0" solo logró clasificar correctamente 219 y de los 250 puntos de la clase "1" clasificó correctamente 247, por otro lado, la RNA propuesta clasifico correctamente los 250 puntos de cada clase obteniendo un 100% de precisión. Respecto al tiempo de ejecución la RNA tradicional en promedio duro 0.599s y la RNA propuesta 0.524, no es tan grande la diferencia, pero se puede notar que el disminuir la cantidad de neuronas y/o capas impacta en el tiempo de ejecución.

- **Experimento 5:**

  Los resultados del último experimento fueron los peores, esto por la complejidad del *Dataset* que se utilizó, la RNA tradicional obtuvo un 89% de precisión y para esto utilizo dos capas ocultas, una con 6 neuronas y otra con 2, por otro lado, la RNA logro un 77% de precisión utilizando una sola capa oculta con una neurona, esta configuración es la que mejor balance entre arquitectura de la red y % de precisión obtuvo; por esta razón la RNA propuesta obtuvo un menor tiempo de ejecución que la RNA tradicional, 3.368s y 3.370s respectivamente. Cabe mencionar que en la RNA propuesta pudo haber obtenido mejores resultados de clasificación, pero esto implicaba utilizar más neuronas o capas ocultas en su arquitectura, lo que dejaría de lado el objetivo del enfoque propuesto en este trabajo.

CAPÍTULO VII CONCLUSIONES

De acuerdo con los objetivos específicos de este trabajo podemos concluir lo siguiente:

Se logró realizar la deducción matemática de los diferentes algoritmos de una Red Neuronal Artificial (RNA), esta deducción es la que permitió el desarrollo de la implementación de las RNAs utilizadas para este trabajo.

Respecto a la modificación del algoritmo de *Backpropagation* (BP), se efectuó de manera correcta logrando obtener el enfoque que se utilizó para comprar el rendimiento de una RNA tradicional con la RNA propuesta.

En el Capítulo V se realizó la implementación computacional de todos estos algoritmos, para esto se realizaron diagramas de bloque que muestran el proceso que se utilizó para el entrenamiento y el texteó de las RNAs. Logrando así cumplir con nuestro objetivo.

Se generaron cuatro *Datasets* para la experimentación en este trabajo, estos fueron creados a partir de la posición uniforme de 500 puntos en un plano cartesiano dónde la mitad de estos puntos obtuvo una diferencia con respecto a la otra mitad para generar una separabilidad entre clases, la cual se intentaría lograr clasificar con las RNAs.

Para comparar el rendimiento de la RNA tradicional y la RNA propuesta Se realizaron un total de cinco experimentos, cada uno con diferentes configuraciones y *Datasets*, cada experimento se ejecutó 35 veces con el fin de obtener estadísticos que se pudieran analizar y reportar.

En la en la mayoría de los experimentos se puede observar que el rendimiento de la Red Neuronal Artificial (RNA) propuesta obtuvo altos porcentajes de precisión en

la clasificación de los distintos *Datasets*, en dos de los cinco experimentos ésta logró mejorar la precisión en comparación con la RNA tradicional, sin embargo, esta última también obtuvo mejores porcentajes de clasificación en dos experimentos teniendo así un empate general, ya que en el primer experimento las dos RNAs lograron un 100% de clasificación.

Recordando que el objetivo del enfoque propuesto en este trabajo es el de disminuir la arquitectura de la RNA utilizando una función ψ que permita alterar los datos de entrada y con esto reducir el coste computacional que requeriría una arquitectura más compleja. Este objetivo se logró, por qué la RNA propuesta logró igualar los resultados obtenidos por la RNA tradicional reduciendo el número de capas y/o neuronas, Con respecto al tiempo de ejecución se encontró qué, para que se note una diferencia, la arquitectura de la RNA debe ser bastante compleja, solo en los últimos dos experimentos realizados la RNA propuesta redujo el tiempo de ejecución, esto porque en la configuración de las RNA una diferencia amplia en el número de neuronas y capas.

Con el análisis de estos resultados podemos concluir de manera general, que se logró proponer una RNA cuya unidad básica, el perceptrón, implementa una función generalizada que permite resolver problemas de separabilidad lineal o no lineal y para reducir el coste computacional es necesario utilizar problemas de clasificación más complejos, donde la arquitectura de la red requiera de un gran número de capas y neuronas para que estas se sustituyan por la función generalizada reflejando así la capacidad de optimización de este nuevo enfoque.

Como trabajo futuro, se estudiará la eficacia y efectividad de la propuesta sobre problemas de mayor complejidad, ya que en este tipo de problemas las arquitecturas neuronales, propuestas bajo un enfoque tradicional, tienden a crecer desmesuradamente.

BIBLIOGRAFÍA

Larrañaga, P., Inza, I., & Moujahid, A. (2015). Departamento de Ciencias de la Computación e Inteligencia Artificial Universidad del País Vasco–Euskal Herriko Unibertsitatea.

Chaudhuri, B., & Bhattacharya, U. (2000). Efficient training and improved performance of multilayer perceptron in pattern classification. *Neurocomputing, 34*(1-4), 11-27.

Abid, S., Fnaiech, F., & Najim, M. (2001). A fast feedforward training algorithm using a modified form of the standard backpropagation algorithm . *IEEE Transactions on Neural Networks, 12*(2), 424-430.

Damián, J. (2001). Redes Neuronales: Conceptos Básicos y Aplicaciones. *Grupo de Investigación Aplicada a la Ingeniería Química (GIAIQ)*.

Gholami, A., Bonakdari, H., Hossein Zaji, A., Ajeel Fenjan, S., & Akbar Akhtari, A. (2016). Design of modified structure multi-layer perceptron networks based on decision trees for the prediction of flow parameters in 90° open-channel bends. *Engineering Applications of Computational Fluid Mechanics, 10*(1), 193-208.

Hecht-Nielsen, R. (March de 1988). Neurocomputing: picking the human brain. *IEEE Spectrum, 25*, 36-41.

Hilera, J., & Martínes, V. (1995). *REDES NEURONALES ARTIFICIALES FUNDAMENTOS, MODELOS Y APLICACIONES.* Madrid: RA-MA Editorial.

Kohonen, T. (1988). An introduction to neural computing. *Neural Networks, 1*, 3-16.

McCulloch, W., & Walter, P. (1943). A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics, 5*, 115-133.

Maass, W. (1997). Networks of spiking neurons: The third generation of neural network models. *Institute for Theoretical Computer Science, Technische Universität Graz, Neural Networks, 10*(9), 1659-1671.

Mashor, M. Y. (2010). Hybrid multilayered perceptron networks. *International Journal of Systems Science, 31*(6), 771-785.

Parlos, A. G., Chong, K. T., & Atiya, A. F. (1994). Application of the recurrent multilayer perceptron in modeling complex process dynamics. *IEEE Transactions on Neural Networks, 5*(2), 255-266.

Rosenblatt, F. (1962). *Principles of Neurodynamics: Perceptrons and the Theory of Brain Mechanisms.* Washington, D. C.: Spartan Books.

Rumelhart, D. E., & MacClelland, J. L. (1988). *Parallel Distributed Processing* (Vol. 1). Foundations, MIT Press.

Scalero, R., & Tepedelenlioglu, N. (1992). A fast new algorithm for training feedforward neural networks. *IEEE Transactions on Signal Processing, 40*(1), 202-210.

Subhi Al-Batah, M., Ashidi Mat Isa, N., Zuhairi Zamli, K., & Azizi Azizli, K. (2010). Modified Recursive Least Squares algorithm to train the Hybrid Multilayered Perceptron (HMLP) network. *Applied Soft Computing, 10*(1), 236-244.

Suyken, J., & Vandewalle, J. (1999). Training multilayer perceptron classifiers based on a modified support vector method. *IEEE Transactions on Neural Networks, 10*(4), 907-911.

Thomas, P., & Suhner, M.-C. (2014). A New Multilayer Perceptron Pruning Algorithm for Classification and Regression Applications. *Neural Processing Letters, 42*, 437-458.

Waleed, Y., Tharwat, A., F. Hassanin, M., Gaber, T., Hassanien, A. E., & Kim, T.-H. (2015). A New Multi-layer Perceptrons Trainer Based on Ant Lion Optimization Algorithm. *2015 Fourth International Conference on Information Science and Industrial Applications (ISI)*, 40-45.

Werboz, P. J. (1974). Beyond Regression: New Tools for Prediction and Analysis in Behavioral Sciences. *Tesis Doctoral. Universidad de Harvard*.

Widrow, B., & Hoff, M. E. (1988). Neural nets for adaptive filtering and adaptive patterns recognition. *IEEE Computer*, 25-39.

ANEXO 1 – CÓDIGO DE PYTHON

Red Neuronal Artificial (RNA) tradicional

```python
import numpy as np

# Funciones de activación y sus derivadas
sigmoid = lambda x: 1 / (1 + np.e**-x)
der_sigmoid = lambda x: x * (1 - x)

tanh = lambda x: (np.e**x - np.e**-x) / (np.e**x + np.e**-x)
der_tanh = lambda x: 1 - tanh(x)**2

relu = lambda x: x if x >= 0 else 0
der_relu = lambda x: 1 if x >= 0 else 0

class NeuralNetwork():

    def __init__(self, layers, f_activation = sigmoid,
                 der_f_activation = der_sigmoid):
        """
        Clase NeuralNetwork
        Args:
            layers: Capa de entrada - N capas ocultas - Capa de salida (list)
            f_act: Función de activación (function)
            der_f_act: Derivada de la función de activación (function)
        """
        self.n_lay = len(layers) - 1
        self.f_act = f_activation
        self.der_f_act = der_f_activation
        self.bias = []
        self.weights = []

        for i in range(self.n_lay):
            self.bias.append(np.random.rand(1, layers[i + 1]) * 2 - 1)
            self.weights.append(np.random.rand(layers[i],
            layers[i + 1]) * 2 - 1)

    def prediction(self, X):
        """
        Realiza una predicción de la clasificación de los datos ingresados
        Args:
            X: Datos de entrada
        Returns:
            outputs = Salidas de cada una de las capas de la red (list)
```

```
        """
        outputs = [X]
        for i in range(self.n_lay):
            z = outputs[i] @ self.weights[i] + self.bias[i]
            a = self.f_act(z)
            outputs.append(a)
        return outputs

    def training(self, X, Y, lr = 0.05, epochs = 1000):
        """
        Realiza el entrenamiento de la red
        Args:
            X: Datos de entrada
            Y: Valores deseados
            lr: Larning rate
            epochs: Número de épocas de entrenamiento
        """
        for _ in range(epochs):
            outputs = self.prediction(X)
            a = outputs[-1]
            e = Y - a
            d = self.der_f_act(a) * e

            deltas = [d]
            for l in reversed(range(self.n_lay - 1)):
                a = outputs[l + 1]
                e = deltas[0] @ self.weights[l + 1].T
                d = self.der_f_act(a) * e
                deltas.insert(0, d)

            for l in range(self.n_lay):
                DW = (outputs[l].T @ deltas[l]) * lr
                self.weights[l] += DW
                self.bias[l] += np.mean(deltas[l], axis=0, keepdims=True) * lr
```

Red Neuronal Artificial (RNA) propuesta

```
import numpy as np
import json

class NewNeuralNetwork():

    def __init__(self, file):
        """
        Clase NewNeuralNetwork - La arquitectura se pasa por un JSON
        Args:
            file: Arquitectura de la red (path file.json)
```

```
        """
        with open(file) as file:
            architecture = json.load(file)
        self.n_lay = len(architecture) - 1
        self.weights = []
        self.bias = []
        self.f_act = []
        self.der_f_act = []
        self.f_psi = []
        for l in range(self.n_lay):
            self.weights.append(np.random.rand(architecture[str(l)]
            ['number_of_attributes'], architecture[str(l + 1)]
            ['number_of_attributes']) * 2 -1)
            self.bias.append(np.random.rand(1, architecture[str(l + 1)]
            ['number_of_attributes']) * 2 -1)
            self.f_act.append( lambda x: eval(architecture[str(l + 1)]
            ['activation_function']))
            self.der_f_act.append(lambda x: eval(architecture[str(l + 1)]
            ['derivative_activation_function']))
            self.f_psi.append(lambda x: eval(architecture[str(l + 1)]
            ['psi_function']))

    def prediction(self, X):
        """
        Método para realizar la predicción de las N salidas en cada una
        de las capas de la red
        Args:
            X: Datos de entrada (matriz)
        Returns:
            outputs: Salidas de cada una de las capas
                     cE, cO1, cO2 ... cON, cS (list)
        """
        outputs = [X]
        for l in range(self.n_lay):
            z = self.f_psi[l](outputs[l]) @ self.weights[l] + self.bias[l]
            a = self.f_act[l](z)
            outputs.append(a)
        return outputs

    def training(self, X, Y, lr = 0.05, epochs = 1000):
        """
        Método para realizar el entrenamiento de la red
        Args:
            X: Datos de entrada (matriz)
            Y: Salidas esperadas (matriz)
            lr: Learning rate (float)
            epochs: Número de épocas de entrenamiento (int)
```

```
        """
        for _ in range(epochs):
            outputs = self.prediction(X)
            a = outputs[-1]
            e = Y - a
            d = self.der_f_act[-1](a) * e

            deltas = [d]
            for l in reversed(range(self.n_lay - 1)):
                a = outputs[l + 1]
                e = deltas[0] @ self.weights[l + 1].T
                d = self.der_f_act[l](a) * e
                deltas.insert(0, d)

            for l in range(self.n_lay):
                DW = self.f_psi[l](outputs[l].T) @ deltas[l] * lr
                self.weights[l] += DW
                self.bias[l] += np.mean(deltas[l], axis=0, keepdims=True) * lr

def generate_accuracy(num_inputs, m_classes, yp):
    """
    Método para obtener el accuracy de los Datasets sintéticos
    Args:
        num_inputs: Número de entradas
        m_clasess: Matriz de clases
        yp: Salida de la RNA
    Outputs:
        m_accuracy: Matriz de confusión
        accuracy: El accuracy de la RNA
    """
    m_accuracy = np.zeros((2, 2))
    idx = 0
    for i in range(len(m_classes)):
        for j in range(int(num_inputs/2)):
            for k in range(len(m_classes[i])):
                predict = 0.0
                if yp[idx] > 0.5:
                    predict = 1.0
                if predict == m_classes[i][k]:
                    m_accuracy[i][k] += 1
            idx += 1
    acc = 0
    for i in range(len(m_accuracy)):
        for j in range(len(m_accuracy[i])):
            if i == j:
                acc += m_accuracy[i][j]
    accuracy = (acc * 100) / idx
```

```
    print('Matriz de confusión \n', m_accuracy)
    print(f'El accuracy es de un {accuracy}%')
```

Experimentación

```
from NNV1 import *
from NNV4 import *
from PLOT import *
from sklearn.datasets import make_circles
import time

# Datasets sintéticos
df = pd.read_csv('/Users/gibbystack/PythonProjects/Titulacion/3. Red
Neuronal/Datasets/cos.csv').values
X = df[:, :-1]
Y = df[:, -1:]


# Dataset Sklearn
# X, Y = make_circles(n_samples=500, factor=0.5, noise=0.05)
# Y = Y[:, np.newaxis]

# Inicio de tiempo de ejecución de las RNAs
start_nn = time.time()
# Configuración de la RNA tradicional
nn = NeuralNetwork(layers = [len(X[0]), 6, 2, 1])
nn.training(X, Y, lr = 0.05, epochs = 10000)
yp1 = nn.prediction(X)
# Término de tiempo de ejecución de la RNA tradicional
end_nn = time.time()

# Configuración de la RNA propuesta
nn2 = NewNeuralNetwork('/Users/gibbystack/PythonProjects/Titulacion/3. Red
Neuronal/Architectures/ARCV1.json')
nn2.training(X, Y, lr = 0.05, epochs = 10000)
yp2 = nn2.prediction(X)
# Término de tiempo de ejecución de la RNA propuesta
end_nn2 = time.time()

# Matriz de clases para los Datasets sintéticos
m_classes = np.array((
    [0.0, 1.0],
    [0.0, 1.0]
))

# Impresión de resultados
print('=======RNA TRADICIONAL=======')
```

```
generate_accuracy(len(yp1[-1]), m_classes, yp1[-1])
print(f'Tiempo de ejecución: {end_nn - start_nn}')

print('=======RNA PROPUESTA=======')
generate_accuracy(len(yp2[-1]), m_classes, yp2[-1])
print(f'Tiempo de ejecución: {end_nn2 - end_nn}')

# Impresión de gráficos de clasificación
print_plot(X, Y, yp1[-1], yp2[-1])
```