PONTIFICIA UNIVERSIDAD CATÓLICA DE VALPARAÍSO

Fernando Esteban Molina Ovando

Desarrollo de un perceptrón Multicapa en Raspberry Pi

Informe Proyecto de Título de Ingeniero Electrónico

Escuela de Ingeniería Eléctrica
Facultad de Ingeniería

Valparaíso, 04 de marzo de 2019

Desarrollo de un Perceptrón Multicapa en Raspberry Pi

Fernando Esteban Molina Ovando

Informe Final para optar al título de Ingeniero Electrónico,
aprobada por la comisión de la
Escuela de Ingeniería Eléctrica de la
Facultad de Ingeniería de la
Pontificia Universidad Católica de Valparaíso
conformada por

Sr. David Velásco López
Profesor Guía

Sr. Francisco Alonso Villalobos
Segundo Revisor

Sr. Sebastián Fingerhuth Massmann
Secretario Académico

Valparaíso, 04 de marzo de 2019

A mi sobrino Tomás Esteban.

Agradecimientos

A mi madre Ester y padre Isaac por su amor, comprensión y apoyo en todas las etapas vividas. A mi hermano Isaac por ser mi compañero y amigo en este camino. A mi hermana Denisse por su cariño y por traer al mundo esa luz que nos iluminó a todos de nuevo. A mis tías Eliana y Amanda que, gracias a su apoyo incondicional, hicieron posible que cualquier dificultad fuese menor. A mi tío Rubén por su amistad y cariño que me brinda siempre. A mis amigas Constanza y Vanessa que siempre me brindaron apoyo y alegría. A mis profesores de tesis por guiarme en este nuevo desafío. Y a todos mis compañeros de universidad, que de una u otra forma brindaron motivación para terminar mis estudios.

Valparaíso, 4 de marzo de 2019

Fernando Molina Ovando

Resumen

La presente investigación, se enmarca dentro del trabajo que busca desarrollar herramientas de inteligencia artificial en un computador de placa reducida Raspberry Pi, a través del estudio de instrumentos que sirven para el desarrollo de algoritmos de inteligencia artificial y redes neuronales que son necesarios para la implementación del perceptrón multicapa (PMC), como estrategia para la clasificación de patrones numéricos.

La metodología utilizada para el desarrollo de la red neuronal, consiste en un mecanismo de aprendizaje supervisado, que mediante su algoritmo por corrección de errores adapte el sistema para minimizar el error obtenido, y así acercarse al valor deseado, generando un aprendizaje a través de la actualización de los pesos de las conexiones de la red.

Este trabajo busca establecer una guía que entregue parámetros que faciliten la elaboración de una segunda etapa de implementación física; consistente en desarrollar un sistema de reconocimiento en tiempo real, mediante el uso de cámaras y sensores que permitan la realización de una nueva etapa de desarrollo

Palabras claves: Redes neuronales artificiales, perceptrón, inteligencia artificial, Raspberry Pi, sistema de reconocimiento de patrones.

Abstract

This research is part of the work that seeks to develop artificial intelligence tools in a computer with a small, single-board Raspberry Pi. This is done through the study of instruments used in the development of artificial intelligence algorithms and neural networks that are necessary for the implementation of the multilayer perceptron (MLP), as a strategy for numerical patterns classification.

The methodology used for the development of the neural network consists of a supervised learning mechanism, which by means of its error correction algorithm adapts the system to minimize the error obtained, and thus approach the desired value. This generates learning through the update of the weights of the network connections.

This work seeks to establish a guide that provides parameters to facilitate the establishment of a second stage: physical implementation. This consists of developing a real time recognition system by using cameras and sensors that allow the realization of the new stage of development

Key words: Artificial neuronal network, perceptron, artificial intelligence, raspberry pi, pattern recognition system.

Índice general

Introducción

La inteligencia artificial viene desarrollándose progresivamente desde los años 50's con los escritos de Alan Turing sobre maquinarias pensantes [1]. En la actualidad se puede considerar que todo sistema tecnológico que interactúa con el ser humano está basado en sistemas que incluyen la inteligencia artificial. Los campos que aborda la IA son tan diversos y amplios que casi todo lo que se usa cotidianamente basa su funcionamiento en ésta. Algunos de los campos que aborda la inteligencia Artificial son; el reconocimiento de patrones, reconocimiento de voz, los negocios y toma de decisiones y el aprendizaje profundo o "Deep Learning", entre otros. Entre las principales aplicaciones del reconocimiento de patrones se encuentra; reconocimiento de caracteres, visión por máquina y reconocimiento de voz.

Un ejemplo actual de este tipo de inteligencia artificial basado en el reconocimiento de voz, es la aplicación Siri de Apple [2]. Éste sistema se basa en transcribir el lenguaje humano y redirigirlo hacia ordenes tales como; realizar una llamada, escribir un mensaje de texto, leer documentos, programar alarmas, entre tantas otras tareas cotidianas que pueden facilitar la vida del usuario. Por otro lado, la inteligencia artificial también puede ser utilizada en el campo de los negocios, específicamente en la toma de decisiones, posibilitando negocios más rentables gracias a una variedad de aplicaciones corporativas que permiten la toma de decisiones de manera automática. Un ejemplo de ello, es la empresa Advanced Systems Concepts Inc. [3] dedicada a mejorar el rendimiento del área de negocios y de las operaciones TI a través del desarrollo de software. Las plataformas de aprendizaje profundo o "Deep Learning", se utilizan principalmente para clasificar datos y para el reconocimiento de patrones, siendo éste último quien se dedica a extraer las características esenciales de objetos para poder clasificarlos en categorías o clases. La empresa "Deep Instict" [4] utiliza el aprendizaje profundo en la ciberseguridad, empleando inteligencia artificial para la detección de malware y para prevenir cualquier tipo de amenaza virtual. Otro campo importante que desarrolla la inteligencia artificial en la actualidad, es el reconocimiento de imágenes basado en identificar y detectar objetos o características relevantes en una imagen o video, éste tipo de reconocimiento también es utilizado para verificar usuarios mediante el rostro y detección de placas de autos entre otras aplicaciones. La empresa China "Sensetime" [5] es una de las líderes en el campo del reconocimiento de imágenes, se ha convertido en el mayor proveedor de algoritmos de inteligencia artificial con una gama alta de servicios tales como: reconocimiento facial, reconocimiento de imágenes, reconocimiento de objetos, análisis de videos, tele-conducción y conducción autónoma, entre otras.

El machine learning, también conocido por sus siglas “ML”, es una rama de la inteligencia artificial cuyo principal objetivo es desarrollar técnicas que permitan a los computadores aprender. En la actualidad, el Machine Learning, va ganando mayor fuerza debido a que proporciona: algoritmos, herramientas de desarrollo, big data y capacitación, que se utilizan para clasificación y predicción. Compañías como Google, Amazon, Fractal Analytics venden plataformas machine learning. El mundo de la inteligencia artificial sigue creciendo cada año con más potencia, y más empresas se están enfocando en satisfacer las necesidades de los usuarios mediante las técnicas y campos que ésta ofrece. Las redes neuronales están ubicadas en la base de la inteligencia artificial, siendo un elemento fundamental para su funcionamiento. Las redes neuronales artificiales son modelos inspirados y basados en las redes neuronales biológicas, por ende, se caracterizan por su capacidad de aprender desde la experiencia, vale decir, modifican su estructura interna (pesos) como respuesta a su entorno. La unidad básica de una red neuronal artificial consiste en un procesador llamado neurona, este procesador posee la capacidad de calcular la suma ponderada de sus entradas, luego aplica una función de activación que sirve para obtener una respuesta que será transmitida a las neuronas de todas las conexiones de la red. La capacidad de procesamiento de estas redes es sustentada en la conectividad que tienen las neuronas que la componen. Las redes neuronales artificiales pueden abstraer las características principales de los patrones de entrada, es decir, cuando estas redes son entrenadas con un conjunto de datos en dicha fase, podrá reconocer nuevos patrones de entrada que se encuentren incompletos o distorsionados. Gracias a esta característica, el sistema buscará similitudes entre los datos aprendidos y los nuevos datos. Las redes neuronales artificiales son utilizadas en diversas aplicaciones tales como: reconocimiento de caracteres, compresión de imágenes, procesamiento de alimentos, utilización militar, diagnóstico de máquinas, análisis de firmas, etc. Entre las redes neuronales conocidas se encuentran: el perceptrón simple y multicapa, ambos ampliamente utilizados en el mundo de la inteligencia artificial gracias a su fácil implementación y cómodos algoritmos que hacen que dichas redes sean de las más utilizadas en esta área.

Raspberry Pi ha sido utilizada en estos últimos años como una herramienta para desarrollar todo tipo de proyectos, entre los que se encuentran; controladores midi, lecturas de huellas para acceso, alarmas y control de riego, entre otros. Debido a su versatilidad y sus características similares al de un computador, plantea las bases para una investigación sobre inteligencia artificial y su aplicación en redes neuronales artificiales.

Este trabajo busca realizar un estudio sobre las redes neuronales artificiales, específicamente; el perceptrón simple y el perceptrón multicapa, poniendo mayor énfasis en el perceptrón multicapa. El perceptrón simple se caracteriza por poseer una estructura de dos niveles o capas, en el cual, todo el procesamiento se desarrolla en la capa de salida, siendo ésta una de sus principales desventajas junto con poder hacer clasificaciones de patrones que solamente sean separables linealmente, como es el caso de una compuerta lógica OR o AND. El perceptrón multicapa fue desarrollado para resolver el problema que presenta el perceptrón simple, éste problema es solucionado aumentando el número de capas, agregando las denominadas capas ocultas, que reciben este nombre porque no interactúan directamente con el entorno, y se encuentran entre la capa de entrada y la capa de salida.

1 Antecedentes generales y propuesta

1.1 Planteamiento del problema

Las redes neuronales artificiales son ampliamente utilizadas para el reconocimiento de patrones, reconocimiento de imágenes y clasificación de datos. Una de las redes más utilizadas es el perceptrón multicapa gracias a su algoritmo de fácil implementación. El uso de una red neuronal artificial requiere de un estudio previo, que sea capaz de dar cuenta cómo implementar un sistema neuronal y un método para el uso de algoritmos y ecuaciones que posibiliten el desarrollo de una red artificial. Para ello, se utiliza la plataforma Raspberry Pi, que otorga una ventaja de procesamiento por ser un computador de placa reducida, y por su portabilidad a la hora de ser implementada.

El proyecto busca desarrollar una guía que sirva como base para poder elaborar modelos neuronales de tipo Perceptrón Multicapa e implementarlo en un Raspberry Pi, dejando los conceptos de redes neuronales artificiales para una segunda etapa que incluyan sensores que permitan el funcionamiento de esta red en tiempo real o para el mejoramiento del sistema de reconocimiento desarrollado.

1.2 Formas de abordar el problema

Los Modelos Neuronales de tipo Perceptrón Multicapa se han utilizado comúnmente como clasificadores, debido a que son modelos que se presentan sólidos, eficientes y eficaces para la clasificación de problemas complejos, alcanzando óptimos resultados en comparación con modelos clásicos antes desarrollados. La Raspberry Pi aparece en los últimos años como una alternativa mucho más sencilla para desarrollar sistemas de reconocimiento de patrones, por sobre otros ya existentes.

La Raspberry Pi surge como una estrategia para fomentar en los niños el aprendizaje de la electrónica y la programación, para así ser capaces de desarrollar el uso y comprensión de sistemas computacionales, teniendo como objetivo principal el dar acceso a dispositivos portables y de bajo costo. Con el pasar de los años, sus posibilidades y prestaciones han logrado convertirla en base de todo tipo de proyectos, que han logrado maximizar sus usos en el campo de la electrónica e informática.

1.3 Estado del Arte

Son diversos los trabajos que se han realizado entorno a las redes neuronales artificiales, utilizando el perceptrón multicapa como solución a problemas, generalmente, siendo implementado como un sistema de reconocimiento de patrones, o clasificador de datos. Los trabajos más relevantes y que sirven de guía para este trabajo de investigación son los siguientes:

1.3.1 Neuronal network system of traffic signs recognition [6]

Este artículo describe un enfoque para la detección y el reconocimiento de señales de tráfico en tiempo real con cambios de iluminación y distancia. Este trabajo se llevó a cabo con la utilización de un Raspberry Pi 2 y una cámara web Hama AC-150. Para la detección de las señales de tráfico se utiliza un filtro de colores con operadores morfológicos y algoritmo de Canny, este algoritmo es utilizado para la detección de bordes en imágenes. La metodología utilizada por el autor es la siguiente: un procesamiento de imágenes que contempla la detección de bordes, dilatación y segmentación de la imagen; la extracción de características; clasificación; y el reconocimiento de los patrones. Se entrena con cinco señales de tráfico distinta para probar el algoritmo del perceptrón multicapa, éste consta con 43 neuronas en la capa de entrada y 5 neuronas en la capa de salida. Finalmente, el sistema es capaz de reconocer señales de tráfico de 20 cm de diámetro en tiempo real a una distancia que varía entre 2-2.5 metros.

1.3.2 Fundamentos para la implementación de red neuronal perceptrón multicapa mediante software [7]

Este trabajo consiste en la implementación de un perceptrón multicapa mediante el software MATLAB. Se utiliza una red multicapa para el reconocimiento de patrones numéricos, mediante la herramienta “nntool” que proporciona el software antes mencionado. Ésta interfaz gráfica permite la creación de redes neuronales artificiales donde se pueden considerar: el número de neuronas por capas; el número de capas que se desean ocupar; la tasa de aprendizaje; y el tipo de aprendizaje que se desea llevar a cabo. La metodología utilizada es la siguiente: selección de los patrones de entrada; entrenamiento de la red; y comprobación de la red mediante ejemplos no antes visto en la fase de entrenamiento. Se utiliza en la capa de entrada 1024 neuronas, en la primera capa oculta 100 neuronas, en la segunda capa oculta 50 neuronas, y en la capa de salida se utilizan 10 neuronas. Finalmente, el autor logra crear una red perceptrón multicapa mediante el software Matlab, reconociendo todos datos ingresados de manera correcta.

1.3.3 Reconocimiento de patrones mediante redes neuronales artificiales [8]

Este articulo muestra una solución para reconocer patrones numéricos mediante la implementación de un perceptrón multicapa en Matlab. Éste software tiene una herramienta denominada *Neurotrain-patter* que sirve para el desarrollo de redes neuronales artificiales. La metodología utilizada por los autores es la siguiente: dar una estructura a los datos de entrada, es decir, representar los patrones de entrada mediante una matriz; transformar la matriz en un vector lineal; y dar una estructura a los datos de salida. Finalmente, mediante las herramientas

que brinda el software Maltlab, los autores logran crean una red perceptrón multicapa eficiente que logra reconocer los números 3,4 y 5.

1.4 Objetivos

A continuación, se detallan los objetivos para el desarrollo del proyecto del perceptrón multicapa, mencionando los objetivos generales y específicos.

1.4.1 Objetivos Generales

- Investigar acerca de herramientas de inteligencia artificial en sistemas de Raspberry Pi.

1.4.2 Objetivos específicos

- Estudiar herramientas para el desarrollo de algoritmos de inteligencia artificial y redes neuronales.
- Elaborar diseño de implementación de redes neuronales de tipo Perceptrón simple y multicapa.
- Implementar un perceptrón multicapa parametrizable utilizando Raspberry Pi.
- Desarrollar un reconocedor de patrones numéricos.

2 Redes Neuronales Artificiales

Las redes neuronales artificiales intentan representar el modelo biológico del cerebro humano mediante modelos matemáticos. Una red neuronal artificial está compuesta por una gran cantidad de elementos sencillos llamados neuronas; estas neuronas se encuentran organizadas en niveles o capas y forman una unidad de procesamiento potente al estar interconectadas entre capas.

2.1 Neurona Biológica

El elemento estructural más esencial en el sistema de comunicación neuronal es la célula nerviosa o neurona. Éstas son las encargadas de recibir la información que proviene desde otras neuronas o receptores en forma de impulsos y transmitirla a través de sus terminales hacia otras neuronas. Hay células nerviosas que están en relación con receptores, mediante los cuales llega información proveniente desde el exterior o el interior del organismo hasta las redes neuronales.

El sistema de comunicación neuronal está compuesto por tres partes:

1. **Receptores:** Se encuentran en las células sensoriales y son las encargadas de recibir la información en forma de estímulos que proviene desde el ambiente o desde el interior del organismo.
2. **El sistema nervioso:** Es el encargado de recibir la información, almacenar y enviarla a los órganos efectores y a otras zonas del sistema nervioso.
3. **Órganos efectores:** Son los encargados de ejecutar las respuestas del sistema nervioso

En la Figura 2-1 se puede ver un diagrama del sistema de comunicación neuronal

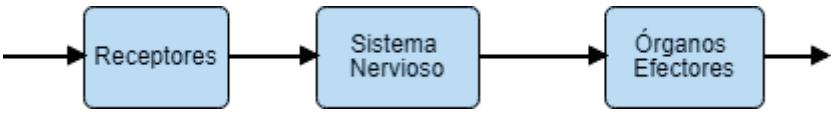

Figura 2-1 Sistema de Comunicación neuronal

El proceso de comunicación de las células nerviosas consiste en impulsos eléctricos entre neuronas a través de las dendritas, éstas se conectan a las salidas de otras células nerviosas para producir la sinapsis (mecanismo de comunicación direccional entre neuronas). La sinapsis altera

la efectividad de la señal transmitida debido a un parámetro, el peso. El aprendizaje es el resultado de la modificación del peso y junto con el procesamiento de la información se genera el mecanismo básico de la memoria [9]. Las dendritas son las zonas receptivas de una neurona, siendo el axón una línea de transmisión y los botones sinápticos que se encuentran en los terminales del axón son los encargados de comunicar los impulsos a otras neuronas. En el soma se suman las entradas de todas las dendritas, si estas entradas sobrepasan un cierto umbral, entonces se transmitirá un pulso a lo largo del axón, en caso contrario, no se transmitirá. El impulso que llega a una sinapsis y el que sale de ella no son iguales en general, debido a que la sinapsis modifica el pulso, ya sea reforzándolo o debilitándolo. En la Figura 2-2 se puede ver una neurona biológica con las principales partes que la componen.

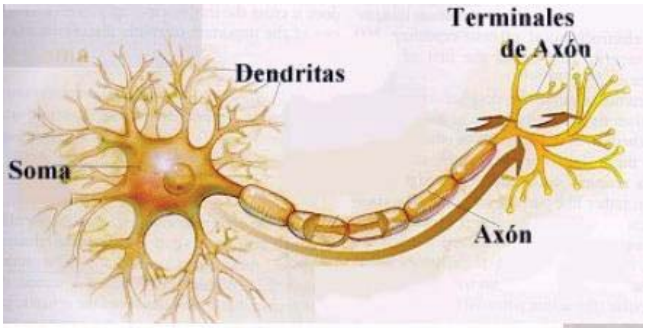

Figura 2-2: Neurona Biológica (Fuente: http://ponce.inter.edu)

2.2 Modelo Neuronal

La neurona artificial trata de simular el comportamiento de una neurona biológica basándose en el sistema de comunicación neuronal. Las entradas o receptores de información del modelo neuronal están representadas por x_1, x_2 y se pueden extender hasta x_n dependiendo de la aplicación que se requiera, éstas entradas son las encargadas de recibir la información proveniente del exterior y propagarlas hacia las neuronas que se encuentran en la siguiente capa o nivel. Los pesos sinápticos están representados por w_1, w_2 hasta w_n, y estos se modifican mediante un algoritmo para que la red se adapte y aprenda. En el bloque sumador se obtienen las sumas ponderadas de cada una de las entradas que luego son evaluadas en una función de activación, ésta se encarga de transformar la entrada total en la respuesta de la neurona, limitando el rango de respuesta de dicha neurona. En la Figura 2-3 se puede apreciar el modelo de una neurona artificial.

La salida del modelo neuronal está determinada por la ecuación (2-1).

$$y = f\left(\sum_{i=1}^{n} x_i w_i + w_o\right) \tag{2-1}$$

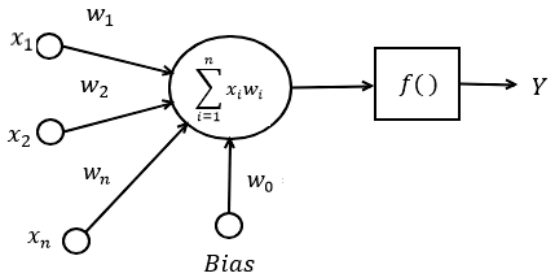

Figura 2-3: Modelo neuronal artificial

En general:

x_1, x_2, x_n: Entradas de la neurona.

w_1, w_2, w_n: Pesos de la red.

w_0: Peso asociado al Bias.

$Bias$: Neurona adicional.

$\sum_{n=1}^{n} w_i w_i$: Bloque sumador.

$f()$: Función de activación.

Y: Salida de la neurona.

2.3 Clasificación redes neuronales por su arquitectura

Las redes neuronales se encuentran estructuradas en capas o niveles, donde existen cierta cantidad de neuronas por cada capa o nivel. El número de neuronas distribuido a lo largo de la red va a depender de la aplicación que se quiera llevar a cabo. Una característica de las redes neuronales es que las neuronas por capa se encuentran totalmente conectadas con las neuronas de la capa siguiente compartiendo información.

2.3.1 Redes Monocapa

Las redes monocapa se caracterizan por estar compuestas por una capa de entrada y una capa de salida. Las neuronas ubicadas en la capa de entrada se encuentran conectadas a través de sus pesos con las neuronas que se encuentran en la capa de salida, compartiendo información mediante esta conexión. Las redes monocapa son rápidas en términos de procesamiento debido a que no demandan demasiados cálculos y actualizaciones, pero no deben ser usadas en casos en los que se presenten problemas de no linealidades significativas en el tiempo. En la Figura 2-4 se muestra una red monocapa la cual está compuesta por tres neuronas en la capa de entrada, una de ellas representa el Bias el cuál generalmente se utiliza como entrada de valor 1 y éste no cambia con el ingreso de patrones de entrada, en la capa de salida se encuentra una neurona que es la encargada de realizar los cálculos y entregar la respuesta de la red.

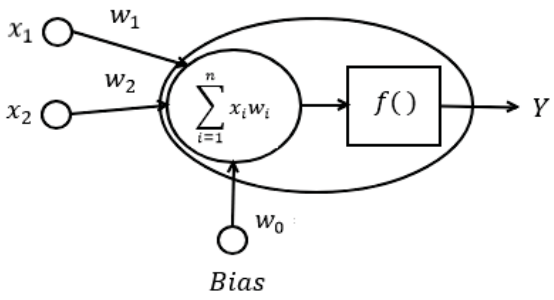

Figura 2-4: Red Monocapa

2.3.2 Redes Multicapa

Las redes multicapa se pueden definir como varias redes monocapa conectadas en cascada. A diferencia de las redes monocapa, las multicapas poseen un nivel intermedio entre la salida y la entrada denominado capa oculta, recibe este nombre debido a que este nivel no tiene una relación directa con el entorno. Las neuronas ubicadas en la capa de entrada se encargan de recibir la información que proviene del exterior y propagarla hacia la capa oculta, en este nivel (capa oculta) se realiza el procesamiento de la información que luego es llevado hacia la capa de salida para obtener la respuesta de la red ante el estímulo que recibió de los patrones o vectores de entrada. En la Figura 2-5 se muestra la arquitectura de una red multicapa, compuesta por tres neuronas en la capa de entrada, tres neuronas en la capa oculta y una neurona en la capa de salida.

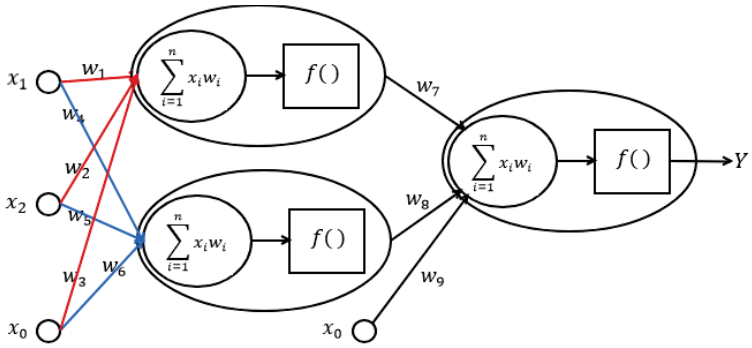

Figura 2-5: Red Multicapa

2.4 Clasificación de redes neuronales por su aprendizaje

El aprendizaje es el mecanismo mediante el cual una red modifica sus pesos en respuesta a los patrones de entrada, es el proceso más importante debido a que en esta etapa se define el buen funcionamiento y capacidad para resolver problemas. Las redes neuronales artificiales son sistemas que aprenden a base de ejemplos y estos deben ser significativos, esto quiere decir, que debe existir un número suficiente de ejemplos debido a que, si existe un número reducido de patrones de entrenamiento la red no será capaz de adaptar sus pesos de forma eficaz. El conjunto de entrenamiento también debe ser representativo, es decir, los ejemplos que componen el conjunto de entrenamiento deben ser diversos porque si existen muchos ejemplos de un solo tipo

la red se puede especializar en dicho subconjunto de datos y no serviría para aplicación general. El aprendizaje de una red consiste en determinar los valores precisos de los pesos para cada una de las conexiones, de modo que las capacite para resolver problemas eficientemente. El aprendizaje comienza cuando se introducen los ejemplos del conjunto de entrenamiento y la red modifica los pesos de las conexiones de modo que la salida de la red sea la más parecida a la salida deseada. Una vez que concluye la fase de entrenamiento, es necesario comprobar si la red está capacitada para clasificar patrones que no haya visto en el proceso de entrenamiento, si es que ésta no cumple un criterio de convergencia, se debe repetir el proceso y todos los ejemplos deben volver a ser introducidos. En la Figura 2-6 se muestra un esquema de los mecanismos de aprendizaje en redes neuronales artificiales.

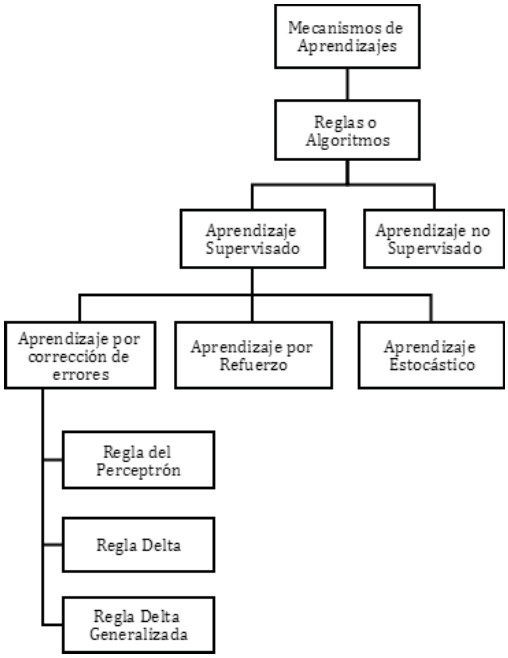

Figura 2-6: Mecanismos de aprendizajes

2.4.1 Aprendizaje Supervisado

La característica principal de este método es que el proceso de aprendizaje es controlado mediante un agente externo, éste es quién determina cuál debiese ser la salida de la red para los patrones de entrada. Dentro del aprendizaje supervisado se pueden encontrar tres tipos de modelos: aprendizaje por corrección de errores, aprendizaje por refuerzo y aprendizaje estocástico.

1. **Aprendizaje por corrección de errores**

El aprendizaje por corrección de errores y consiste en presentar al sistema un conjunto de patrones de entrenamiento de los cuales se debe conocer su salida. Con este tipo de aprendizaje se intenta minimizar el error producido por la salida deseada y de la red. Este método se emplea iniciando aleatoriamente los pesos de las conexiones de toda la red, luego se presenta el conjunto de patrones de entrenamiento, se obtienen las salidas para dichos patrones, se comparan las

salidas deseadas por cada patrón con las obtenidas por la red, luego se debe verificar si el error es cero, en el caso que no sea cero se deben modificar los pesos de las conexiones mediante un criterio, en caso contrario, se debe pasar al siguiente patrón de entrenamiento. En la figura 2-7, se puede observar un esquema de aprendizaje por corrección de errores.

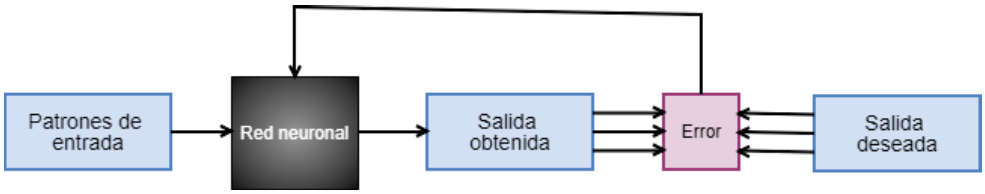

Figura 2-7: Aprendizaje supervisado

2. Aprendizaje por refuerzo

Es un tipo de aprendizaje más lento que el de corrección por error, la particularidad de este método es que no se conocen las salidas deseada para cada patrón de entrada, más bien se conoce cómo debería ser el comportamiento general del sistema ante la presentación de diferentes entradas. La función del agente externo o supervisor es indicar mediante una señal de refuerzo si la salida obtenida por la red se ajusta o no a la salida. [10]

3. Aprendizaje estocástico

Este tipo de aprendizaje consta de ir modificando el valor de los pesos de las conexiones aleatoriamente y evaluar su efecto a través del objetivo deseado y de distribuciones de probabilidades. La red Bolztman Machine utiliza este tipo de aprendizaje [11].

2.4.2 Aprendizaje No Supervisado

Este método de aprendizaje no cuenta con un agente externo que determine la salida deseada para la red. El aprendizaje consiste en presentar patrones de entrada a la red sin que conozca la salida, es decir, la red debe ser capaz de reconocer en los patrones presentados similitudes, regularidades, características o categorías que se puedan establecer en los datos presentados y poder hacer una clasificación encontrando similitudes en los patrones. Entre sus características se encuentra que generalmente se utilizan con una arquitectura simple, necesitan menos tiempo de entrenamiento que las redes que utilizan aprendizaje supervisado, y las redes muestran un grado de auto organización. En la Figura 2-8 se puede apreciar un esquema que representa el aprendizaje no supervisado.

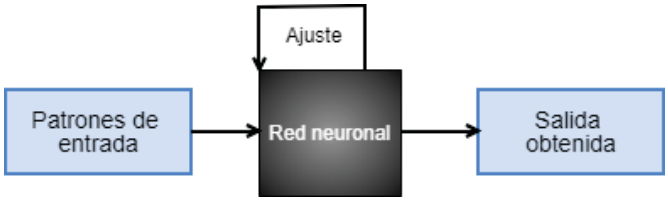

Figura 2-8: Aprendizaje no supervisado

2.5 Funciones de Activación

Las funciones de activación se encargan de calcular el estado de actividad de una neurona transformando la entrada total en un valor de activación cuyo rango generalmente se encuentra entre [0; 1] o de [-1; 1], estos valores indican que la neurona puede estar activa (1) o inactiva (-1 o 0). A continuación, se presentan las funciones de activación más utilizadas en las redes neuronales artificiales. En la Figura 2-9 se puede apreciar la forma característica de la función escalón. En la Figura 2-10 se puede observar la gráfica de la función lineal. En la Figura 2-11 se puede apreciar la curva característica de una función Sigmoidal.

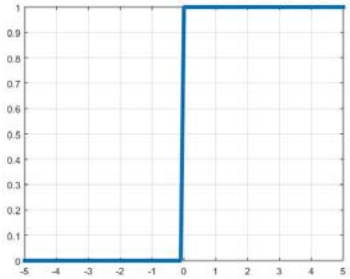

Figura 2-9: Función Escalón

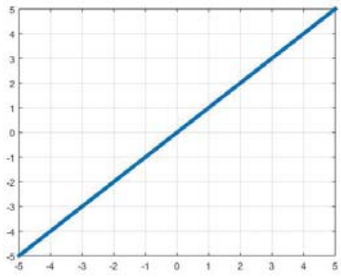

Figura 2-10: Función Lineal

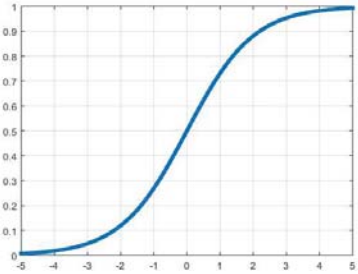

Figura 2-11: Función Sigmoidal

2.5.1 Función escalón

Esta función de activación se caracteriza principalmente porque crea neuronas que clasifican los patrones de entrada en dos categorías diferentes, esta característica la hace particularmente útil en la elaboración de la red perceptrón simple. La ecuación (2-2) representa la expresión matemática de la función escalón.

$$f(x) = \begin{cases} 1 \; si \; x \geq 0 \\ 0 \; si \; x \; < 0 \end{cases} \tag{2-2}$$

2.5.2 Función Lineal

Este tipo de función es utilizada en la red de tipo Adaline y su aplicación principalmente es en el área del procesamiento de señales. Generalmente se utilizan para la elaboración de filtros debido a que éstas son capaces de eliminar ruido en señales portadoras [12]. La ecuación (2-3) representa la expresión matemática de la función Lineal.

$$f(x) = x \tag{2-3}$$

2.5.3 Función Sigmoidal

La función Sigmoidal es una de las más utilizadas en las redes neuronales de tipo perceptrón multicapa debido a la flexibilidad y rango de resultados que esta ofrece, ésta función permite a las redes aprender variaciones no lineales. La ecuación (2-4) representa la expresión matemática de la función Sigmoidal.

$$f(x) = \frac{1}{1 + e^{-x}} \tag{2-4}$$

2.6 Bias

El Bias actúa como un nodo en una red neuronal artificial que siempre está activo, es decir, su valor se establece en 1 sin tener en cuenta los patrones de entrada. Al agregar Bias se aumenta la flexibilidad para ajustar los pesos, en general, permite que la red ajuste los pesos cuando todas las entradas son iguales a 0. Por lo general, se agrega un solo Bias para la capa de entrada y uno por capa oculta que exista en la red, en la capa de salida no es necesario, también es posible no agregarlo en las capas ocultas, todo va a depender del diseño de la red. El Bias no es un término de una red neuronal artificial, más bien, es un término algebraico a considerar, de la ecuación (2-1) se puede determinar la ecuación de la recta que está definida de la siguiente forma: $y = mx + c$; donde c es representado por Bias. Si este valor fuese 0, la recta definida pasaría siempre por el origen, por lo tanto, el valor del Bias puede tener cualquier valor y su funcionalidad recae en poder mover la gráfica y de esta forma resolver situaciones más complejas. En la Figura 2-12 se puede ver la estructura de una red neuronal artificial considerando un Bias por cada capa, también se puede observar, que al Bias no le llega información de las neuronas, más bien, éste nodo aporta información al resto de la red.

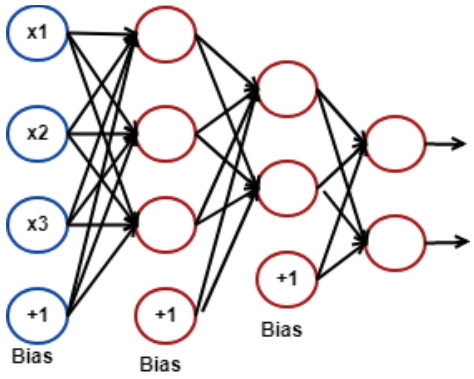

Figura 2-12 Estructura de una red neuronal artificial con Bias por capas

2.7 Tasa de aprendizaje

Un dato importante a considerar es la tasa de aprendizaje (α), ésta determina la velocidad de aprendizaje de la red y su valor puede variar entre $0 < \alpha \leq 1$. Si se escoge un valor muy cercano a uno, la convergencia de la red será más rápida, pero se perderán valores óptimos donde el error es mínimo, en caso contrario, si se escoge un valor cercano a cero, tomará más tiempo en converger la red, pero sin saltarse valores óptimos donde el error es mínimo. Generalmente, se suelen usar valores más cercanos a cero, en los cuales el tiempo de convergencia de la red tome más tiempo, pero la ventaja es que los pesos de las conexiones se pueden adaptar de mejor forma, sin perder valores óptimos. En caso contrario, para probar el funcionamiento del algoritmo en la red, se puede utilizar un valor igual a 1 y de esta forma, lograr una convergencia rápida.

3 Perceptrón Simple

El modelo perceptrón simple se creó como un sistema que fuese capaz de realizar tareas de clasificación de forma automática, la principal idea de concebir este modelo consistía en que, a base de un conjunto de patrones de entrada presentados a la red, ésta fuese capaz de determinar ecuaciones de las superficies que hacían de frontera de dichos patrones. Los patrones de entrenamiento son los que aportan la información necesaria para que la red construya dichas superficies, y también cumpla la función de discriminador o clasificador. El modelo al finalizar el proceso de entrenamiento es capaz de determinar a qué clase pertenece cualquier ejemplo nuevo.

3.1 Arquitectura Perceptrón Simple

La arquitectura de la red se basa en una estructura monocapa, donde existe un conjunto de neuronas de entradas que se dedican a recibir la información que viene del exterior. El número de neuronas que se requieren en la entrada va a estar determinadas por el tipo de problema que se desee a resolver, del mismo modo para las neuronas de la capa de salida. Las neuronas de entradas están totalmente conectadas a las neuronas de la capa de salida, y estas últimas son las que determinan las superficies de discriminación del sistema.

En la Figura 3-1 se presenta el modelo de un perceptrón simple con dos entradas y una salida. Donde x_1 y x_2 son las entradas de la red, mientras que la salida corresponde a Y. Los pesos de las conexiones son w_1 y w_2. Se considera un nodo Bias en la capa de entrada, para lograr una convergencia más rápida de la red, y su utilización va a depender del diseñador de la red, este parámetro adicional generalmente es usado con valor 1. La neurona de salida es la encargada de realizar todo el procesamiento matemático que luego es evaluado en una función de activación. La regla del perceptrón es responder +1 si el patrón de entrada pertenece a la clase A, ó 0 (también puede ser -1, dependiendo de la función de activación) si el patrón de entrada pertenece a la clase B.

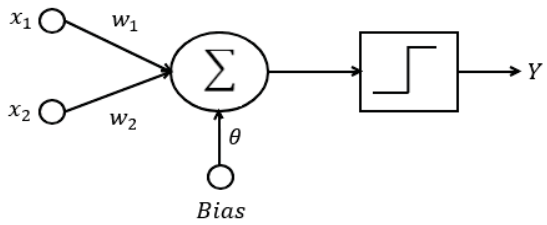

Figura 3-1: Arquitectura Perceptrón simple

La salida de la red está determinada por la ecuación (3-1).

$$y = f\left(\sum_{i=1}^{n} x_i w_i + \theta\right) \tag{3-1}$$

Donde f corresponde a la función de activación, para el caso del perceptrón simple, éste utiliza la función de activación escalón.

3.2 Función de Activación Escalón

La función de activación utilizada por el perceptrón simple es la función escalón como se mencionó anteriormente. Las neuronas con este tipo de función suelen llamarse binarias debido a que en su salida se pueden distinguir dos niveles, estos pueden ser 1 ó 0, como también 1 ó -1. Esta función es de gran utilidad en este modelo, debido a que el perceptrón simple al tratarse de un discriminador de clases, una función binaria puede ser fácilmente implementada para resolver problemas de clasificación para dos categorías o clases. Utilizando la ecuación (3-1) se puede clasificar de la siguiente forma: Si en la salida de la red se obtiene un 1, la entrada pertenece a la categoría A; Si en la salida de la red se obtiene un 0, la entrada pertenece a la categoría B. (ver figura 3-2)

La ecuación (3-1) se transforma en la ecuación de la recta como se puede ver en la ecuación (3-2).

$$x_1 w_1 + x_2 w_2 + \theta = 0 \tag{3-2}$$

La pendiente de la ecuación (3-2) es $m = -w_1/w_2$ y en el origen de ordenada pasa por $-\theta/w_1$. Por lo tanto, la red define una recta que, al ser solución al problema, éste discriminará entre las clases existentes en los patrones de entrenamiento. El perceptrón simple sólo puede discriminar entre dos clases linealmente separables, es decir, cuyas regiones de decisión que puedan ser separadas mediante un hiperplano, como se aprecia en la Figura 3-2.

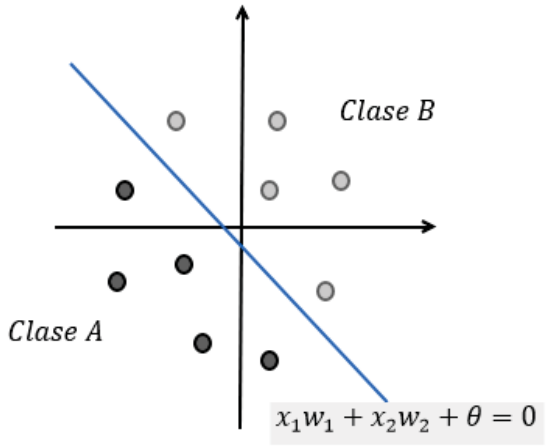

Figura 3-2: Separación en dos clases

3.3 Algoritmo de Aprendizaje

El algoritmo utilizado por el perceptrón es de tipo supervisado y trata de un aprendizaje por corrección de errores. Para el proceso de entrenamiento se debe disponer de un conjunto de patrones de entrada, de los cuales se debe conocer a qué categoría pertenecen, es decir, por cada patrón de entrada que se ingrese a la red se debe conocer su salida. Se debe determinar la ecuación del hiperplano que clasifica a los patrones en distintas clases, esta ecuación se deduce de los ejemplos que forman el conjunto de entrenamiento, y este proceso se realiza paulatinamente modificando los pesos de todas las conexiones hasta encontrar los valores que determinan dicha ecuación. Esta ecuación está definida por: $x_1w_1 + x_2w_2 + \cdots + x_nw_n + \theta = 0$ y es la encargada en hacer la clasificación en dos tipos, como se puede apreciar en la figura (3-2) [13]. La actualización de los pesos de cada una de las conexiones de la red se va a realizar si es que la salida del perceptrón es incorrecta al responder a un patrón de entrada, en caso contrario, si la red entrega en su salida el valor deseado, estos no se modificarán. Por lo tanto, el proceso de aprendizaje consiste en ir modificando los pesos de las conexiones de la red siempre y cuando exista un error producido por la diferencia entre la salida deseada y la salida entregada por la red, si no hay error en la salida, no se modifican los pesos.

3.3.1 Descripción del algoritmo

El algoritmo del perceptrón puede ser determinado de la siguiente forma:

1. **Inicialización de los pesos y Bias:** Inicialmente se asignan valores aleatorios a cada uno de los pesos de las conexiones w_i y al Bias ($w_0 = \theta$). El Bias es equivalente a un peso adicional y este se escribe como w_0 cuya entrada es 1 ($x_0 = 1$).
2. **Ingreso de los patrones de entrada:** El conjunto de patrones de entrada deben ser ingresados uno a uno y se debe conocer su salida, de esta forma la red puede comparar con la salida deseada y obtener un error.
3. **Cálculo de la salida de la red:** Para determinar la salida de la red, se hace uso de la ecuación (3-1), donde f es la función escalón.
4. **Actualización de los pesos:** Si existe error en la salida de la red, los pesos de las conexiones deben ser actualizados mediante la ecuación (3-3).

$$w_i(t+1) = w_i(t) + \alpha \cdot [d(t) - y(t)] \cdot x_i(t) \tag{3-3}$$

   Donde:
   $w_i(t+1)$: peso actualizado
   w_i: peso actual
   α: tasa de aprendizaje.
   $d(t) - y(t)$: error de la red, diferencia entre salida deseada y de la red.
   x_i: parámetro de entrada
5. El proceso se repite desde el punto 2 hasta que finalice el entrenamiento de la red.

3.3.2 Diagrama de bloques del algoritmo

En la Figura 3-3 se describe mediante un diagrama de bloques el algoritmo utilizado por el perceptrón simple.

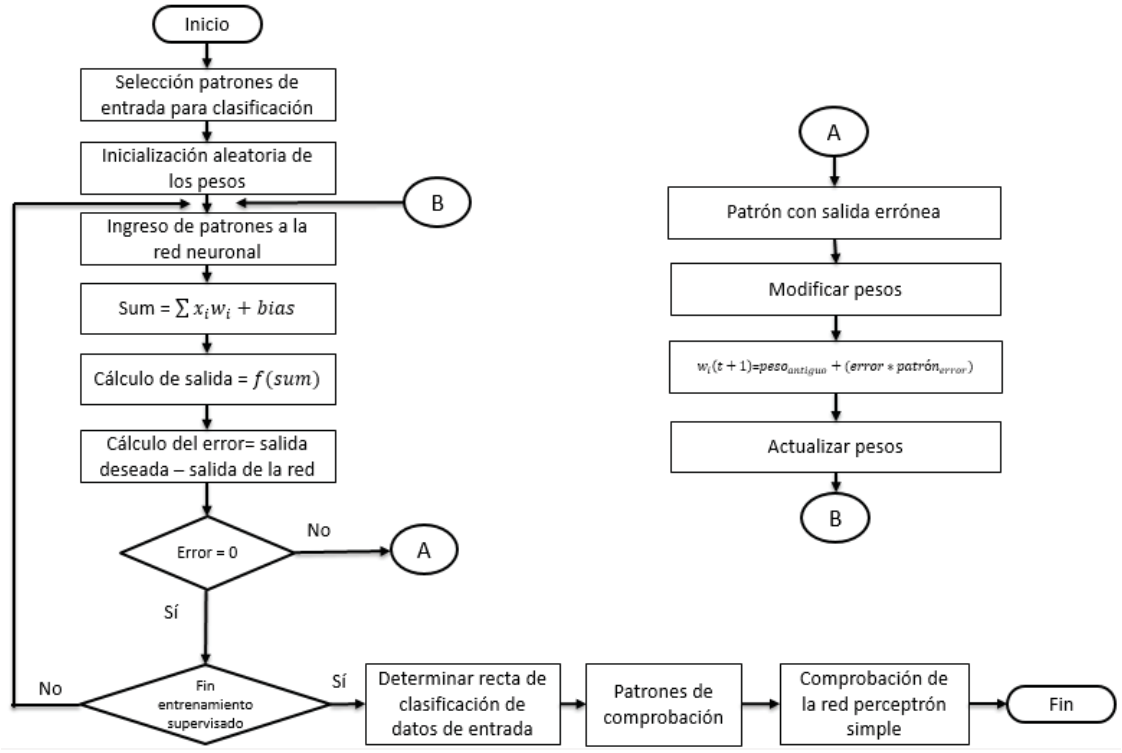

Figura 3-3 Diagrama de bloques del algoritmo del perceptrón simple

3.4 Ejemplo de aprendizaje con algoritmo

A continuación, se presentan tres ejemplos para probar el funcionamiento del algoritmo del perceptrón simple, éstos ejemplos son tratados mediante las tablas de verdad de las compuertas lógicas: OR, AND y XOR. Las funciones lógicas OR y AND son linealmente separable, mientras que XOR no lo es.

3.4.1 Función lógica OR

Para representar el aprendizaje del perceptrón simple, se utiliza la función lógica OR. De esta función se conocen las salidas para cada patrón de entrada, si las entradas son 0 en la salida se obtiene un nivel bajo (0), si una de las entradas o ambas son 1 en la salida se obtiene un nivel alto (1). Si la red en su salida entrega un valor distinto al conocido por la función OR, se deben modificar los pesos de las conexiones, y una vez actualizados los pesos, se debe reingresar el patrón para comprobar que la red aprendió correctamente dicho patrón. Se utiliza la función de activación escalón y una tasa de aprendizaje $\alpha = 1$.

A continuación, se muestra en la Tabla 3-1 los valores que identifican a la función lógica OR con la cual se realiza el proceso de aprendizaje para el perceptrón simple.

Tabla 3-1: Función lógica OR

x_1	x_2	y
0	0	0
0	1	1
1	0	1
1	1	1

Primeramente, la inicialización de los valores de los pesos es aleatoria, incluyendo el valor del Bias que su peso se representa con w_0. En la Figura 3-4 se puede observar el modelo del perceptrón simple con sus valores iniciales. Para el ejemplo se escogieron los siguientes valores aleatorios: $w_0 = -1.5; w_1 = 0.5;\ w_2 = 1$

1. **Primer patrón (0,0)**

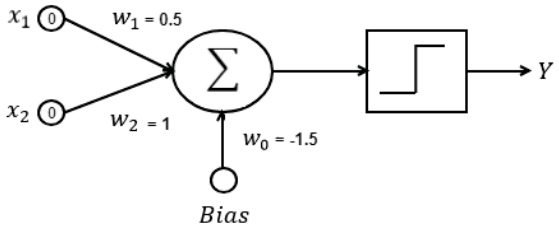

Figura 3-4: Perceptrón simple con valores iniciales

Se calcula la salida de la red mediante la ecuación (3-1).

$$y = f\left(\sum_{i=1}^{n} x_i w_i + \theta\right) = f([0 \cdot\ 0.5] + [0 \cdot\ 1] + [1 \cdot\ -1.5]) = f(0 + 0 - 1.5) = f(-1.5) = 0$$

Luego de obtener la salida de la red, se compara con la salida deseada, si esta presenta un error se deben modificar los pesos, en caso contrario los pesos se mantienen y se debe ingresar el siguiente patrón.

$$error = d(t) - y(t) = 0 - 0 = 0$$

2. **Segundo patrón (0,1)**

En la Figura 3-5 se observa el perceptrón simple con los valores del segundo patrón de entrada, pero conservando el valor de los pesos debido a que el error fue cero en la iteración anterior.

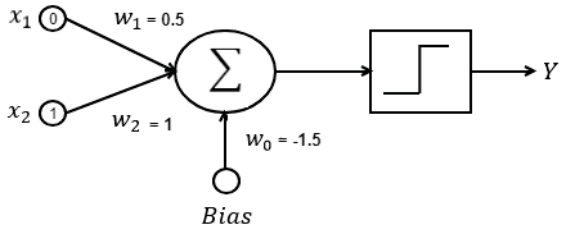

Figura 3-5: Perceptrón con segundo patrón

Se utiliza la ecuación (3-1) para obtener la salida de la red.

$$y = f\left(\sum_{i=1}^{n} x_i w_i + \theta\right) = f([0 \cdot \ 0.5] + [1 \cdot \ 1] + [1 \cdot \ -1.5]) = f(0 + 1 - 1.5) = f(-0.5) = 0$$

$$error = d(t) - y(t) = 1 - 0 = 1$$

Al obtener un error se deben modificar los pesos de las conexiones y volver a ingresar el patrón para comprobar que la red aprendió dicho patrón. En la Tabla 3-2 se ven los valores de los nuevos pesos de las conexiones al utilizar la ecuación (3-3).

Tabla 3-2 Actualización de los pesos

$w_0(t+1) = w_0(t) + \alpha \cdot [\mathrm{d(t)} - \mathrm{y(t)}] \cdot \ x_0(t) = -1.5 + 1 \cdot [1] \cdot \ 1 = -0.5$
$w_1(t+1) = w_1(t) + \alpha \cdot [\mathrm{d(t)} - \mathrm{y(t)}] \cdot \ x_1(t) = 0.5 + 1 \cdot [1] \cdot \ 0 = 0.5$
$w_2(t+1) = w_1(t) + \alpha \cdot [\mathrm{d(t)} - \mathrm{y(t)}] \cdot \ x_2(t) = 1 + 1 \cdot [1] \cdot \ 1 = 2$

Luego de la actualización de los pesos, se debe volver a ingresar el mismo patrón y comprobar si se obtiene error. En la Figura 3-6 se puede apreciar el perceptrón simple con los nuevos valores de los pesos.

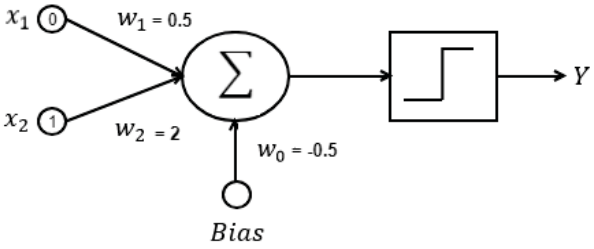

Figura 3-6: Perceptrón con pesos actualizados

Se utiliza la ecuación (3-1) para obtener la salida de la red con los nuevos pesos.

$$y = f\left(\sum_{i=1}^{n} x_i w_i + \theta\right) = f([0 \cdot \ 0.5] + [1 \cdot \ 2] + [1 \cdot \ -0.5]) = f(0 + 2 - 0.5) = f(1.5) = 1$$

$$error = d(t) - y(t) = 1 - 1 = 0$$

Al obtener error cero se puede pasar al siguiente patrón de entrada.

3. Tercer patrón (1,0)

En la Figura 3-7 se puede observar el perceptrón simple con el tercer patrón de entrada.

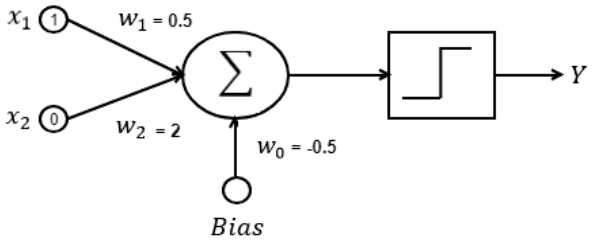

Figura 3-7: Perceptrón con tercer patrón de entrada

Se utiliza la ecuación (3-1) para obtener la salida de la red.

$$y = f\left(\sum_{i=1}^{n} x_i w_i + \theta\right) = f([1 \cdot \ 0.5] + [0 \cdot \ 2] + [1 \cdot \ -0.5]) = f(0.5 + 0 - 0.5) = f(0) = 1$$

$$error = d(t) - y(t) = 1 - 1 = 0$$

Al obtener error cero se puede pasar al último patrón de entrada.

4. Cuarto patrón (1,1)

En la Figura 3-8 se puede apreciar el perceptrón simple con el último patrón de entrada

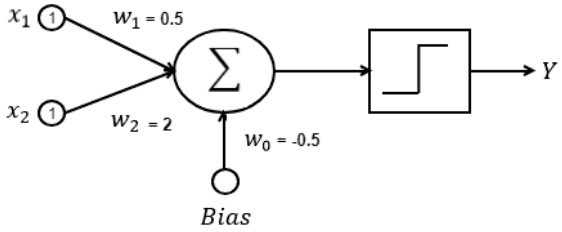

Figura 3-8: Perceptrón con cuarto patrón de entrada

Se utiliza la ecuación (3-1) para obtener la salida de la red.

$$y = f\left(\sum_{i=1}^{n} x_i w_i + \theta\right) = f([1 \cdot \ 0.5] + [1 \cdot \ 2] + [1 \cdot \ -0.5]) = f(0.5 + 2 - 0.5) = f(2) = 1$$

$$error = d(t) - y(t) = 1 - 1 = 0$$

Finalmente, después de ingresar el último patrón, es importante comprobar que los últimos pesos calculados sirven para cada uno de los patrones de entrada, como se puede ver en la Tabla 3-3.

Terminada la fase de entrenamiento, la red queda con los pesos fijos. En la Tabla 3-3 se presenta el proceso de iteración que hizo la red.

Tabla 3-3: Proceso de Iteración

Iteración	x_1	x_2	$d(t)$	w_0	w_1	w_2	y	error	$w_0(t+1)$	$w_1(t+1)$	$w_2(t+1)$
1	0	0	0	-1.5	0.5	1	0	0	-	-	-
	0	1	1	-1.5	0.5	1	0	1	-0.5	0.5	2
2	0	1	1	-0.5	0.5	2	1	0	-	-	-
	1	0	1	-0.5	0.5	2	1	0	-	-	-
	1	1	1	-0.5	0.5	2	1	0	-	-	-
	0	0	0	-0.5	0.5	2	0	0	-	-	-

La red define una recta que, en el caso de ser solución al problema ésta separará entre las clases existentes en los patrones de entrada. Este proceso iterativo forma parte del entrenamiento y la modificación de los pesos se realiza cada vez que la red encuentra un error en la salida, este proceso concluye una vez que el error sea cero. En la Figura 3-9 se muestra la clasificación que hizo la red para los patrones de entrada de una función lógica OR, con ecuación de la recta $0.5x_1 + 2x_2 - 0.5 = 0$ encontrada al modificar los pesos en la fase de entrenamiento. Por lo tanto, el perceptrón simple clasifica en dos categorías los patrones de entradas, como se puede apreciar a continuación.

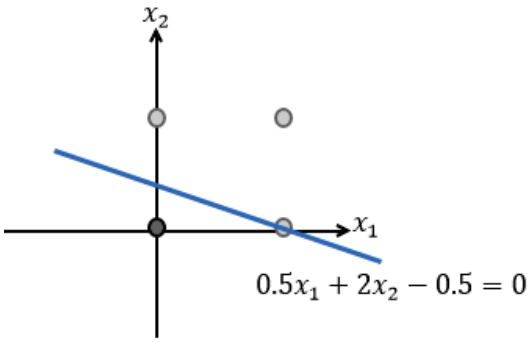

Figura 3-9: Recta determinada por el perceptrón simple después de la última iteración

3.4.2 Función Lógica AND

En la Tabla 3-4 se puede apreciar la compuerta lógica AND, ésta se caracteriza por presentar en su salida un nivel bajo (0), cada vez que una de sus entradas presenta un nivel bajo, si ambas entradas están en un nivel alto (1) la salida será un nivel alto. Para el ejemplo se escogieron los siguientes valores aleatorios: $w_0 = -2; w_1 = 1.5;\ w_2 = 2.5$ con $\alpha = 1$.

Tabla 3-4 Función lógica AND

x_1	x_2	y
0	0	0
0	1	0
1	0	0
1	1	1

1. **Primer patrón (0,0)**

En la Figura 3-10 se puede observar el perceptrón simple con el primer patrón de entrada y los valores iniciales.

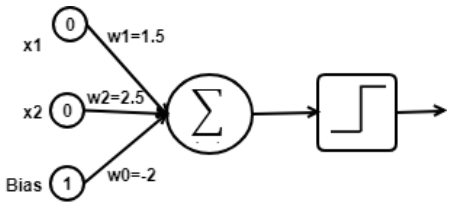

Figura 3-10 Perceptrón con valores inciales

Se utiliza la ecuación (3-1) para obtener la salida de la red.

$$y = f\left(\sum_{i=1}^{n} x_i w_i + \theta\right) = f([0 \cdot \ 1.5] + [0 \cdot \ 2.5] + [1 \cdot \ -2]) = f(0 + 0 - 2) = f(-2) = 0$$

$$error = d(t) - y(t) = 0 - 0 = 0$$

2. **Segundo patrón (0,1)**

En la Figura 3-11 se puede observar el perceptrón simple con el segundo patrón de entrada.

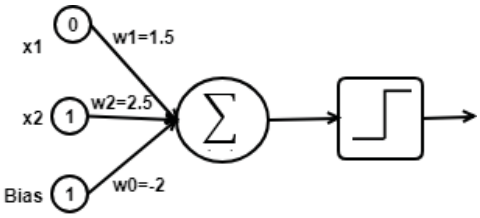

Figura 3-11 Perceptrón con segundo patrón de entrada

Se utiliza la ecuación (3-1) para obtener la salida de la red.

$$y = f\left(\sum_{i=1}^{n} x_i w_i + \theta\right) = f([0 \cdot \ 1.5] + [1 \cdot \ 2.5] + [1 \cdot \ -2]) = f(0 + 2.5 - 2) = f(0.5) = 1$$

$$error = d(t) - y(t) = 0 - 1 = -1$$

Al obtener un error en la salida se deben modificar los pesos de las conexiones mediante la ecuación (3-3), como se aprecia en la Tabla 3-5.

Tabla 3-5 Actualización de los pesos

$w_0(t+1) = w_0(t) + \alpha \cdot [\mathrm{d(t)} - \mathrm{y(t)}] \cdot x_0(t) = -2 + 1 \cdot [-1] \cdot 1 = -3$
$w_1(t+1) = w_1(t) + \alpha \cdot [\mathrm{d(t)} - \mathrm{y(t)}] \cdot x_1(t) = 1.5 + 1 \cdot [-1] \cdot 0 = 1.5$
$w_2(t+1) = w_1(t) + \alpha \cdot [\mathrm{d(t)} - \mathrm{y(t)}] \cdot x_2(t) = 2.5 + 1 \cdot [-1] \cdot 1 = 1.5$

Una vez actualizados los pesos, se debe volver a calcular la salida de la red con el mismo patrón de entrada, y de esta forma poder corroborar el buen funcionamiento de la red. En la Figura 3-12, se puede observar el perceptrón simple con los pesos actualizados.

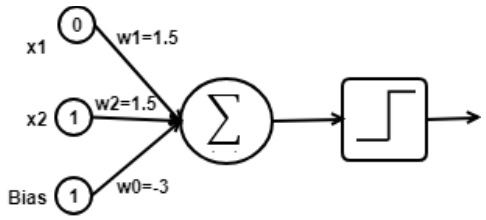

Figura 3-12 Perceptrón con valores actualizados

Se calcula la salida mediante la ecuación (3-1).

$$y = f\left(\sum_{i=1}^{n} x_i w_i + \theta\right) = f([0 \cdot 1.5] + [1 \cdot 1.5] + [1 \cdot -3]) = f(0 + 1.5 - 3) = f(-1.5) = 0$$

$$error = d(t) - y(t) = 0 - 0 = 0$$

3. Tercer patrón (1,0)

En la Figura 3-13 se puede observar el perceptrón simple con el tercer patrón de entrada.

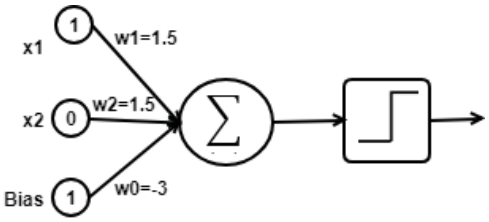

Figura 3-13 Perceptrón con tercer patrón de entrada

Se utiliza la ecuación (3-1) para obtener la salida de la red.

$$y = f\left(\sum_{i=1}^{n} x_i w_i + \theta\right) = f([1 \cdot 1.5] + [0 \cdot 1.5] + [1 \cdot -3]) = f(1.5 + 0 - 3) = f(-1.5) = 0$$

$$error = d(t) - y(t) = 0 - 0 = 0$$

4. Cuarto patrón (1,1)

En la Figura 3.14 se puede observar el perceptrón simple con el cuarto patrón de entrada.

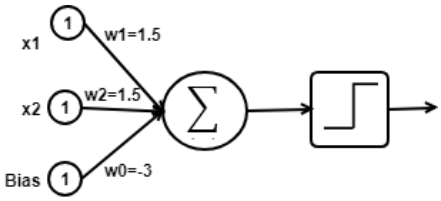

Figura 3-14 Perceptrón con cuarto patrón de entrada

Se utiliza la ecuación (3-1) para obtener la salida de la red.

$$y = f\left(\sum_{i=1}^{n} x_i w_i + \theta\right) = f([1 \cdot\ 1.5] + [1 \cdot\ 1.5] + [1 \cdot\ -3]) = f(1.5 + 1.5 - 3) = f(0) = 1$$

$$error = d(t) - y(t) = 1 - 1 = 0$$

Es importante comprobar que los pesos calculado sirvan para cada uno de los patrones de entrada, como se puede observar en la Tabla 3-6, se puede observar todas las iteraciones realizadas para obtener los pesos de la red.

Tabla 3-6 Proceso de Iteración compuerta AND

Iteración	x_1	x_2	$d(t)$	w_0	w_1	w_2	y	error	$w_0(t+1)$	$w_1(t+1)$	$w_2(t+1)$
1	0	0	0	-2	1.5	2.5	0	0	-	-	-
	0	1	0	-2	1.5	2.5	1	-1	-3	1.5	1.5
2	0	1	1	-3	1.5	1.5	1	0	-	-	-
	1	0	1	-3	1.5	1.5	0	0	-	-	-
	1	1	1	-3	1.5	1.5	0	0	-	-	-
	0	0	0	-3	1.5	1.5	0	0	-	-	-

Finalmente, el sistema genera la siguiente ecuación de la recta $1.5x_1 + 1.5x_2 - 3 = 0$, que es la encargada de la clasificación de los patrones de entrada en dos categorías, como se puede apreciar en la Figura 3.15.

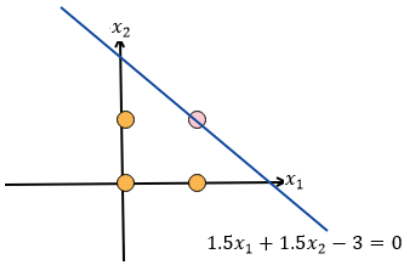

Figura 3-15 Recta determinada por el perceptrón para una función AND

El perceptrón simple puede separar en dos categorías las entradas que sean linealmente separables como es el caso de las compuertas lógicas OR y AND. Al ser una red que utiliza aprendizaje supervisado, es necesario conocer las salidas de cada uno de los patrones que ingresan a la red, y de esta forma determinar el error que se genera al comparar la salida que obtuvo el sistema con la salida deseada. Cada vez que se genere un error en la salida, la red va a modificar los pesos de sus conexiones mediante el uso su algoritmo, hasta que en la salida el error sea cero. Al ingresar el último patrón de entrada y adaptar los pesos de la red, ésta va a definir una recta que separará en dos categorías los patrones de entrada, quedando fijos los pesos de las conexiones después de la última iteración.

3.4.3 Función Lógica XOR

Para demostrar el problema que presenta el perceptrón simple referente a clasificar entradas que no sean linealmente separable, se va a hacer un análisis de la compuerta lógica XOR utilizando los mismos valores iniciales que en el ejemplo de la función lógica OR. $w_0 = -1.5; w_1 = 0.5;\ w_2 = 1$ con tasa de aprendizaje $\alpha = 1$. En la Tabla 3-7 se puede ver la función lógica XOR

Tabla 3-7 Función lógica XOR

x_1	x_2	y
0	0	0
0	1	1
1	0	1
1	1	0

1. **Primer patrón (0,0)**

En la Figura 3-16 se puede apreciar el perceptrón simple con sus valores iniciales.

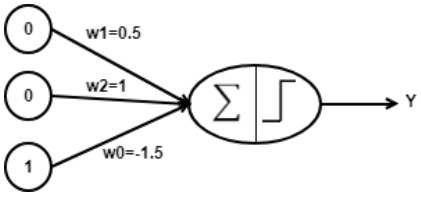

Figura 3-16 Perceptrón simple con los valores iniciales.

Se calcula la salida de la red mediante la ecuación (3-1).

$$y = f\left(\sum_{i=1}^{n} x_i w_i + \theta\right) = f([0 \cdot \ 0.5] + [0 \cdot \ 1] + [1 \cdot \ -1.5]) = f(0 + 0 - 1.5) = f(-1.5) = 0$$

$$error = d(t) - y(t) = 0 - 0 = 0$$

Al obtener error igual a cero se puede pasar al siguiente patrón.

2. Segundo patrón (0,1)

En la Figura 3-17 se puede apreciar el perceptrón simple con el segundo patrón, conservando los valores de los pesos anteriores.

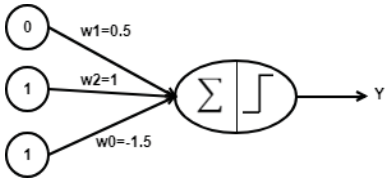

Figura 3-17 Perceptrón con el segundo patrón de entrada

Se calcula la salida de la red mediante la ecuación (3-1)

$$y = f\left(\sum_{i=1}^{n} x_i w_i + \theta\right) = f([0 \cdot \ 0.5] + [0 \cdot \ 1] + [1 \cdot \ -1.5]) = f(0 + 0 - 1.5) = f(-1.5) = 0$$

$$error = d(t) - y(t) = 1 - 0 = 1$$

Al obtener error en la salida, se deben modificar los pesos de las conexiones, como se puede apreciar en la Tabla 3-8.

Tabla 3-8 Actualización de los pesos

$w_0(t+1) = w_0(t) + \alpha \cdot [\mathrm{d(t) - y(t)}] \cdot \ x_0(t) = -1.5 + 1 \cdot [1] \cdot \ 1 = -0.5$
$w_1(t+1) = w_1(t) + \alpha \cdot [\mathrm{d(t) - y(t)}] \cdot \ x_1(t) = 0.5 + 1 \cdot [1] \cdot \ 0 = 0.5$
$w_2(t+1) = w_1(t) + \alpha \cdot [\mathrm{d(t) - y(t)}] \cdot \ x_2(t) = 1 + 1 \cdot [1] \cdot \ 1 = 2$

En la Figura 3-18 se presenta el perceptrón con los pesos actualizados.

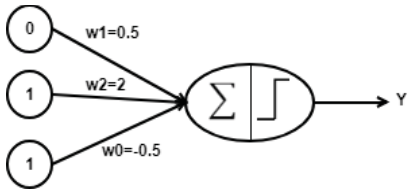

Figura 3-18 Perceptrón con pesos actualizados

Se calcula la salida de la red mediante la ecuación (3-1)

$$y = f\left(\sum_{i=1}^{n} x_i w_i + \theta\right) = f([0 \cdot \ 0.5] + [1 \cdot \ 2] + [1 \cdot \ -0.5]) = f(0 + 2 - 0.5) = f(1.5) = 1$$

$$error = d(t) - y(t) = 1 - 1 = 0$$

3. Tercer patrón (1,0)

En la Figura 3-19 se presenta el perceptrón con el tercer patrón de entrada.

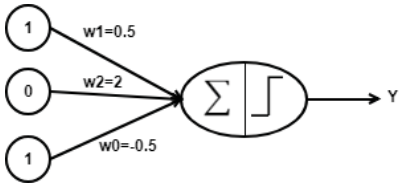

Figura 3-19 Perceptrón con tercer patrón de entrada

Se calcula la salida de la red mediante la ecuación (3-1).

$$y = f\left(\sum_{i=1}^{n} x_i w_i + \theta\right) = f([1 \cdot \ 0.5] + [0 \cdot \ 2] + [1 \cdot \ -0.5]) = f(0.5 + 0 - 0.5) = f(0) = 1$$

$$error = d(t) - y(t) = 1 - 1 = 0$$

Al no obtener error se puede pasar al último patrón de entrada.

4. Cuarto patrón (1,1)

En la Figura 3-20 se presenta el perceptrón simple con el último patrón de entrada.

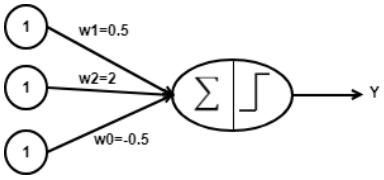

Figura 3-20 Perceptrón con cuarto patrón de entrada

Se utiliza la ecuación (3-1) para determinar la salida de la red.

$$y = f\left(\sum_{i=1}^{n} x_i w_i + \theta\right) = f([1 \cdot\ 0.5] + [1 \cdot\ 2] + [1 \cdot\ -0.5]) = f(0.5 + 2 - 0.5) = f(2) = 1$$

$$error = d(t) - y(t) = 0 - 1 = -1$$

Al obtener error se deben calcular los nuevos pesos como se puede ver en la Tabla 3-9 utilizando la ecuación (3-3).

Tabla 3-9 Actualización de los pesos

$w_0(t+1) = w_0(t) + \alpha \cdot [\mathrm{d(t)} - \mathrm{y(t)}] \cdot\ x_0(t) = -0.5 + 1 \cdot [-1] \cdot\ 1 = -1.5$
$w_1(t+1) = w_1(t) + \alpha \cdot [\mathrm{d(t)} - \mathrm{y(t)}] \cdot\ x_1(t) = 0.5 + 1 \cdot [-1] \cdot\ 1 = -0.5$
$w_2(t+1) = w_1(t) + \alpha \cdot [\mathrm{d(t)} - \mathrm{y(t)}] \cdot\ x_2(t) = 2 + 1 \cdot [-1] \cdot\ 1 = 1$

Luego de la actualización de los pesos, se debe volver a ingresar el mismo patrón y comprobar si se obtiene error. En la Figura 21 se puede apreciar el perceptrón simple con el último patrón de entrada y los nuevos pesos calculados.

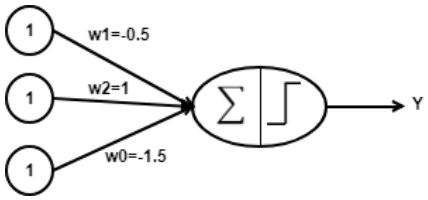

Figura 3-21 Perceptrón con pesos actualizados

Se calcula la salida de la red mediante la ecuación (3-1).

$$y = f\left(\sum_{i=1}^{n} x_i w_i + \theta\right) = f([1 \cdot\ -0.5] + [1 \cdot\ 1] + [1 \cdot\ -1.5]) = f(-0.5 + 1 - 1.5) = f(-1) = 0$$

$$error = d(t) - y(t) = 0 - 0 = 0$$

En la Tabla 3-10 se pueden ver los resultados al analizar cada uno de los patrones de entrada con los últimos pesos obtenidos, se puede apreciar que los pesos calculados en la última iteración no son los necesario para que la red logre la convergencia y también se puede observar que la red comienza a oscilar lo cual no permite que se adapten los pesos. Se puede observar que en la columna “error” esta nunca logra una estabilización, es decir, nunca llega a un valor de cero. Se puede concluir que la red perceptrón simple no es buena para clasificar datos que no sean linealmente separables.

Tabla 3-10 Proceso de iteración función lógica XOR

Iteración	x_1	x_2	$d(t)$	w_1	w_2	w_0	y	error	$w_1(t+1)$	$w_2(t+1)$	$w_0(t+1)$
1	0	0	0	0.5	1	-1.5	0	0	-	-	-
	0	1	1	0.5	1	-1.5	0	1	0.5	2	-0.5
	1	0	1	0.5	1	-1.5	0	1	-	-	-
	1	1	0	0.5	1	-1.5	1	-1	-	-	-
2	0	0	0	0.5	2	-0.5	0	0	-	-	-
	0	1	1	0.5	2	-0.5	1	0	-	-	-
	1	0	1	0.5	2	-0.5	1	0	-	-	-
	1	1	0	0.5	2	-0.5	1	-1	-0.5	1	-1.5
3	0	0	0	-0.5	1	-1.5	0	0	-	-	-
	0	1	1	-0.5	1	-1.5	0	1	-0.5	2	-0.5
	1	0	1	-0.5	1	-1.5	0	1	-	-	-
	1	1	0	-0.5	1	-1.5	0	0	-	-	-
4	0	0	0	-0.5	2	-0.5	0	0	-	-	-
	0	1	1	-0.5	2	-0.5	1	0	-	-	-
	1	0	1	-0.5	2	-0.5	0	1	0.5	2	0.5
	1	1	0	-0.5	2	-0.5	1	-1	-	-	-

4 Perceptrón Multicapa

El perceptrón multicapa es una generalización del perceptrón simple, y nace de las limitaciones presentadas de este último referente al problema de la separabilidad no lineal. Estas limitaciones presentadas por el perceptrón simple hicieron necesario agregar capas intermedias entre la capa de salida y la de entrada, denominándose a este arreglo "capa oculta". El perceptrón multicapa está compuesto por la capa de entrada, capa oculta y capa de salida, donde las neuronas ubicadas en la capa de entrada son las encargadas de propagar la información que viene desde el exterior y propagar dicha información hacia las neuronas de la siguiente capa. Las neuronas de la capa oculta se encargan de realizar un procesamiento no lineal de la información que procesan y finalmente, la capa de salida es la encargada de proporcionar al exterior la respuesta de la red para cada uno de los patrones de entrada.

4.1 Arquitectura Perceptrón Multicapa

En la Figura 4-1 se muestra la arquitectura de un perceptrón multicapa compuesta en su capa de entrada por los patrones x_1 y x_2 más el Bias, la capa oculta está compuesta por tres neuronas y cada una de ellas tiene asociada una función de activación Sigmoidal y finalmente en la capa de salida se encuentra una neurona que tiene asociada una función Sigmoidal.

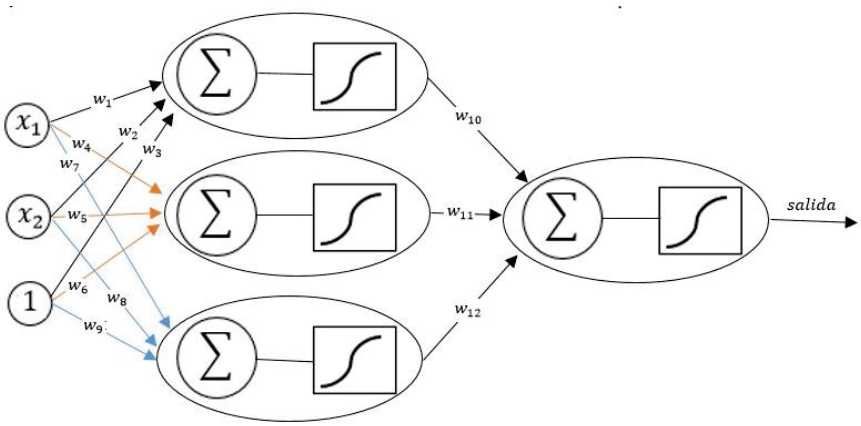

Figura 4-1: Arquitectura Perceptrón Multicapa

4.2 Función de Activación

La función típica utilizada en las redes neuronales artificiales es la Sigmoidal debido a que posee como imagen un rango continuo de valores que varían entre [0; 1]. Esta es una función creciente que presenta dos niveles de saturación, un máximo con valor 1 y un mínimo con valor 0. Una de las particularidades del uso de esta función es que si en hay una entrada nula esta presenta una salida de 0.5, esto quiere decir, que presenta actividad en ausencia de estimulación. Otra de sus cualidades es que puede acomodar señales con altas intensidades sin producir saturación, como también, puede recibir señales de baja intensidad sin excesiva atenuación y finalmente, posee una derivada fácil de calcular.

4.2.1 Función sigmoidal y derivada

La función Sigmoidal debido a la flexibilidad y rango de resultados que esta ofrece permite a las redes aprender variaciones no lineales y es ideal para el uso del perceptrón multicapa.

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

La ecuación (4-1) muestra la derivada de la función Sigmoidal que es utilizada por el algoritmo de retropropagación del perceptrón multicapa para adaptar sus pesos.

$$f'(x) = f(x)(1 - f(x))dx \tag{4-1}$$

4.3 Algoritmo de Retropropagación

El algoritmo utilizado por el perceptrón multicapa es llamado de retropropagación o "backpropagation". Este algoritmo se separa en dos etapas, la primera consiste en hacer una propagación hacia delante de la información proveniente de las neuronas que se encuentran en la capa de entrada, esta información es enviada hacia cada una de las neuronas que se encuentran en la capa siguiente pasando por cada conexión hasta llegar a la capa de salida, en este punto, las neuronas ubicadas en la capa de salida se encargan de generar la respuesta de red, esta respuesta es comparada con la salida deseada generando un error. La segunda parte del algoritmo consiste en propagar este error desde la capa de salida hasta la capa de entrada pasando por todas las conexiones de la red, y a medida que el error se propaga por las conexiones, estas se ven afectadas y van modificando su valor. El objetivo del algoritmo es encontrar los pesos adecuados de cada una de las conexiones de la red para obtener una salida los más parecida de lo que se desea, minimizando el error obtenido.

4.3.1 Propagación hacia adelante

Al igual que el perceptrón simple, se realiza una propagación hacia adelante con la información que viene desde las neuronas de la capa de entrada pasando por cada una de las conexiones hasta las que se encuentran en la capa de salida, obteniendo la respuesta de la red. A continuación, se explica el algoritmo del perceptrón multicapa, mencionando cada una de las ecuaciones que son

utilizadas para la posterior programación del mismo. En la Figura 4-2, se encuentran las siguientes variables:

x_1, x_2, x_o: Entradas de la red

$w_1, w_2, w_3, w_4, w_5, w_6, w_7, w_8$: Pesos de la red

zh_1: Sumatoria de los valores que ingresan a la primera neurona de la capa oculta.

h_1: Salida de la primera neurona de la capa oculta.

zh_2: Sumatoria de los valores que ingresan a la segunda neurona de la capa oculta.

h_2: Salida de la segunda neurona de la capa oculta.

$zout$: Sumatoria de los valores que ingresan a la neurona de la capa de salida.

out: Salida de la red.

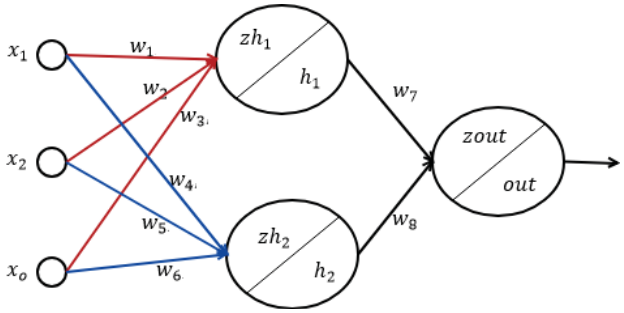

Figura 4-2 Propagación hacia adelante

A continuación, se muestra el procesamiento matemático que realiza cada neurona en la propagación hacia adelante.

1. Primera neurona oculta

$$zh_1 = x_1 \cdot w_1 + x_2 \cdot w_2 + b_1 \cdot 1$$

$$h_1 = \frac{1}{1 + e^{-zh_1}}$$

2. Segunda neurona oculta

$$zh_2 = x_1 \cdot w_3 + x_2 \cdot w_4 + b_2 \cdot 1$$

$$h_2 = \frac{1}{1 + e^{-zh_2}}$$

3. Neurona de salida

$$zout = h_1 \cdot w_5 + h_2 \cdot w_6 + b_2 \cdot 1$$

$$out = \frac{1}{1 + e^{-zo}}$$

4. La ecuación (4-2) muestra el error cuadrático medio

$$Error = \frac{(out - s_d)^2}{2} \tag{4-2}$$

Donde:

s_d: salida deseada.

out: salida de la red.

4.3.2 Propagación hacia atrás

La segunda fase del algoritmo consiste en mover el error a nivel de capas, es decir, primeramente, se debe hacer un recorrido desde la salida hasta la capa oculta y luego se debe hacer un recorrido desde la salida hasta la capa de entrada, de esta forma los pesos se ven afectados y se actualizan con cada iteración realizada. A continuación, se hace un análisis del error a nivel de capas.

4.4 Recorrido del error hasta la capa oculta

Para analizar cómo afecta el error desde la salida hasta la capa oculta, se debe calcular la derivada parcial del error con respecto al peso que corresponda según la neurona que se quiera analizar en la capa oculta, en este caso, se analiza la primera neurona de la capa oculta y se utiliza el peso w_7. Por lo tanto, se calcula la derivada parcial del error con respecto a w_7. En la Figura 4-3 se puede observar cómo el error se mueve a través de las conexiones hasta llegar a la capa oculta. Al utilizar la regla de la cadena se obtiene la fórmula del recorrido del error hasta la capa oculta como se muestra en la ecuación (4-3).

$$\frac{\partial E}{\partial w_7} = \frac{\partial E}{\partial out} \cdot \frac{\partial out}{\partial zout} \cdot \frac{\partial zout}{\partial w_7} \tag{4-3}$$

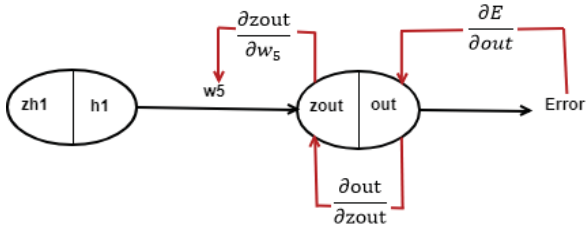

Figura 4-3: Recorrido del error hasta la capa oculta

De la ecuación 4-3 se calcula cada término por separado para determinar el recorrido del error desde la capa de salida hasta la capa oculta como se muestra a continuación:

1. $\partial E/\partial out$

Primero se calcula la derivada parcial del error con respecto a out, considerando el error cuadrático medio como: $E = \frac{(s_d - out)^2}{2}$. Al realizar la derivada se obtiene la ecuación (4-4)

$$\frac{\partial E}{\partial out} = out - s_d \tag{4-4}$$

2. $\partial out/\partial zout$

Para determinar el siguiente término, se debe derivar la función Sigmoidal debido a que es la función que se utiliza en esa neurona y la retropropagación hacia atrás que realiza esta. Considerando la derivada de la función Sigmoidal $f'(x) = f(x)(1 - f(x))dx$, se obtiene la ecuación (4-5)

$$\frac{\partial out}{\partial zout} = out(1 - out) \tag{4-5}$$

3. $\partial zout/\partial w_7$

Considerando la expresión $zout = h_1 w_7 + h_2 w_8$, derivando con respecto a w_7 se obtiene la ecuación (4-6).

$$\frac{\partial zout}{\partial w_7} = h_1 \tag{4-6}$$

4.5 Recorrido del error hasta la capa de entrada

En la Figura 4-4 se puede observar el recorrido del error y para calcular éste error desde la capa de salida hasta la capa de entrada, se debe calcular la derivada parcial del error con respecto a w_1, como se muestra en la ecuación (4-7).

$$\frac{\partial E}{\partial w_1} = \frac{\partial E}{\partial h_1} \cdot \frac{\partial h_1}{\partial zh_1} \cdot \frac{\partial zh_1}{\partial w_1} \tag{4-7}$$

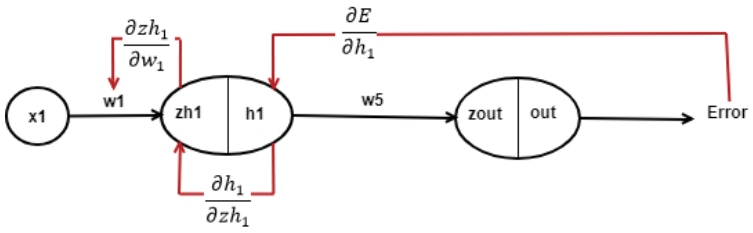

Figura 4-4: Recorrido del error hasta la capa de entrada

De la ecuación (4-7) se calcula cada término por separado para determinar el recorrido del error desde la capa de salida hasta la capa de entrada como se muestra a continuación:

1. $\boldsymbol{\partial E/\partial h_1}$

Se debe realizar la regla de la cadena para determinar este término, por lo tanto, se obtiene $E/\partial h_1 = \partial E/\partial out \cdot \partial out/\partial zout \cdot \partial zout/\partial h_1$ de la cual sólo faltaría determinar $\partial zout/\partial h_1$; Considerando $zout = h_1 w_7 + h_2 w_8$, al derivar con respecto a h_1, se obtiene $\partial zout/\partial h_1 = w_7$. En la ecuación (4-8) se muestra el resultado final del primer término.

$$\frac{\partial E}{\partial h_1} = (out - s_d) \cdot \big(out(1 - out)\big) \cdot w_7 \tag{4-8}$$

2. $\boldsymbol{\partial h_1/\partial zh_1}$

Este término se obtiene derivando la función Sigmoidal, considerando la derivada de la función Sigmoidal $f'(x) = f(x)(1 - f(x))dx$, se obtiene la expresión de la ecuación (4-9).

$$\frac{\partial h_1}{\partial zh_1} = h_1(1 - h_1) \tag{4-9}$$

3. $\boldsymbol{\partial zh_1/\partial w_1}$

Considerando la expresión $zh_1 = x_1 w_1 + x_2 w_2 + x_0 w_3$, derivando con respecto a w_1 se obtiene la ecuación (4-10).

$$\frac{\partial zh_1}{\partial w_1} = x_1 \tag{4-10}$$

4.6 Actualización de los pesos

Primeramente, se debe determinar la derivada del error con respecto a cada peso de la red y luego hacer la actualización mediante la ecuación (4-11).

$$w_i^+ = w_i - \alpha \frac{\partial E}{\partial w_i} \tag{4-11}$$

Donde:

w_i^+: nuevo peso a calcular

w_i : valor antiguo del peso

α : tasa de aprendizaje

$\partial E/\partial w_i$: derivada parcial del error con respecto al peso w_i

4.7 Diagrama de bloques del algoritmo

En la Figura 4-5 se describe mediante un diagrama de bloques el algoritmo utilizado por el perceptrón multicapa.

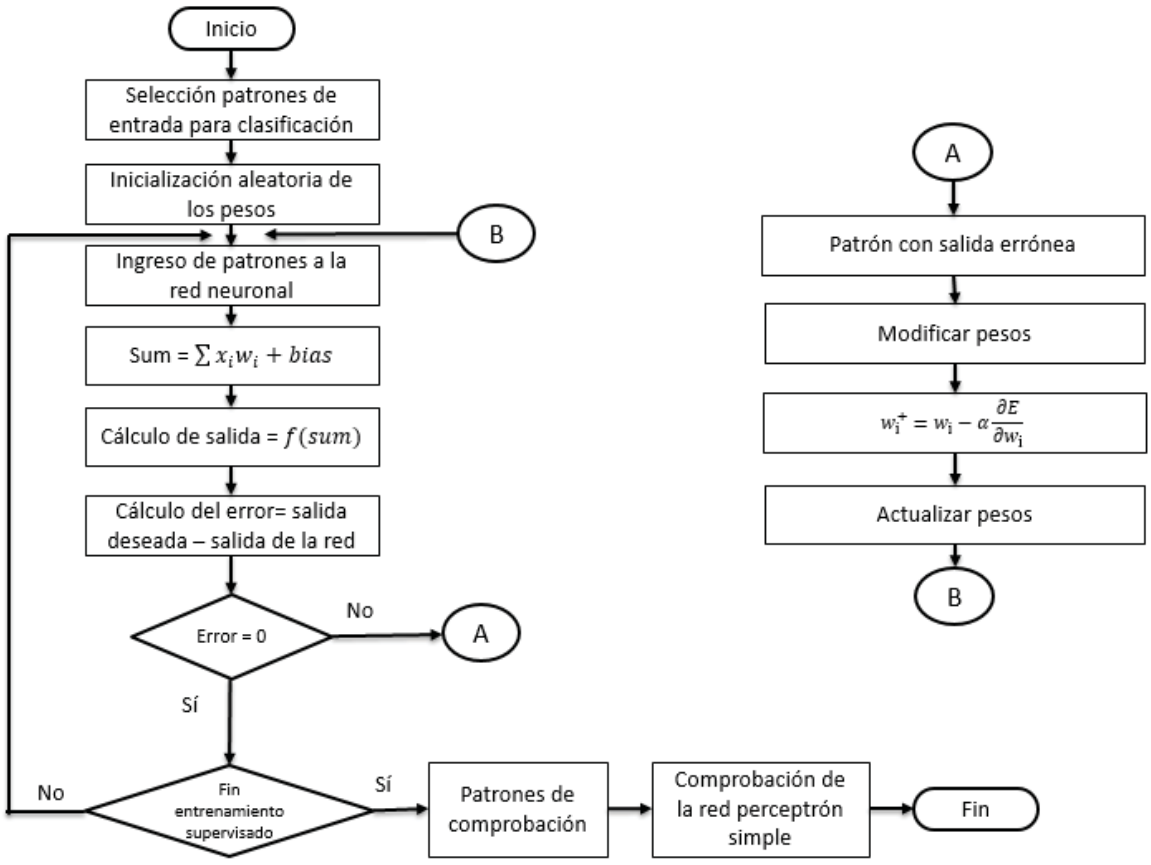

Figura 4-5 Diagrama de bloque del algoritmo

4.8 Ejemplo Algoritmo del Perceptrón multicapa

A continuación, se presenta un ejemplo para entender el algoritmo del perceptrón multicapa. Para este ejemplo, se consideran 2 entradas más el Bias en la capa de entrada, en la capa oculta se encuentran 2 neuronas más el Bias y en la capa de salida 1 neurona. Se desea que para entradas con nivel bajo (cercano a 0) la red muestre en su salida un nivel bajo (cercano a 0), los valores de las entradas y la salida respectivamente son: $x_1 = 0.01;\ x_2 = 0.1; y = 0.01$. Los valores de los pesos son los siguientes: $w_1 = 0.1;\ w_2 = 0.2;\ w_3 = 0.3;\ w_4 = 0.4;\ w_5 = 0.5;\ w_6 = 0.6; b_1 = 0.35;\ b_2 = 0.4$. Para la tasa de aprendizaje se escoge un valor = 1 de modo que éste facilite los cálculos.

4.8.1 Propagación hacia adelante

Se calculan los valores para la primera y segunda neurona oculta, la neurona de salida y el error obtenido por la red. En la Figura 4-6 se puede apreciar la red perceptrón multicapa a utilizar.

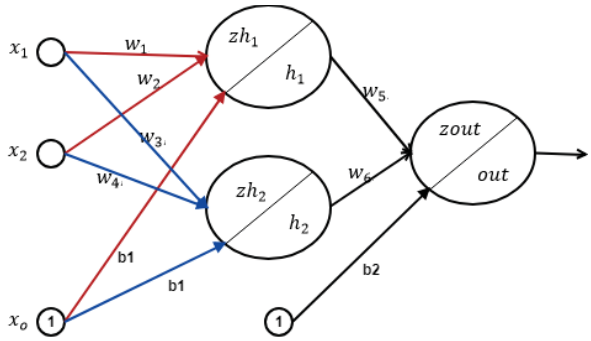

Figura 4-6 Perceptrón Multicapa

1. Primera neurona oculta

$$zh_1 = x_1 \cdot w_1 + x_2 \cdot w_2 + b_1 \cdot 1 = 0.01 \cdot 0.1 + 0.1 \cdot 0.2 + 0.35 \cdot 1 = 0.371$$

$$h_1 = \frac{1}{1 + e^{-zh_1}} = \frac{1}{1 + e^{-0.371}} = 0.5917$$

2. Segunda neurona oculta

$$zh_2 = x_1 \cdot w_3 + x_2 \cdot w_4 + b_2 \cdot 1 = 0.01 \cdot 0.3 + 0.1 \cdot 0.4 + 0.4 \cdot 1 = 0.443$$

$$h_2 = \frac{1}{1 + e^{-zh_2}} = \frac{1}{1 + e^{-0.443}} = 0.6089$$

3. Neurona de salida

$$zout = h_1 \cdot w_5 + h_2 \cdot w_6 + b_2 \cdot 1 = 0.5917 \cdot 0.5 + 0.6089 \cdot 0.6 + 0.4 \cdot 1 = 1.06119$$

$$out = \frac{1}{1 + e^{-zo}} = \frac{1}{1 + e^{-1.06119}} = 0.74291$$

4. Error cuadrático medio

$$E = \frac{(sd - out)^2}{2} = \frac{(0.01 - 0.74291)^2}{2} = 0.26857$$

Al observar el error, se concluye que la red no está bien adaptada, y se deben modificar los pesos de las conexiones mediante el algoritmo de retropropagación.

4.8.2 Propagación hacia atrás

1. **Cálculo de $\partial E / \partial w_5$**

En la Figura 4-7 se puede apreciar el movimiento del error desde la capa de salida hasta la capa oculta, llegando hasta w_5.

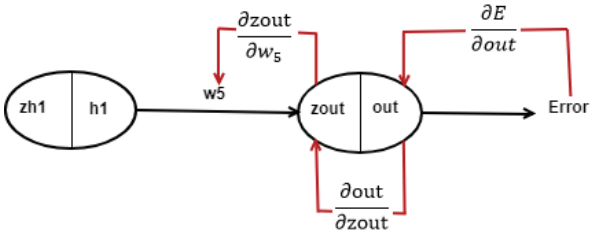

Figura 4-7 Cálculo de ∂E/∂w$_5$

Se debe calcular el error con respecto al peso w_5 como se ve a continuación:

$$\frac{\partial E}{\partial w_5} = \frac{\partial E}{\partial out} \cdot \frac{\partial \text{out}}{\partial \text{zout}} \cdot \frac{\partial \text{zout}}{\partial w_5} = (out - sd) \cdot \big(\text{out} \cdot (1 - \text{out})\big) \cdot (zh_1)$$

$$\frac{\partial E}{\partial w_5} = (0.74291 - 0.01) \cdot \big(0.74291 \cdot (1 - 0.74291)\big) \cdot (0.371) = 0.05193$$

2. Cálculo de $\partial E/\partial w_6$

En la Figura 4-8 se puede apreciar el movimiento del error desde la capa de salida hasta la capa oculta, llegando hasta el peso w_6.

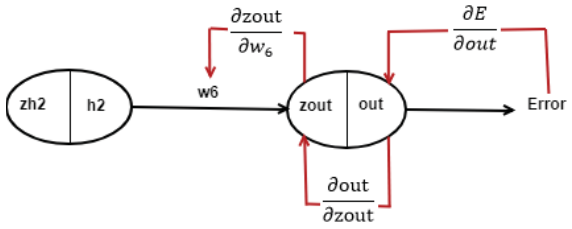

Figura 4-8 Cálculo de ∂E/∂w$_6$

Se calcula el error con respecto a w_6

$$\frac{\partial E}{\partial w_6} = \frac{\partial E}{\partial out} \cdot \frac{\partial \text{out}}{\partial \text{zout}} \cdot \frac{\partial \text{zout}}{\partial w_6} = (out - sd) \cdot \big(\text{out} \cdot (1 - \text{out})\big) \cdot (zh_2)$$

$$\frac{\partial E}{\partial w_6} = (0.74291 - 0.01) \cdot \big(0.74291 \cdot (1 - 0.74291)\big) \cdot (0.443) = 0.06201$$

3. Cálculo de $\partial E/\partial w_1$

En la Figura 4-9 se puede apreciar el movimiento del error desde la salida hasta la capa de entrada, llegando hasta w_1.

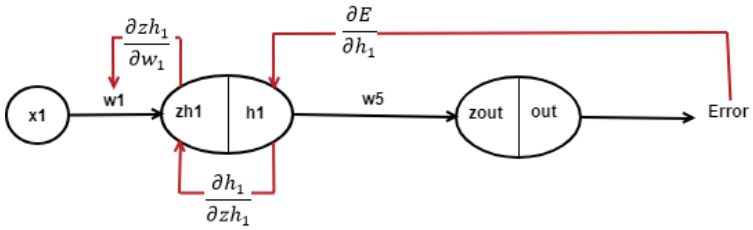

Figura 4-9 Cálculo de $\partial E/\partial w_1$

Se calcula el error con respecto a w_1

$$\frac{\partial E}{\partial w_1} = \frac{\partial E}{\partial h_1}\frac{\partial h_1}{\partial zh_1}\frac{\partial zh_1}{\partial w_1} = \left[(out - sd)\cdot\left(out\cdot(1-out)\right)\cdot(w_5)\right]\cdot\left(h_1(1-h_1)\right)\cdot(x_1)$$

$$\frac{\partial E}{\partial w_1} = \left[(0.74291 - 0.01)\cdot\left(0.74291\cdot(1-0.74291)\right)\cdot(0.5)\right]\cdot\left(0.5917\cdot(1-0.5917)\right)\cdot(0.01)$$

$$\frac{\partial E}{\partial w_1} = 0.0001691$$

4. Cálculo de $\partial E/\partial w_2$

En la Figura 4-10 se puede apreciar el movimiento del error desde la salida hasta la capa de entrada, llegando hasta w_2.

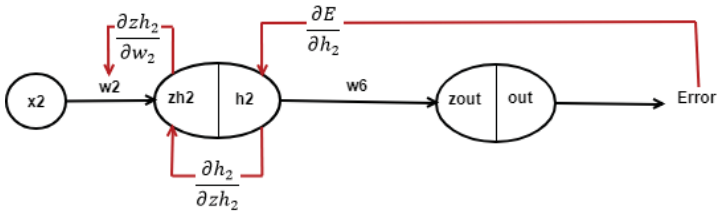

Figura 4-10 Cálculo de $\partial E/\partial w_2$

Se calcula el error con respecto a w_2

$$\frac{\partial E}{\partial w_2} = \frac{\partial E}{\partial h_2}\frac{\partial h_2}{\partial zh_2}\frac{\partial zh_2}{\partial w_2} = \left[(out - sd)\cdot\left(out\cdot(1-out)\right)\cdot(w_6)\right]\cdot\left(h_2(1-h_2)\right)\cdot(x_2)$$

$$\frac{\partial E}{\partial w_2} = \left[(0.74291 - 0.01)\cdot\left(0.74291\cdot(1-0.74291)\right)\cdot(0.6)\right]\cdot\left(0.6089\cdot(1-0.6089)\right)\cdot(0.1)$$

$$\frac{\partial E}{\partial w_2} = 0.00200012$$

5. Cálculo de $\partial E/\partial w_3$

En la Figura 4-11 se puede apreciar el movimiento del error desde la salida hasta la capa de entrada, llegando hasta w_3.

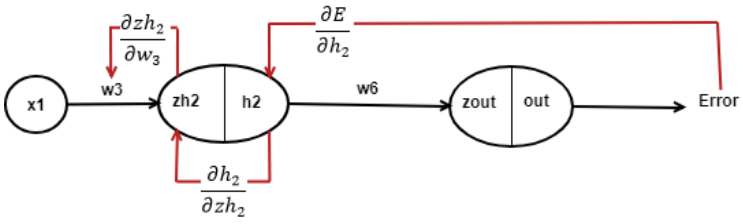

Figura 4-11 Cálculo de ∂E/∂w$_3$

$$\frac{\partial E}{\partial w_3} = \frac{\partial E}{\partial h_2}\frac{\partial h_2}{\partial zh_2}\frac{\partial zh_2}{\partial w_3} = \left[(out - sd)\cdot\left(out\cdot(1-out)\right)\cdot(w_6)\right]\cdot\left(h_2(1-h_2)\right)\cdot(x_1)$$

$$\frac{\partial E}{\partial w_3} = \left[(0.74291 - 0.01)\cdot\left(0.74291\cdot(1-0.74291)\right)\cdot(0.6)\right]\cdot\left(0.6089\cdot(1-0.6089)\right)\cdot(0.01)$$

$$\frac{\partial E}{\partial w_3} = 0.00020001$$

6. Cálculo de $\partial E/\partial w_4$

En la Figura 4-12 se puede apreciar el movimiento del error desde la salida hasta la capa de entrada, llegando hasta w_6.

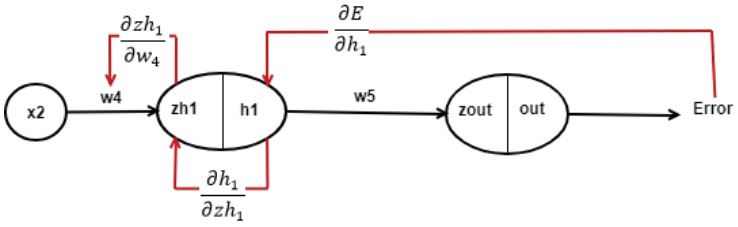

Figura 4-12 Cálculo de ∂E/∂w$_4$

$$\frac{\partial E}{\partial w_4} = \frac{\partial E}{\partial h_1}\frac{\partial h_1}{\partial zh_1}\frac{\partial zh_1}{\partial w_4} = \left[(out - sd)\cdot\left(out\cdot(1-out)\right)\cdot(w_5)\right]\cdot\left(h_1(1-h_1)\right)\cdot(x_2)$$

$$\frac{\partial E}{\partial w_2} = \left[(0.74291 - 0.01)\cdot\left(0.74291\cdot(1-0.74291)\right)\cdot(0.5)\right]\cdot\left(0.5917\cdot(1-0.5917)\right)\cdot(0.1)$$

$$\frac{\partial E}{\partial w_4} = 0.00169092$$

4.8.3 Actualización de los pesos

De la ecuación (4-11) se obtienen los nuevos valores de los pesos para las conexiones de la red.

$$w_1^+ = w_1 - \alpha\frac{\partial E}{\partial w_1} = 0.1 - 1\cdot 0.0001691 = 0.0998309$$

$$w_2^+ = w_2 - \alpha\frac{\partial E}{\partial w_2} = 0.2 - 1\cdot 0.00200012 = 0.19799988$$

$$w_3^+ = w_3 - \alpha \frac{\partial E}{\partial w_3} = 0.3 - 1 \cdot 0.00020001 = 0.29979999$$

$$w_4^+ = w_4 - \alpha \frac{\partial E}{\partial w_4} = 0.4 - 1 \cdot 0.00169092 = 0.39830908$$

$$w_5^+ = w_5 - \alpha \frac{\partial E}{\partial w_5} = 0.5 - 1 \cdot 0.05193 = 0.44807$$

$$w_6^+ = w_6 - \alpha \frac{\partial E}{\partial w_6} = 0.6 - 1 \cdot 0.06201 = 0.537988$$

Luego de actualizar los pesos, el algoritmo hace nuevamente una propagación hacia adelante para comprobar que el error disminuyó.

1. Primera neurona oculta

$$zh_1 = x_1 \cdot w_1 + x_2 \cdot w_2 + b_1 \cdot 1 = 0.01 \cdot 0.0998 + 0.1 \cdot 0.19799 + 0.35 \cdot 1 = 0.381$$

$$h_1 = \frac{1}{1 + e^{-zh_1}} = \frac{1}{1 + e^{-0.381}} = 0.5941114$$

2. Segunda neurona oculta

$$zh_2 = x_1 \cdot w_3 + x_2 \cdot w_4 + b_2 \cdot 1 = 0.01 \cdot 0.29979999 + 0.1 \cdot 0.39830909 + 0.35 \cdot 1 = 0.392$$

$$h_2 = \frac{1}{1 + e^{-zh_2}} = \frac{1}{1 + e^{-0.392}} = 0.596764$$

3. Neurona de salida

$$zout = h_1 \cdot w_5 + h_2 \cdot w_6 + b_2 \cdot 1 = 0.5941114 \cdot 0.44807 + 0.596764 \cdot 0.537988 + 0.4 \cdot 1 = 1.055116$$

$$out = \frac{1}{1 + e^{-zo}} = \frac{1}{1 + e^{-1.055116}} = 0.741756$$

4. Error cuadrático medio

$$E = \frac{(sd - out)^2}{2} = \frac{(0.01 - 0.741756)^2}{2} = 0.25365404$$

En conclusión, con la primera iteración se logra disminuir el error cometido por la red y la salida de la red comienza a acercarse al valor deseado, estos cambios son pequeños, sin embargo, se deben considerar varias iteraciones para lograr cambios más significativos y obtener una disminución considerable en el error cometido. En la Tabla 4-1 se puede apreciar el proceso de iteración de la red, que para llegar al valor deseado (0.01), se deben realizar aproximadamente 3000 iteraciones. (programación ejemplo Anexo C)

Tabla 4-1 Proceso de Iteración perceptrón multicapa

Iteración	Salida de la red	Error cometido por la red
1	0.741756	0.26773343
2	0.722256	0.25365404
3	0.701478	0.23907106
4	0.679544	0.22414448
50	0.173816	0.13829373
75	0.131905	0.00757371
80	0.126455	0.00690252
100	0.110363	0.00503636
1000	0.03165	0.000234
2000	0.023165	0.0000866
3000	0.019582	0.0000459

En la Figura 4-13 se puede apreciar la curva obtenida al graficar el error vs iteraciones, esta curva es la característica de aprendizaje de un perceptrón multicapa. En el gráfico se puede observar que el error disminuye considerablemente hasta la iteración 50, luego este proceso se vuelve más lento y el algoritmo se demora más en obtener los valores de los pesos para seguir reduciendo el error. Esto es debido a que, cuando el algoritmo realiza la retropropagación, los cambios que se generan en las derivadas con respecto a los pesos son cada vez más pequeños, y por lo tanto, esto afecta en la disminución del error.

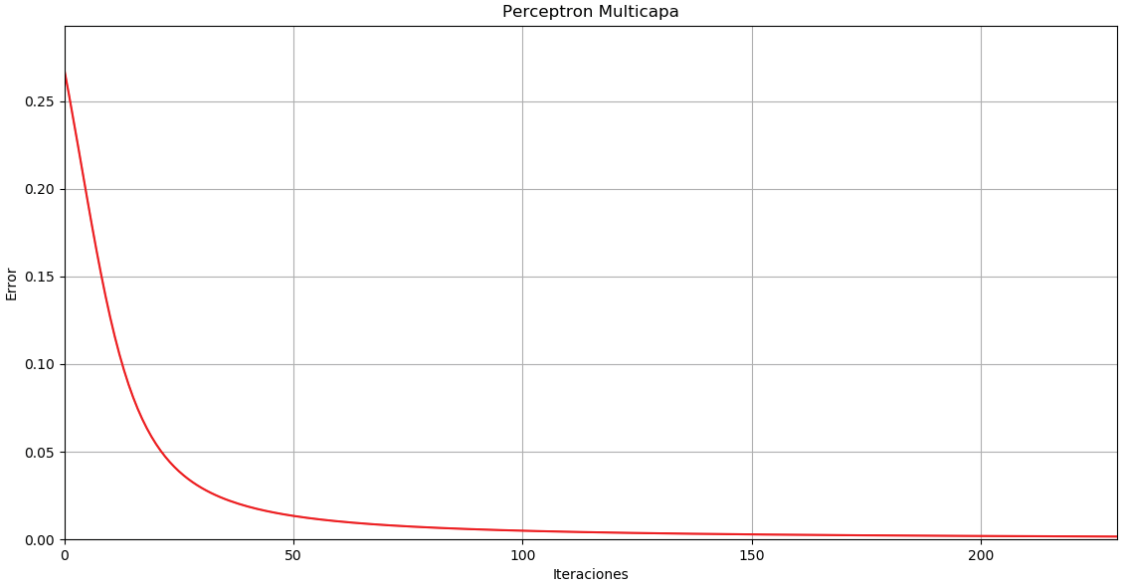

Figura 4-13 Gráfico Error vs Iteraciones

5 Diseños de Redes del tipo Perceptrón

En esta sección se detalla la estrategia utilizada para realizar el diseño del perceptrón simple y el perceptrón multicapa, se muestra el cómo trabajar con los patrones de entrada, la representación de la salida, y la selección de los patrones para ser utilizados en la fase de entrenamiento y en la fase de comprobación.

5.1 Diseño de perceptrón Simple

Primeramente, se establece la red neuronal artificial de tipo Perceptrón simple para desarrollar la clasificación de entradas linealmente separables como son; las compuertas lógicas OR y AND. Éstas compuertas estás definidas con dos entradas y una salida, por lo tanto, el diseño va a estar determinado conforme a esas especificaciones. Se considera el Bias en la capa de entrada para tener una convergencia más rápida en la red.

5.1.1 Arquitectura Perceptrón Simple

Para la red perceptrón simple, sólo basta tener una neurona de salida que determine la clasificación para los patrones de entrada. En la Tabla 5-1 se detallan la cantidad de neuronas utilizadas por capa.

Tabla 5-1: Cantidad de neuronas por capa

	Capa de entrada	Capa de salida
Número de neuronas	2	1
Bias	1	0

5.2 Diseño Perceptrón Multicapa

Para el diseño de una red neuronal artificial multicapa se debe considerar: el tipo de problema que se está abordando; la estructura de los datos de entrada; y cuántos datos son los que se quieren clasificar o reconocer. En los siguientes puntos se van a tomar en cuenta todas las consideraciones antes mencionadas, antes de crear la red perceptrón multicapa.

5.2.1 Perceptrón multicapa Como Reconocedor de Patrones

El reconocimiento de patrones trata de identificar características únicas que identifican un objeto de los demás de la misma especie. El sistema de reconocimiento de patrones debe asignar a cada objeto su clase o categoría y toma como base los siguientes procesos: adquisición, extracción de las características y toma de decisiones.

Para trabajar con una red neuronal, primeramente, se debe establecer un número de neuronas para la capa de entrada debido a que ésta va a definir la cantidad de datos que ingresarán a la red. La cantidad de neuronas en la capa de salida va a definir en cuántos ejemplos se deben clasificar los datos de entrada. Por lo tanto, se puede definir la red neuronal como el sistema que se ve en la Figura 5-1.

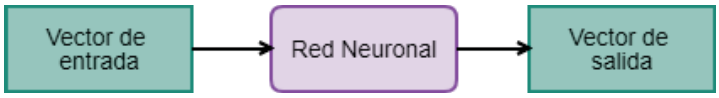

Figura 5-1: Sistema de Reconocimiento

5.2.2 Representación de los patrones de entrada

Para la representación de los patrones de entrada, se utiliza una matriz de 6 columnas por 7 filas en la cual cada celda ocupa un valor que puede ser 1 ó 0 dependiendo del patrón que se requiera representar [8], donde el valor (1) va a representar el trazo de la imagen y el valor cero (0) el caso contrario, como se puede apreciar en la Figura 5-2, que representa el número 2. Para determinar cada uno de los patrones de entrada es necesario considerar ejemplos que sean significativos y representativos.

0	1	1	1	1	0
1	0	0	0	0	1
0	0	0	0	1	0
0	0	0	1	0	0
0	0	1	0	0	0
0	1	0	0	0	0
1	1	1	1	1	1

Figura 5-2: Representación de patrones

Los datos de entrada por comodidad son trabajados como un vector lineal, como se puede apreciar en la Figura 5-3, por lo tanto, es conveniente transformar la matriz en un vector lineal de tamaño 42, donde cada una de las celdas que representan la matriz son redimensionadas a un vector de 1 fila por 42 columnas.

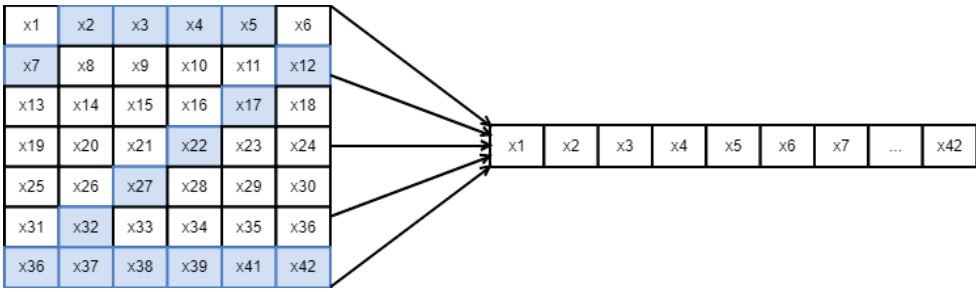

Figura 5-3: Transformación en un vector lineal

Finalmente, el vector de entrada queda expresado matemáticamente como se ve en la ecuación (5-1) y se puede ver gráficamente en la Figura 5-4.

$$X = [x_1, x_2, x_3, x_4, \dots, x_{42}] \tag{5-1}$$

x1	x2	x3	x4	x5	x6	...	x42
0	1	1	1	1	0		1

Figura 5-4: Vector de entrada

5.2.3 Representación de la salida

Por cada patrón de entrada que ingrese a la red, es necesario conocer su salida, y de esta forma poder hacer la discriminación o clasificación de los datos de entrada. En este caso, se eligieron los números del 0 al 9 para hacer la clasificación y entrenamiento, de esta forma se elige la cantidad de neuronas que deben conformar la capa de salida, para este caso la capa de salida de la red va a contar con 10 neuronas debido a que son diez los números a clasificar, y sólo una de estas neuronas se activará cuando reconozca al número que corresponde.

El vector de salida queda determinado como se puede ver en la ecuación (5-2)

$$Y = [y_1, y_2, y_3, \dots, y_{10}] \tag{5-2}$$

En la figura 5-5 se puede apreciar que, al ingresar un patrón de entrada, la salida de la red va a hacer la distinción de acuerdo a cómo se prepare la salida, en este caso, la segunda neurona de la capa de salida queda programada para que se active (1) cuando reconozca un número 2, mientras que las otras neuronas de salida permanezcas inactivas (0).

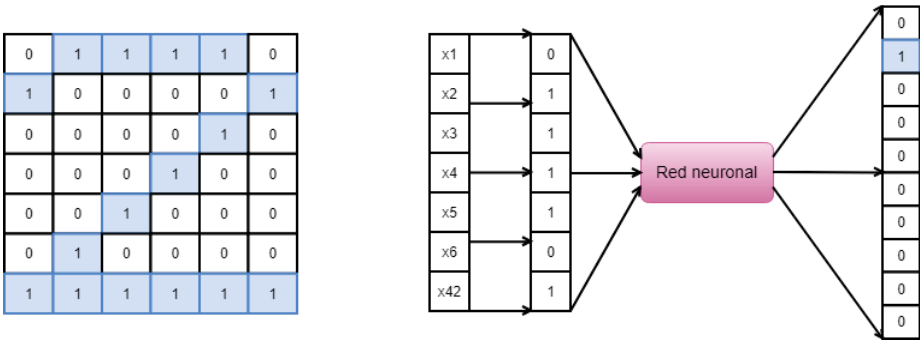

Figura 5-5: Representación de la salida

Finalmente, la segunda neurona de la capa de salida queda programada para reconocer los patrones de entrada que representen el número 2, mientras que las 9 neuronas restantes queden inactivas. De esta forma se realiza el entrenamiento de la red, al conocer la salida de cada uno de los patrones de entrada, esta irá modificando los pesos sus conexiones de modo que la salida de la red sea lo más parecida a la salida que se desea. Para todos los ejemplos que componen el número 2, la salida de la red queda programada para que la segunda neurona se active mientras que las otras neuronas permanezcan inactivas.

5.2.4 Arquitectura Perceptrón Multicapa

En la Tabla 5-2 se describe la cantidad de neuronas ocupadas por cada capa de la red.

Tabla 5-2 Cantidad de neuronas por capa

	Capa de entrada	Capa oculta	Capa de salida
Número de Neuronas	42	20	10
Bias	1	1	0

El número de neuronas empleado para las capas de entrada y salida va a depender exclusivamente de la aplicación en particular, en este caso, se utilizan 42 neuronas en la entrada debido a que los patrones que ingresan a la red están compuestos por una matriz de 6 columnas por 7 filas, que luego son redimensionados en un vector lineal de tamaño 42. Para la capa de salida se requieren 10 neuronas, debido a que se necesita clasificar los números del 0 al 9. Para la capa oculta, no existe un método que determine de manera óptima la cantidad de neuronas que se necesitan con precisión en dicha capa, sólo existen métodos de estimación para determinar la cantidad de neuronas que pueden ser utilizadas. Para este caso, se utilizó la siguiente estimación:

$$h = \sqrt{m \cdot n} \tag{5-3}$$

Donde:

h = Neuronas capa oculta

m = Neuronas capa de entrada

n = Neuronas capa de salida

5.3 Patrones de Entrenamiento y Comprobación

El aprendizaje de la red es el proceso mediante el cual ésta modifica sus pesos en respuesta a los patrones de entrada presentados. El conjunto de patrones de entrada debe ser significativo y representativo, es decir, debe haber un número suficiente de ejemplos debido a que si existe un número reducido de patrones de entrenamiento la red no será capaz de adaptar sus pesos de forma eficaz, y representativo quiere decir, que los ejemplos que componen el conjunto de entrenamiento deben ser diversos, de tal modo que la red pueda aprender de distintos tipos y no se especialice en un subconjunto de datos.

5.3.1 Selección de Patrones

Se entrena la red para que esta pueda reconocer los números del 0 al 9, para llevar a cabo esto se utilizan 20 patrones de entrenamiento, 2 ejemplos distintos por cada número escogido. Sobre estos ejemplos se espera que la red aprenda y luego pueda hacer la discriminación a base de lo aprendido, el aprendizaje consiste en ir modificando los pesos de sus conexiones mediante el algoritmo de retropropagación, moviendo el error desde la salida e ir pasando por cada neurona de cada una de las capas que componen la red. En la Figura 5-6 se pueden observar los patrones de entrada seleccionados para la fase de entrenamiento.

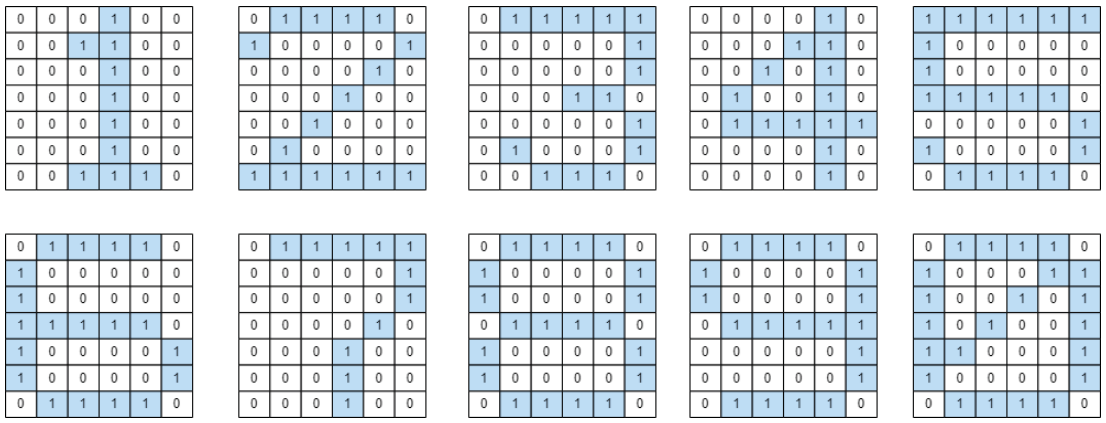

Figura 5-6: Patrones de entrenamiento

Estos 20 patrones de entrada componen la base para el entrenamiento de la red, mientras más ejemplos representativos y significativos es mejor para el perceptrón multicapa, porque de esta forma va a hacer una clasificación más eficaz, sin especializarse en un subconjunto de entrenamiento si es que estos fuesen muy parecidos entre sí.

5.3.2 Activación de las Neuronas en la capa de salida

Se debe asignar un valor de activación o inactivación a cada neurona de salida para saber cuál de éstas va a realizar el proceso de reconocimiento, en este caso, se asigna como valor activo por neurona el valor 0.95 y la inactividad está representada por el valor 0.05. En la Tabla 5-3 se muestran los valores que debe tener cada neurona de salida para el patrón de entrada correspondiente. Esto quiere decir, que la neurona de la capa de salida N1 se va a activar con un valor de 0.95 cuando ingrese un patrón que representa el número 1 y el resto va a permanecer inactivas con un valor de 0.05, y así respectivamente con las demás neuronas de salida.

Tabla 5-3: Valores de salida por neurona

	Neuronas en la capa de salida									
Patrón de entrada	N 1	N 2	N 3	N 4	N 5	N 6	N 7	N 8	N 9	N 10
Número 1	0.95	0.05	0.05	0.05	0.05	0.05	0.05	0.05	0.05	0.05
Número 2	0.05	0.95	0.05	0.05	0.05	0.05	0.05	0.05	0.05	0.05
Número 3	0.05	0.05	0.95	0.05	0.05	0.05	0.05	0.05	0.05	0.05
Número 4	0.05	0.05	0.05	0.95	0.05	0.05	0.05	0.05	0.05	0.05
Número 5	0.05	0.05	0.05	0.05	0.95	0.05	0.05	0.05	0.05	0.05
Número 6	0.05	0.05	0.05	0.05	0.05	0.95	0.05	0.05	0.05	0.05
Número 7	0.05	0.05	0.05	0.05	0.05	0.05	0.95	0.05	0.05	0.05
Número 8	0.05	0.05	0.05	0.05	0.05	0.05	0.05	0.95	0.05	0.05
Número 9	0.05	0.05	0.05	0.05	0.05	0.05	0.05	0.05	0.95	0.05
Número 0	0.05	0.05	0.05	0.05	0.05	0.05	0.05	0.05	0.05	0.95

5.3.3 Patrones de Comprobación

Después que termine el proceso de entrenamiento, es necesario verificar el funcionamiento de la red, para esto se deben ingresar patrones que no se hayan visto en la fase de entrenamiento. Se escogen 10 patrones para la comprobación de la red como se puede apreciar en la Figura 5-7.

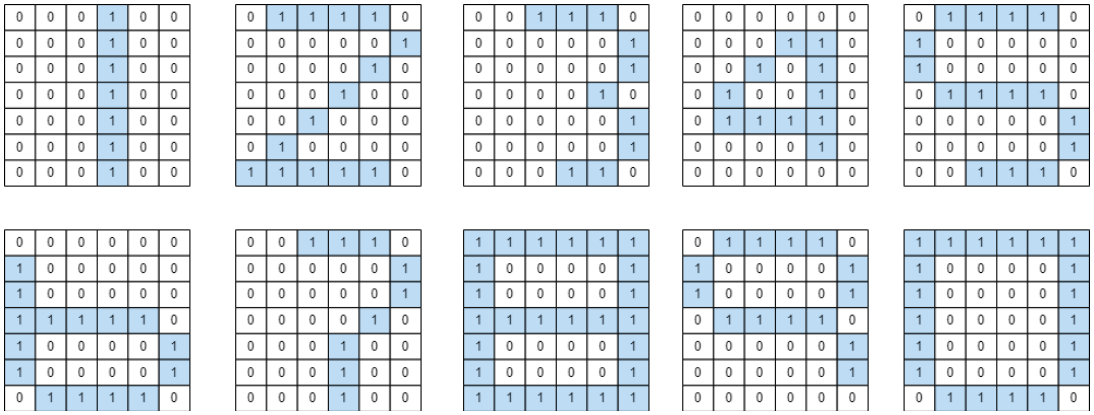

Figura 5-7: Patrones de comprobación

5.3.4 Patrones Desconocidos por la Red

Se desea que el perceptrón multicapa de este proyecto funcione reconociendo número del cero al nueve, pero se quiere verificar ¿Qué pasaría si se ingresan datos que no corresponden a los vistos en la fase de entrenamiento?, para esto, se seleccionan tres nuevos patrones correspondientes a las letras A, B y C, como se puede apreciar en la Figura 5-8. Se espera a que el perceptrón multicapa encuentre similitudes entre los datos aprendidos y los patrones desconocidos, generando una respuesta en base a los patrones aprendidos en la fase de entrenamiento.

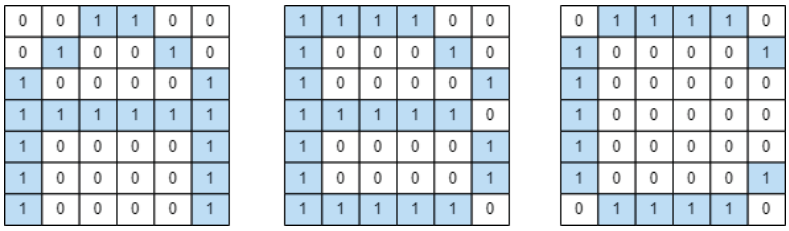

Figura 5-8 Patrones desconocidos por la red

6 Implementación Perceptrón Multicapa

A continuación, se detallan los pasos para lograr la implementación de la red neuronal en la plataforma Raspberry Pi, primeramente, dando a conocer las características del dispositivo a utilizar, el tipo de lenguaje de programación y cómo lograr la comunicación entre PC y Raspberry Pi.

6.1 Raspberry Pi

Raspberry Pi es un computador de placa reducida, o placa simple de bajo costo desarrollado en el Reino Unido por la fundación Raspberry Pi, cuyo objetivo principal es estimular el aprendizaje y enseñanzas de las ciencias de la computación en las escuelas [14]. En la Figura 6-1 se puede observar la Raspberry Pi desde una vista superior.

Figura 6-1: Raspberry Pi 2 Modelo B (Fuente: https://www.raspberrypi.org)

El modelo que se utiliza para el proyecto es la Raspberry pi 2 modelo B, es la segunda versión de la Raspberry Pi y este modelo fue lanzado en el año 2014, presenta una gran cantidad de mejoras con respecto a sus antecesores, fue el primer dispositivo en utilizar un procesador de distinto modelo, pero de misma marca, este fue el BCM2836, también, en vez de 1 núcleo utiliza 4 núcleos, y de 700 MHz a 900 MHz. La memoria RAM es duplicada y pasa de 512 MB a 1 GB, también posee 40 pines de propósito general (GPIO), y mantiene los cuatro puertos USB. No presenta salida RCA. Presenta una salida de video y audio a través de un conector HDMI. En la Figura 6-2 se pueden apreciar los puertos GPIO de la Raspberry Pi.

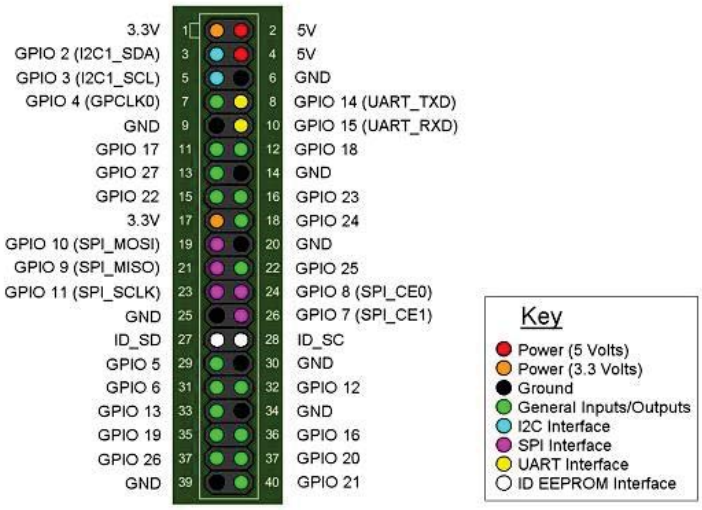

Figura 6-2: Puertos GPIO disponibles en Raspberry Pi 2 (Fuente: https://www.intelidomo.com)

6.2 Python

Python es un lenguaje de programación interpretado de alto nivel y multipropósito, esto último quiere decir que soporta orientación a objetos y programación funcional [15]. En la actualidad, es uno de los lenguajes más utilizados para el desarrollo de software al ser un lenguaje potente, con una sintaxis clara y concisa, flexible y de código abierto. Este lenguaje de programación puede ser utilizado en diversos sistemas operativos tales como: Windows, Mac Os X y Linux. Python no tiene un ámbito específico y puede ser empleado para desarrollo de software con carácter científico, comunicaciones de redes, creación de juegos y aplicaciones web. En la figura 6-3 se puede apreciar el logo que identifica a Python

Figura 6-3: Logo Python (Fuente: https://www.python.org/)

En la Figura 6-4 se muestran las publicaciones de las versiones de Python, considerando las tres principales (Python1, Python2 y Python3). Las versiones obsoletas están representadas en color rojo y las versiones en negro son las cuales siguen publicando actualizaciones [16].

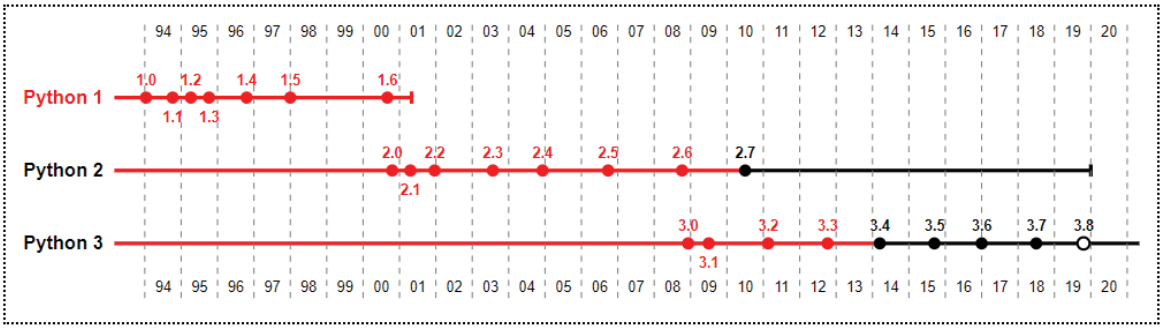

Figura 6-4: Versiones principales de Python (Fuente: http://www.mclibre.org)

6.3 Comunicación

Para establecer comunicación entre el computador y Raspberry Pi, es necesario conocer la dirección IP de ésta, si no, asignarle una dirección IP estática. Para el desarrollo del proyecto, se utiliza un cable ethernet para comunicarse con la Raspberry Pi desde el computador, asignando una dirección IP estática al computador (IP:192.168.1.200) y luego conectándose por SSH a la Raspberry Pi a la IP:192.168.1.10.

6.3.1 Configuración IP estática PC

Primero se configura una dirección IP estática para el PC, se asigna la IP:192.168.1.200 ingresando a propiedades de ethernet y utilizando TCP/IPv4, tal como se muestra en la Figura 6-5 y Figura 6-6.

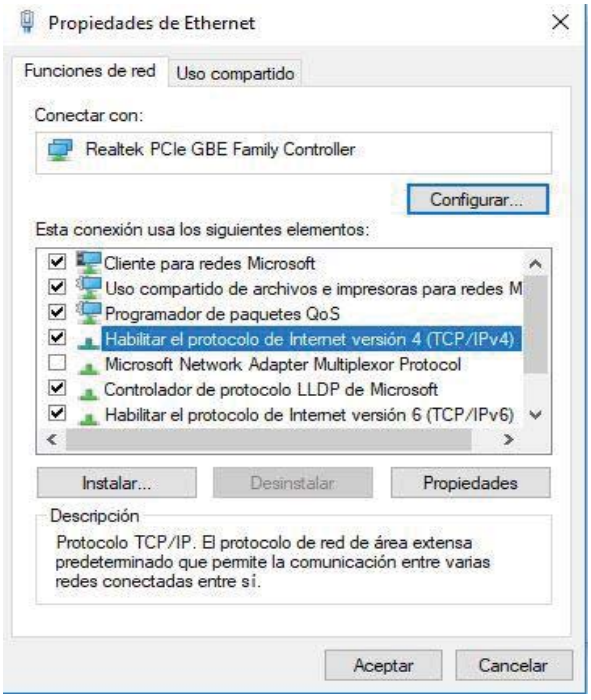

Figura 6-5: Configuración dirección IP PC

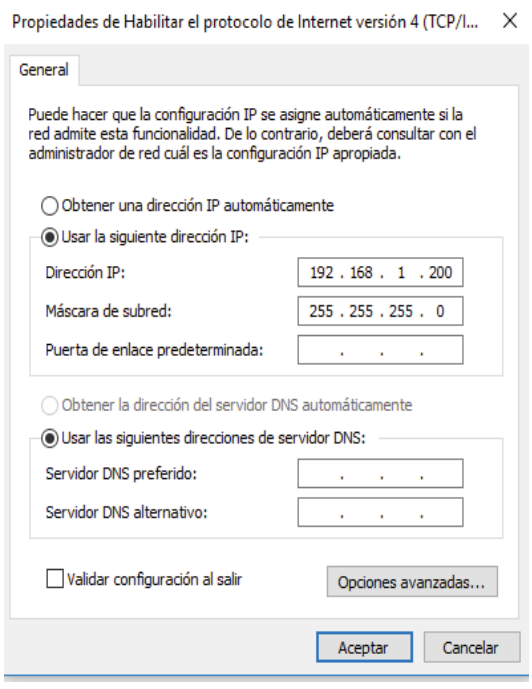

Figura 6-6: Dirección IP Estática

6.3.2 Configuración IP estática Raspberry Pi

Primeramente, se utiliza la Raspberry Pi con una pantalla externa para poder configurar la dirección IP y de esta forma trabajar desde el computador mediante SSH. En la pantalla de comandos de Raspberry Pi, se debe ingresar la siguiente línea de comandos para iniciar la configuración, como se muestra en el Listado 6-1.

Listado 6-1: Configuración de la dirección IP de la Raspberry Pi

```
1 Sudo nano /etc/network/interfaces
2
3
```

Al ingresar el comando, en pantalla se mostrará lo que aparece en la Figura 6-7 y se deben hacer las modificaciones tal como se muestran en dicha imagen. Se debe modificar la línea que dice "**iface eth0 inet dhcp**" por "**iface eth0 inet static**". Luego se deben incorporar las direcciones correspondientes a la dirección IP de la Raspberry Pi (IP:192.168.1.10), la máscara de subred (255.255.255.0) y la dirección IP estática del PC (192.168.1.200). Luego de ingresar los datos, se deben guardar presionando "Ctrl + X" luego apretar "Y" y para finalizar "Enter". Luego se debe reiniciar la Raspberry Pi utilizando la línea de código mostrada en el Listado 6-2.

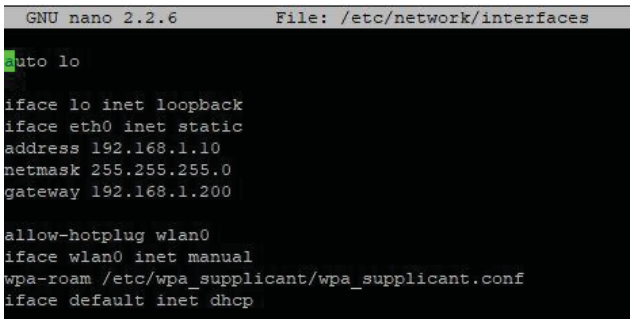

Figura 6-7: Configuración IP RPi

Listado 6-2: Reinicio de Raspberry Pi

```
1 Sudo reboot
2
3
```

6.3.3 Verificación de la Comunicación

Para corroborar que los dispositivos se encuentran comunicados, se puede hacer un ping desde el computador a la Raspberry Pi o desde la Raspberry Pi hacia el computador.

1. **Computador hacia RPi**
   Para verificar la comunicación desde el computador hacia la RPi, se debe ingresar a los comandos de Windows presionando “Windows + R” y escribiendo “cmd”, luego se debe escribir el comando que se indica en el Listado 6-3.

Listado 6-3: Ping desde PC a RPi

```
1 Ping 192.168.1.10
2
3
```

Si la conexión fue exitosa, se mostrará algo parecido a la Figura 6-8

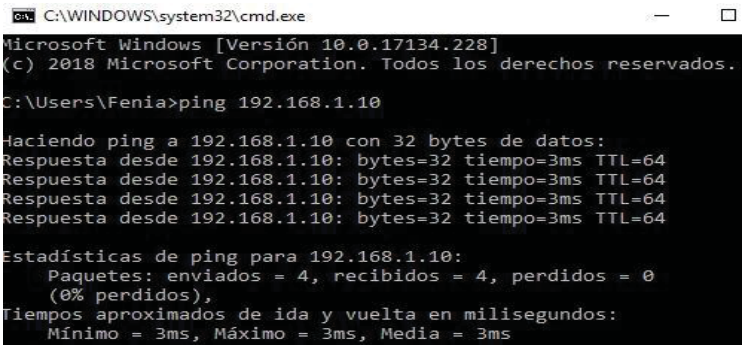

Figura 6-8: Ping hacia RPi

2. **RPi hacia computador**

   Para realizar un ping desde la Raspberry Pi, se debe ingresar el comando del Listado 6-4.

Listado 6-4: Ping RPi hacia PC

```
1 Ping 192.168.1.200
2
3
```

Si la conexión fue exitosa, se mostrará algo parecido a la Figura 6-9. Para cancelar se debe presionar “Ctrl +C”.

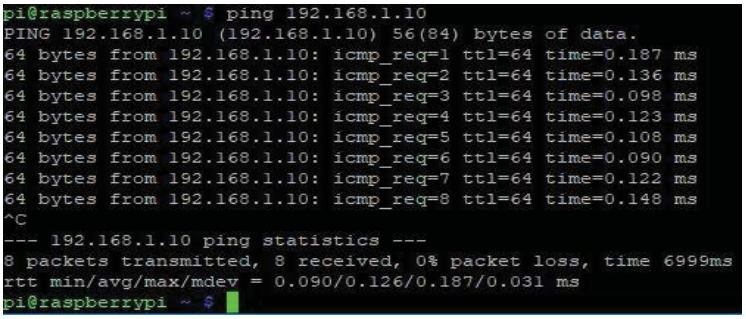

Figura 6-9: Ping hacia PC

7 Resultados Redes Perceptrón

En este capítulo se muestran los resultados obtenidos por la red perceptrón simple y multicapa respectivamente.

7.1 Resultados Red Neuronal Simple

La red perceptrón simple, se analiza con los datos de entrada de las funciones lógicas OR, AND y XOR, el análisis teórico fue desarrollado en el apartado del perceptrón simple, y se van a considerar los mismos valores de pesos y tasa de aprendizaje. El programa muestra los patrones de entrada, salida deseada y pesos iniciales de las conexiones, cómo también, los pesos actualizados y el error producido en cada iteración. Se analiza el comportamiento de la red para las tres compuertas mencionadas anteriormente, y se comprueba las falencias que tiene el perceptrón simple para casos en que las entradas no son linealmente separables.

7.1.1 Compuerta OR

La Tabla 7-1 muestra la función lógica de la compuerta OR. En todo caso, mientras exista un 1 en una de sus entradas, su salida presentará un nivel alto (1), en caso contrario, cuando las entradas son 0, la salida presenta un nivel bajo (0).

Tabla 7-1: Compuerta lógica OR

x_1	x_2	y
0	0	0
0	1	1
1	0	1
1	1	1

En la Figura 7-1 se muestra en pantalla los resultados obtenidos por el perceptrón simple al utilizar una función lógica OR. Los pesos iniciales son: $w_1 = 0.5; w_2 = 1;\ w_0 = -1.5$. el programa realiza 2 iteraciones para determinar los pesos óptimos y no obtener error en la salida. La variable z representa la sumatoria realizada en la neurona de salida para cada uno de los patrones de entrada. La variable *f()* representa a la variable z evaluada en la función de activación escalón. Luego es mostrado el error por cada patrón de entrada y finalmente, la actualización de los pesos,

utilizando el primer patrón que tuvo error para hacer la actualización, es decir, para la primera iteración el error se obtuvo en el patrón (0,1) y el (1,0), por lo tanto, el algoritmo va a escoger el patrón que tiene el primero el error, en este caso viene siendo el patrón (0,1) y con esos datos se realiza la actualización. En la segunda iteración con los nuevos pesos se obtuvo error cero para cada patrón de entrada, por lo tanto, no se hace actualización, y quedan los pesos ajustados.

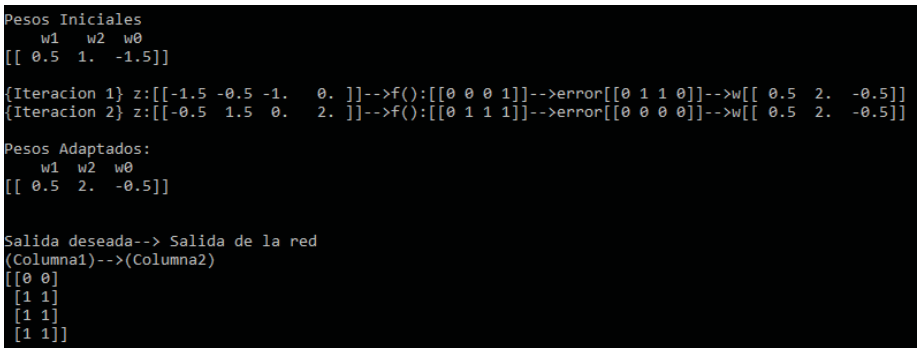

Figura 7-1 Resultados OR

En la Tabla 7-2 se puede ver la iteración realizada por el programa.

Tabla 7-2 Iteración OR

Iteración	x_1	x_2	$d(t)$	w_1	w_2	w_0	y	error	$w_1(t+1)$	$w_2(t+1)$	$w_0(t+1)$
1	0	0	0	0.5	1	-1.5	0	0	-	-	-
	0	1	1	-1.5	1	-1.5	0	1	0.5	2	-0.5
	1	0	1	-1.5	1	-1.5	0	1	-	-	-
	1	1	1	-1.5	1	-1.5	1	0	-	-	-
2	0	0	0	0.5	2	-0.5	1	0	-	-	-
	0	1	1	0.5	2	-0.5	1	0	-	-	-
	1	0	1	0.5	2	-0.5	1	0	-	-	-
	1	1	1	0.5	2	-0.5	0	0	-	-	-

7.1.2 Compuerta AND

La compuerta AND se caracteriza por los valores que se encuentran en la Tabla 7-3. Para todo caso, mientras exista un cero en cualquiera de sus entradas, su salida presenta un nivel bajo (0), y en caso que las entradas presenten un uno, la salida presenta un nivel alto (1).

Tabla 7-3: Compuerta lógica AND

x_1	x_2	y
0	0	0
0	1	0
1	0	0
1	1	1

La Figura 7-2 muestra el recorrido que hace el perceptrón para adaptar sus pesos (como se analizó en el capítulo 3.4.2), la variable "z" representa la sumatoria de los pesos multiplicado por cada entrada del perceptrón, el valor de la izquierda representa la entrada (0,0), el segundo representa (0,1) y así respectivamente. Luego, el valor obtenido en la variable "z" es ingresado a la función de activación escalón que está representada por la variable" f()". Después de evaluar en la función de activación, el programa encuentra el error que genera la red para cada patrón de entrada respectivamente, y finalmente, se encuentran los valores de los pesos actualizados "w".

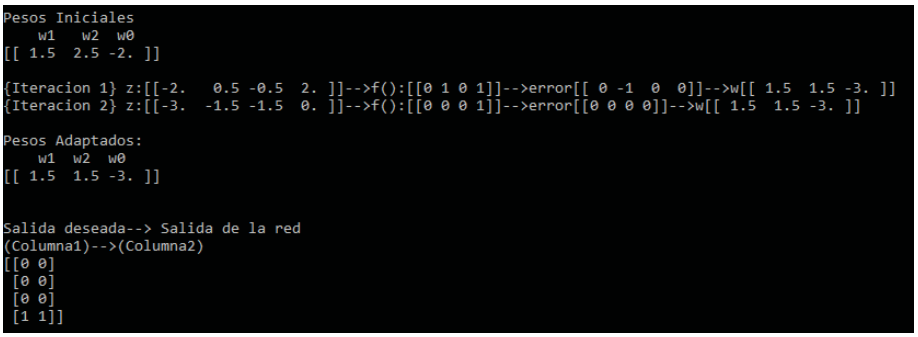

Figura 7-2: Resultados AND

En la Tabla 7-4 se puede ver la iteración realizada por el programa.

Tabla 7-4: Proceso Iteración AND

Iteración	x_1	x_2	$d(t)$	w_1	w_2	w_0	y	error	$w_1(t+1)$	$w_2(t+1)$	$w_0(t+1)$
1	0	0	0	1.5	2.5	-2	0	0	-	-	-
	0	1	0	1.5	2.5	-2	1	-1	1.5	1.5	-3
	1	0	0	1.5	2.5	-2	0	0	-	-	-
	1	1	1	1.5	2.5	-2	1	0	-	-	-
2	0	0	0	1.5	1.5	-3	0	0	-	-	-
	0	1	0	1.5	1.5	-3	0	0	-	-	-
	1	0	0	1.5	1.5	-3	0	0	-	-	-
	1	1	1	1.5	1.5	-3	1	0	-	-	-

7.1.3 Compuerta XOR

La compuerta XOR se caracteriza por los valores que se encuentran en la Tabla 7-5. Se espera que la red no logre convergencia y comience a oscilar.

Tabla 7-5: Compuerta lógica XOR

x_1	x_2	y
0	0	0
0	1	1
1	0	1
1	1	0

En la Figura 7-3 se pueden ver los resultados obtenidos del perceptrón ante una entrada que no es linealmente separable, en este caso, la compuerta XOR. La red no logra adaptar los pesos, y comienza a oscilar, es decir, empieza a repetir pesos que no llevan a la convergencia. En pantalla se puede ver que, a pesar de un aumento en las iteraciones, esta no logra adaptar sus pesos, y con esto queda demostrado los problemas que tiene el perceptrón simple para trabajar con datos que no son linealmente separables. En pantalla se logra ver un apartado que dice “pesos adaptados” estos simplemente representan a los pesos que obtuvo en la última iteración. En la sección del perceptrón simple, se analiza de manera más completa el problema con la compuerta XOR.

```
Pesos Iniciales
    w1   w2  w0
[[ 0.5  1.  -1.5]]

{Iteracion 1} z:[[-1.5 -0.5 -1.   0. ]]-->f():[[0 0 0 1]]-->error[[ 0  1  1 -1]]-->w[[ 0.5  2.  -0.5]]
{Iteracion 2} z:[[-0.5  1.5  0.   2. ]]-->f():[[0 1 1 1]]-->error[[ 0  0  0 -1]]-->w[[-0.5  1.  -1.5]]
{Iteracion 3} z:[[-1.5 -0.5 -2.  -1. ]]-->f():[[0 0 0 0]]-->error[[0 1 1 0]]-->w[[-0.5  2.  -0.5]]
{Iteracion 4} z:[[-0.5  1.5 -1.   1. ]]-->f():[[0 1 0 1]]-->error[[ 0  0  1 -1]]-->w[[0.5 2.  0.5]]
{Iteracion 5} z:[[0.5 2.5 1.  3. ]]-->f():[[1 1 1 1]]-->error[[-1  0  0 -1]]-->w[[ 0.5  2.  -0.5]]
{Iteracion 6} z:[[-0.5  1.5  0.   2. ]]-->f():[[0 1 1 1]]-->error[[ 0  0  0 -1]]-->w[[-0.5  1.  -1.5]]
{Iteracion 7} z:[[-1.5 -0.5 -2.  -1. ]]-->f():[[0 0 0 0]]-->error[[0 1 1 0]]-->w[[-0.5  2.  -0.5]]

Pesos Adaptados:
    w1  w2  w0
[[0.5 2.  0.5]]


Salida deseada--> Salida de la red
(Columna1)-->(Columna2)
[[0 0]
 [1 1]
 [1 0]
 [0 1]]
```

Figura 7-3: Resultados XOR

7.2 Resultados Red Perceptrón Multicapa

Primeramente, se entrena la red con 10 patrones, uno por cada clase y con esto se espera que el sistema adapte sus pesos mediante el algoritmo de retropropagación. En la Figura 7-4 se subraya con rojo el valor de la neurona activa. La primera fila de valores representa la respuesta de las neuronas de la capa de salida al ingresar el patrón del número 1, la segunda fila representa la respuesta de las neuronas de la capa de salida al ingresar el patrón del número 2 y así sucesivamente, donde la última fila representa al número 0. Finalmente, el proceso de entrenamiento de la red perceptrón multicapa se lleva a cabo de manera correcta, al comparar

con los valores deseados. Se puede observar que las neuronas de salida responden a los patrones de entrada como se espera en la Tabla 5-3.

```
salida de la red :
[[9.48e-01 4.91e-02 5.18e-02 4.60e-02 3.29e-05 1.02e-03 5.45e-02 5.35e-04 4.85e-02 5.09e-02]
 [5.13e-02 9.47e-01 4.97e-02 5.13e-02 5.09e-02 4.84e-02 4.06e-02 3.86e-06 5.06e-02 1.12e-03]
 [5.18e-02 5.39e-02 9.50e-01 5.14e-02 5.68e-02 7.43e-06 5.16e-02 5.23e-02 5.44e-02 1.75e-04]
 [5.02e-02 4.93e-02 5.00e-02 9.50e-01 4.91e-02 5.07e-02 5.12e-02 5.01e-02 5.02e-02 5.03e-02]
 [2.53e-04 4.68e-02 5.17e-02 4.54e-02 9.44e-01 5.27e-02 5.85e-02 5.12e-02 4.87e-02 1.21e-04]
 [3.28e-03 5.72e-02 1.50e-03 5.10e-02 5.95e-02 9.46e-01 3.74e-02 5.03e-02 5.85e-02 5.75e-02]
 [5.15e-02 5.95e-04 4.98e-02 5.10e-02 5.18e-02 4.69e-05 9.47e-01 5.07e-02 5.04e-02 5.35e-04]
 [5.40e-03 1.33e-03 5.34e-02 4.91e-02 4.74e-02 4.71e-02 5.16e-02 9.50e-01 4.74e-02 2.84e-02]
 [4.95e-02 5.12e-02 4.75e-02 5.20e-02 9.08e-03 5.12e-02 4.84e-02 4.40e-02 9.43e-01 3.33e-05]
 [5.41e-02 4.65e-02 4.60e-02 5.19e-02 4.31e-03 5.44e-02 4.78e-02 5.03e-02 3.33e-04 9.58e-01]]
```

Figura 7-4: Salida de la red

En la Figura 7-5 se puede observar gráficamente lo que sucede en la primera fila de la Figura 7-4. Al ingresar el patrón que contiene el número 1, la red modifica sus pesos para que las neuronas que se encuentran en la capa de salida respondan a este de acuerdo a lo diseñado, es decir, que se active la primera neurona de salida con un valor de 0.95 y las otras neuronas de salidas presenten inactividad con un valor de 0.05, en este caso, la red obtuvo un 0.948, mientras que las otras neuronas de salida permanecen inactivas con un valor de 0.05, la red obtuvo valores cercanos a 0.05 como también valores más pequeños, estos son aceptables debido a que muestran la inactividad de la neurona en cuestión.

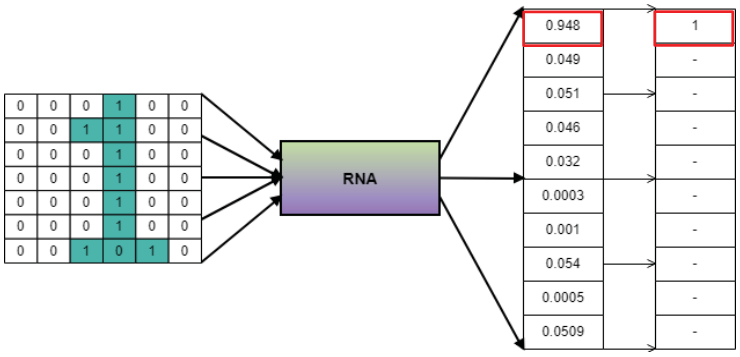

Figura 7-5 Reconocimiento patrón 1

Por lo tanto, el perceptrón multicapa queda con sus pesos fijos después de la fase de entrenamiento. En la Figura 7-6 se puede ver el gráfico del error vs iteraciones realizado por el perceptrón multicapa después de entrenarlo con 20 ejemplos, dos de cada tipo. Aproximadamente, después de 200 iteraciones, éste logra disminuir el error de manera considerable, pero para llegar a los valores deseados se requiere seguir iterando, este proceso es lento, debido a que mientras aumenten las iteraciones, las derivadas del error con respecto a los pesos que calcula el algoritmo, se hacen cada vez más pequeñas, por esto es que se demora más

en llegar a los valores óptimos después de las 200 iteraciones. El programa considera una tasa de aprendizaje de 0.5 y 10000 iteraciones.

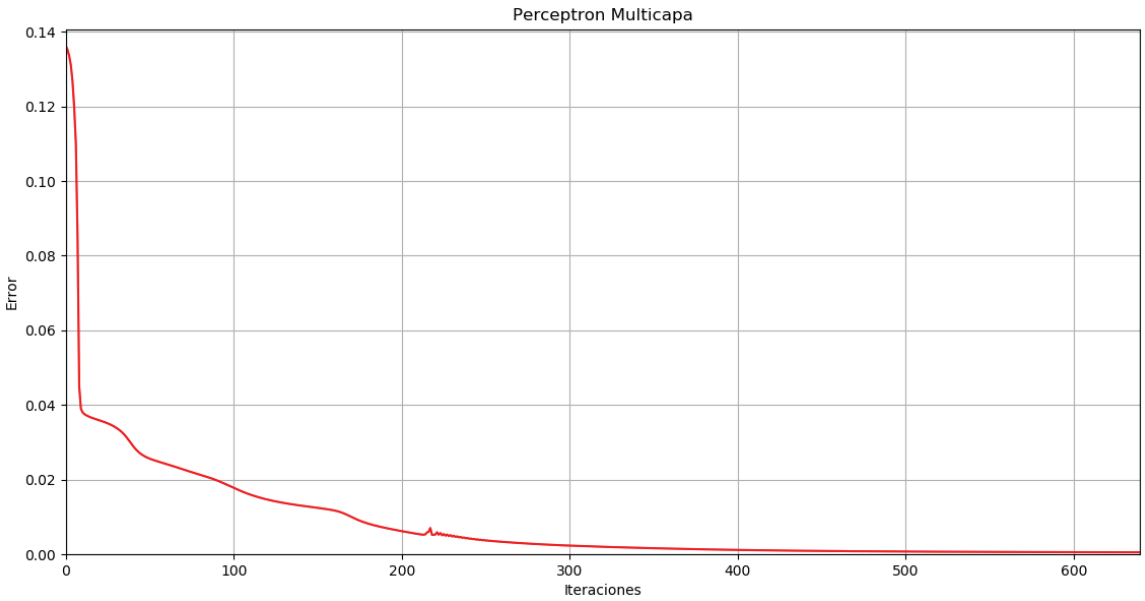

Figura 7-6: Gráfica Error v/s Iteraciones

7.2.1 Comprobación de la red

En la Figura 7-7 se puede apreciar la salida de la red después de la fase de comprobación, las neuronas activas están subrayadas con azul para lograr un reconocimiento visual más rápido, estas neuronas activas corresponden al patrón que representa el número, es decir, para la primera fila, la primera neurona de salida se activa al reconocer el patrón que contiene el número 1 y ésta obtiene un valor cercano a 0.95; para la segunda fila, la segunda neurona se activa cuando ésta reconoce el patrón que contiene al número 2 y ésta obtiene un valor cercano a 0.95 y así sucesivamente con las demás filas. Una vez ingresados todos los patrones para la comprobación de la red, se obtienen valores de activación cercanos a 0.95 y para representar la inactividad se obtienen valores cercanos 0.05 o menores, en cualquier caso, es aceptable debido a que muestra inactividad. Finalmente, la red adapta bien los pesos y se obtienen salidas de acuerdo a lo diseñado, por lo tanto, el algoritmo de retropropagación funciona correctamente y los pesos se adaptan de manera eficiente.

El perceptrón multicapa logra aprender de los patrones ingresados en la fase de entrenamiento adaptando y actualizando los pesos en cada iteración realizada, luego de este proceso, el Perceptrón multicapa pudo hacer una discriminación correcta para nuevos patrones de entrada que ingresaron en una fase de comprobación, logrando el reconocimiento de patrones numéricos que se requería.

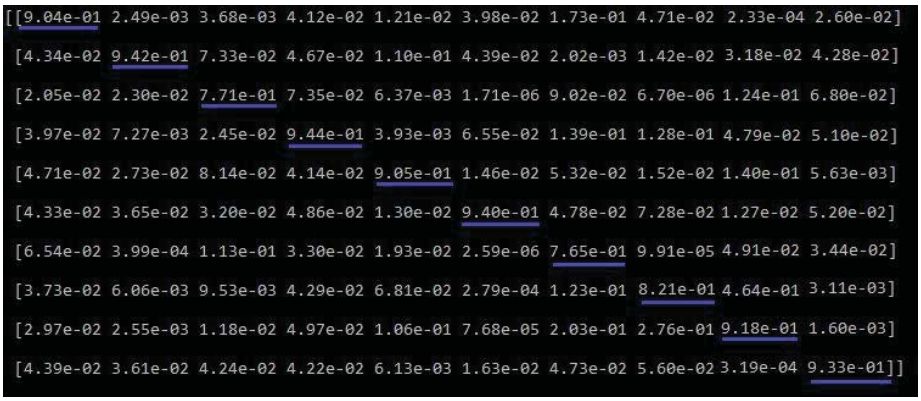

Figura 7-7: Comprobación de la red

7.3 Resultados para patrones desconocidos

El perceptrón multicapa después del proceso de entrenamiento queda funcionando como reconocedor de números desde el cero al nueve. Se ingresan tres patrones nuevos, desconocidos por la red debido a que representan letras. Se espera que el Perceptrón multicapa pueda encontrar similitudes entre los datos aprendidos y los nuevos datos.

Como se puede apreciar en la Figura 7-8, el perceptrón multicapa reconoce estos tres nuevos patrones como 6, 6 y 0 respectivamente. Los valores que se encuentran subrayados con rojo indican el valor que el perceptrón asimiló en dicha neurona, esto quiere decir: se activa la sexta neurona con un valor de 0.96 para el primer patrón ingresado y ésta neurona representa el número 6; se activa la sexta neurona con un valor de 0.97 para el segundo patrón ingresado y ésta neurona representa el número 6; y finalmente, se activa la décima neurona con un valor de 0.652 para el tercer patrón ingresado y ésta representa el número 0. Esto sucede porque las letras A y B, tienen celdas parecidas a las aprendidas en la fase de entrenamiento y el perceptrón encuentra más aciertos con el número 6, pasa lo mismo con la letra C, el perceptrón encuentra más aciertos entre el 0 y ésta letra.

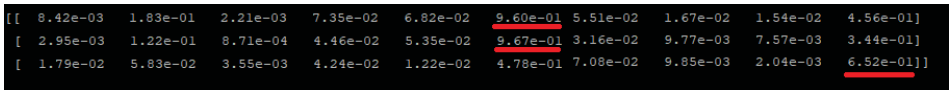

Figura 7-8: Resultados patrones con letra

En la Tabla 7-6 se pueden apreciar todos los datos involucrados en el reconocimiento de los datos desconocidos por la red, se destaca la celda que contiene la neurona activada, para tener una mejor visión de los números que involucran a cada neurona activa en la capa de salida.

Tabla 7-6: Activación en Neuronas de salida

Entrada	N1	N2	N3	N4	N5	N6	N7	N8	N9	N10
A	0.0008	0.183	0.002	0.073	0.068	0.96	0.051	0.016	0.015	0.456
B	0.0029	0.122	0.0008	0.044	0.053	0.967	0.031	0.009	0.007	0.344
C	0.017	0.058	0.003	0.04	0.012	0.47	0.07	0.009	0.02	0.65

La Figura 7-9 y la Figura 7-10 representan la primera y tercera fila de números mostrada en la Figura 7-8 respectivamente. Luego de ingresar el patrón desconocido por la red, ésta intenta encontrar similitudes en los datos aprendidos; en el caso de la letra *A*, la red encontró similitudes con el número 6; y en el caso de la letra *C*, la red encontró similitudes con el número 0.

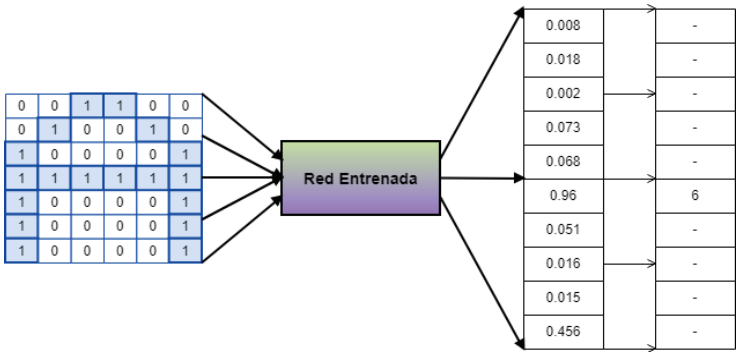

Figura 7-9: Primer patrón desconocido por la red

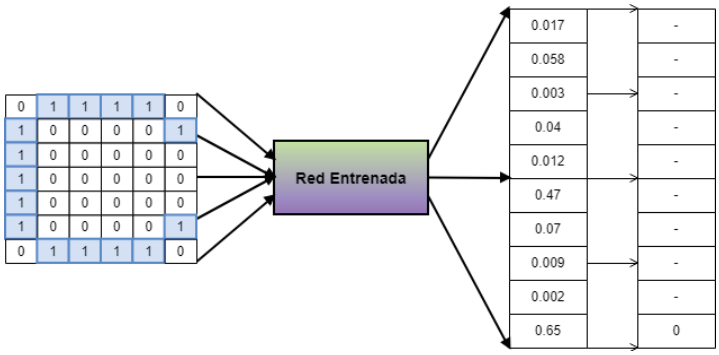

Figura 7-10: Tercer patrón desconocido por la red

Finalmente, se puede concluir que el perceptrón multicapa funciona de manera óptima como un clasificador de patrones, cada vez que se ingresen datos nuevos que no vio en la fase de entrenamiento, este asimilará con los aprendidos en dicha fase y entregará resultados de acuerdo a lo que aprendió anteriormente.

Discusión y conclusiones

La realización del proyecto aborda gran cantidad de contenidos y estudios que fueron llevado a cabo en muchos meses de dedicación tales como: estudio del perceptrón simple y multicapa; estudio de algoritmos para desarrollar redes neuronales artificiales del tipo perceptrón simple y perceptrón multicapa; comunicación entre computador y Raspberry pi; programación en Python; y la implementación de una red neuronal artificial que funcione como reconocedor de patrones numéricos. El diseño de una red neuronal artificial va a depender del tipo de problema que se está dando solución, y éste va a definir la cantidad de neuronas necesarias para la capa de entrada y para la capa de salida, la cantidad de capas y neuronas ocultas que va a presentar la red van a depender exclusivamente del diseñador. En este proyecto, se desarrolló una red neuronal del tipo perceptrón multicapa que fuese capaz de reconocer patrones numéricos desde el cero hasta el nueve. Para el diseño de la red se consideraron: una capa de entrada; una oculta; y una capa de salida. No fue necesario incluir más capas ocultas debido a que con una bastó para dar solución a los requerimientos del problema.

Para el proceso de adaptación y actualización de los pesos de cada una de las conexiones de la red, es necesario realizar un gran número de iteraciones utilizando el algoritmo de retropropagación. Es importante la elección de una buena tasa de aprendizaje, debido a que ésta va a influir en la velocidad de cambios que presentan los pesos en cada iteración realizada. Para la elección de la tasa de aprendizaje se debe considerar: si la tasa de aprendizaje es muy alta, la red va a realizar una convergencia más rápida, pero va a perder valores óptimos donde el error es mínimo; en caso contrario, si la tasa de aprendizaje es demasiado baja, la red va a demorar más tiempo en converger, pero sin perder valores óptimos donde el error es mínimo. El proyecto consideró una tasa de aprendizaje de 0.5 y 10000 iteraciones para la realización del reconocedor de patrones numéricos.

Las elecciones de los patrones de entrada para el proceso de entrenamiento de la red, deben ser significativos y representativos, esto quiere decir: si existe un número reducido de patrones de entrenamiento es posible que la red no sea capaz de adaptar los pesos de cada una de las conexiones de forma eficaz; y que sean representativos quiere decir que la red no se debe especializar en un subconjunto de patrones, por lo tanto, estos deben ser diversos. Queda demostrado que el perceptrón multicapa es capaz de clasificar patrones de entrada que no haya visto en su fase de entrenamiento, incluso si éstos están incompletos o con errores. Esto es debido

a que estas redes son capaces de abstraer las características principales de los patrones que aprenden en el proceso de entrenamiento.

Al ingresar patrones que no correspondan a los aprendidos por la red- en el caso de este proyecto, patrones que correspondan a letras del abecedario- el perceptrón intentará encontrar similitudes con alguno antes visto en la fase de entrenamiento, por lo tanto, para cualquier patrón nuevo que no tenga relación con los aprendidos, la red neuronal artificial responderá con alguno visto anteriormente.

La implementación de una red neuronal perceptrón multicapa es posible realizarla en un computador de placa reducida como es el Raspberry Pi, la cual no presentó mayores problemas al momento de procesar datos. Ésta red perceptrón queda en disposición para futuros trabajos en los que se quiera implementar un sistema de reconocimientos de patrones en forma física, quedando la etapa de clasificación y adaptación funcionando de manera eficiente.

Para trabajos futuros, se puede considerar un mejoramiento en la programación del perceptrón multicapa en Python, y lograr disminuir la cantidad de iteraciones que requiere el programa para adaptar los pesos. También se pueden generar documentos de textos para las salidas del perceptrón y poder hacer un análisis más exhaustivo de lo que pasa con los pesos y el proceso de adaptación de las conexiones, todo esto para mejorar el procesamiento del algoritmo. También se puede considerar cambiar la programación a C.

Bibliografía

[1] BBCMundo, «BBC,» 23 Junio 2012. [En línea]. Available: https://www.bbc.com/mundo/noticias/2012/06/120621_turing_inteligencia_artificial_lp. [Último acceso: 20 Mayo 2018].

[2] Apple, «Apple,» [En línea]. Available: https://www.apple.com/cl/siri/. [Último acceso: 20 Mayo 2018].

[3] Advsyscon, «Advanced Systems Concepts, Inc. (ASCI),» [En línea]. Available: https://www.advsyscon.com/en-us/home. [Último acceso: 20 Mayo 2018].

[4] DeepInstict, «Deep Instict,» [En línea]. Available: https://www.deepinstinct.com/machine-vs-deep/. [Último acceso: 20 Mayo 2018].

[5] SensesTime, «Senses Time,» [En línea]. Available: https://www.sensetime.com/core. [Último acceso: 20 Mayo 2018].

[6] D. M Filatov, K. V. Ignatiev y E. V. Serykh, «Neuronal Network System of Traffic Signs Recognition,» de *2017 XX IEEE International Conference on Soft Computing and Measurements (SCM)*, 2017.

[7] J. F. Castro Gracía, «Fundamentos para la implementación de red neuronal Perceptrón Multicapa mediante Software,» Guatemala, Noviembre de 2006.

[8] H. Vega Huerta, A. Cortez Vásquez, A. M. Huayna, L. Alarcón Loayza y P. Romero Naupari, «Reconocimiento de Patrones Mediante Redes Neuronales Artificiales,» *Revista de Investigaciòn de Sistemas e Informática,* vol. 6, nº 2, pp. 17-26, 2009.

[9] E. Varela Arregocés y E. Campbells Sánchez, «Redes Neuronales Artificiales: Una revisión, Estado del arte, aplicaciones y tendencias futuras,» *Investigación y Desarrollo en TIC,* vol. 2, nº 1, pp. 18-27, 2011.

[10] J. B. Blaya, *Aprendizaje por Refuerzo: Tratamiento Inteligente de la Información y Aplicaciones,* Murcia España, 2009.

[11] Y. Tang, R. Salakhutdinov y G. Hinton, «Robust Boltzmann Machine for Recognition and Denoising,» de *IEEE Computer Vision and Pattern Recognition (CVPR)*, 2012.

[12] E. Medina, H. Cubides, J. Salazar y J. SIgüencia, «Redes Adaline- Filtros Adaptativos,» 2010. [En línea]. Available: https://es.scribd.com/document/52002261/Adeline-1. [Último acceso: 15 Marzo 2018].

[13] I. M. Galván León y P. Isasi Viñuela, Redes neuronales artificiales, un enfoque práctico, Madrid: Pearson Prentice Hall, 2004.

[14] R. Pi, «Raspberry Pi,» [En línea]. Available: https://www.raspberrypi.org/. [Último acceso: 20 Mayo 2018].

[15] Codejobs, 2013. [En línea]. Available: https://www.codejobs.biz/es/blog/2013/03/02/que-es-python. [Último acceso: 8 Abril 2018].

[16] M. Bartolomé Sintes, «MCLibre - Material Curricular Libre,» 2018. [En línea]. Available: http://www.mclibre.org/consultar/python/otros/historia.html. [Último acceso: 13 Abril 2018].

A Código Python Perceptrón Multicapa

Listado A-1

```
#perceptron multicapa
import numpy as np
import matplotlib.pyplot as plt
from pylab import plot, ylim, xlim
np.set_printoptions(precision=2)

entrada                                                                                    =
np.array([[0,0,0,1,0,0,0,0,1,1,0,0,0,0,0,1,0,0,0,0,0,1,0,0,0,0,0,1,0,0,0,0,0,1,0,0
,0,0,1,1,1,0],

        [0,0,0,1,0,0,0,0,0,1,0,0,0,0,0,1,0,0,0,0,0,1,0,0,0,0,0,1,0,0,0,0,0,1,0,0,0,
0,1,1,1,0],

        [0,1,1,1,1,0,1,0,0,0,0,1,0,0,0,0,1,0,0,0,0,1,0,0,0,0,1,0,0,0,0,1,0,0,0,0,1,
1,1,1,1,1],

        [0,1,1,1,1,0,0,0,0,0,0,1,0,0,0,0,1,0,0,0,0,1,0,0,0,0,1,0,0,0,0,1,0,0,0,0,0,
1,1,1,1,1],

[0,1,1,1,1,1,0,0,0,0,0,1,0,0,0,0,0,1,0,0,0,1,1,0,0,0,0,0,0,1,0,1,0,0,0,1,0,0,1,1,1
,0],

[0,0,1,1,1,0,0,0,0,0,0,1,0,0,0,0,0,1,0,0,0,1,1,0,0,0,0,0,0,1,0,0,0,0,0,1,0,0,1,1,1
,0],

[0,0,0,0,1,0,0,0,0,1,1,0,0,0,1,0,1,0,0,1,0,0,1,0,0,1,1,1,1,1,0,0,0,0,1,0,0,0,0,0,1
,0],

[0,0,0,0,0,0,0,0,0,1,1,0,0,0,1,0,1,0,0,1,0,0,1,0,0,1,1,1,1,0,0,0,0,0,1,0,0,0,0,0,1
,0],

[1,1,1,1,1,1,1,0,0,0,0,0,1,0,0,0,0,0,1,1,1,1,1,0,0,0,0,0,1,1,0,0,0,0,1,0,1,1,1,1
,0],

[0,1,1,1,1,0,1,0,0,0,0,0,1,0,0,0,0,0,0,1,1,1,1,0,0,0,0,0,1,0,0,0,0,0,1,0,1,1,1,1
,0],

[0,1,1,1,1,0,1,0,0,0,0,0,1,0,0,0,0,0,1,1,1,1,1,0,1,0,0,0,0,1,1,0,0,0,0,1,0,1,1,1,1
,0],

[0,1,1,1,0,0,1,0,0,0,0,0,1,0,0,0,0,0,0,1,1,1,1,0,1,0,0,0,0,1,1,0,0,0,0,1,0,1,1,1,1
,0],

[0,1,1,1,1,1,0,0,0,0,0,1,0,0,0,0,0,1,0,0,0,0,1,0,0,0,0,1,0,0,0,0,0,1,0,0,0,0,0,1,0
,0],
```

```
[0,0,1,1,1,0,0,0,0,0,0,1,0,0,0,0,0,1,0,0,0,0,1,0,0,0,0,1,0,0,0,0,0,1,0,0,0,0,0,0,0
,0],

[0,1,1,1,1,0,1,0,0,0,0,1,1,0,0,0,0,1,0,1,1,1,1,0,1,0,0,0,0,1,1,0,0,0,0,1,0,1,1,1,1
,0],

[0,1,1,1,1,1,0,0,0,0,1,1,0,0,0,0,1,0,1,1,1,1,0,1,0,0,0,0,1,1,0,0,0,0,1,1,1,1,1,1,1
,1],

[0,1,1,1,1,0,1,0,0,0,0,1,1,0,0,0,0,1,0,1,1,1,1,1,0,0,0,0,0,1,0,0,0,0,0,1,0,1,1,1,1
,0],

[0,1,1,1,1,0,1,0,0,0,0,1,1,0,0,0,0,1,0,1,1,1,1,0,0,0,0,0,0,1,0,0,0,0,0,1,0,0,0,1,1
,0],

[0,1,1,1,1,0,1,0,0,0,1,1,1,0,0,1,0,1,1,0,1,0,0,1,1,1,0,0,0,1,1,0,0,0,0,1,0,1,1,1,1
,0],

[0,1,1,1,1,0,1,0,0,0,0,1,1,0,0,0,0,1,1,0,0,0,0,1,1,0,0,0,0,1,1,0,0,0,0,1,0,1,1,1,1
,0]])


salida_deseada = np.array([[0.95, 0.05, 0.05, 0.05, 0.05, 0.05, 0.05, 0.05, 0.05,
0.05],
                           [0.95, 0.05, 0.05, 0.05, 0.05, 0.05, 0.05, 0.05, 0.05,
0.05],
                           [0.05, 0.95, 0.05, 0.05, 0.05, 0.05, 0.05, 0.05, 0.05,
0.05],
                           [0.05, 0.95, 0.05, 0.05, 0.05, 0.05, 0.05, 0.05, 0.05,
0.05],
                           [0.05, 0.05, 0.95, 0.05, 0.05, 0.05, 0.05, 0.05, 0.05,
0.05],
                           [0.05, 0.05, 0.95, 0.05, 0.05, 0.05, 0.05, 0.05, 0.05,
0.05],
                           [0.05, 0.05, 0.05, 0.95, 0.05, 0.05, 0.05, 0.05, 0.05,
0.05],
                           [0.05, 0.05, 0.05, 0.95, 0.05, 0.05, 0.05, 0.05, 0.05,
0.05],
                           [0.05, 0.05, 0.05, 0.05, 0.95, 0.05, 0.05, 0.05, 0.05,
0.05],
                           [0.05, 0.05, 0.05, 0.05, 0.95, 0.05, 0.05, 0.05, 0.05,
0.05],
                           [0.05, 0.05, 0.05, 0.05, 0.05, 0.95, 0.05, 0.05, 0.05,
0.05],
                           [0.05, 0.05, 0.05, 0.05, 0.05, 0.95, 0.05, 0.05, 0.05,
0.05],
                           [0.05, 0.05, 0.05, 0.05, 0.05, 0.05, 0.95, 0.05, 0.05,
0.05],
                           [0.05, 0.05, 0.05, 0.05, 0.05, 0.05, 0.05, 0.95, 0.05,
0.05],
                           [0.05, 0.05, 0.05, 0.05, 0.05, 0.05, 0.05, 0.95, 0.05,
0.05],
                           [0.05, 0.05, 0.05, 0.05, 0.05, 0.05, 0.05, 0.05, 0.95,
0.05],
                           [0.05, 0.05, 0.05, 0.05, 0.05, 0.05, 0.05, 0.05, 0.95,
0.05],
                           [0.05, 0.05, 0.05, 0.05, 0.05, 0.05, 0.05, 0.05, 0.05,
0.95],
                           [0.05, 0.05, 0.05, 0.05, 0.05, 0.05, 0.05, 0.05, 0.05,
0.95]])

tasa = 0.5; ciclos = 20000; muestras = 100; n_oculta = 20; error_plot = []
bias1 = 0.25; bias2 = 0.4
def sigmoide (x):
    return 1 / ( 1 + np.exp(-x))
def d_sigmoide(x):
    return x * (1 - x)
np.random.seed(1)
```

```
w1 = 1 * np.random.random((len(entrada[0]), n_oculta)) - 1
w2 = 1 * np.random.random((len(w1[0]), len(salida_deseada[0]))) - 1

#iteraciones
for iteracion in xrange(ciclos):
    #propagacion hacia adelante

    entrada_h = np.dot(entrada, w1) + bias1
    salida_h = sigmoide(entrada_h)
    entrada_o = np.dot(salida_h, w2) + bias2
    salida_o = sigmoide(entrada_o) ## salida de la red

    ##error cuadratico medio = 1/2 *(salida obtenida - salida deseada)^2
    error_salida_o = 0.5 * (np.power((salida_o - salida_deseada),2))
    suma_error = (np.mean(np.abs(error_salida_o)))
    error_plot.append(suma_error)
    #propagacion hacia atras

    # capa de salida
    ## (dE/dw2) = (dE/ds_o)(ds_o/de_o)(de_o/dw2)
    d_error_salida_o = salida_o - salida_deseada ## (dE/ds_o)
    d_salida_derivada = d_sigmoide(salida_o) ## (ds_o/de_o)
    d_pesos = salida_h ##(de_o/dw2)
    ##(dE/dw2) resultado final de la capa de salida
    d_capa_salida = np.dot(d_pesos.T,(d_error_salida_o * d_salida_derivada))

    # capa oculta
    ##(dE/dw1)=(dE/ds_h)(ds_h/de_h)(de_h/dw1)
    d_salida_h = np.dot(d_error_salida_o * d_salida_derivada, w2.T) ##(dE/ds_h)
    d_entrada_h = d_sigmoide(salida_h) ## (ds_h/de_h)
    d_entrada = entrada ##(de_h/dw1)
    ## (dE/dw1) resultado final de la capa oculta
    d_capa_oculta = np.dot(d_entrada.T, d_salida_h * d_entrada_h)

    #actualizacion de los pesos capa oculta y salida
    ##w1 = w1 - tasa * dE/dw
    w1 -= tasa * d_capa_oculta
    w2 -= tasa * d_capa_salida
    '''if (iteracion% muestras) == 0 : ##muestras del error en la salida
        print("Iteracion  (%s)--->Error:  %s--->pesos  %s  -->%s"  %  (iteracion+1,
suma_error,w1, w2))'''

print("\nsalida de la red : \n%s" % (salida_o))

## patrones de corroboracion
X                                                                                =
np.array([[0,0,0,1,0,0,0,0,0,1,0,0,0,0,0,1,0,0,0,0,0,1,0,0,0,0,0,1,0,0,0,0,0,1,0,0
,0,0,1,0,0],

[0,1,1,1,1,0,0,0,0,0,0,1,0,0,0,0,1,0,0,0,0,1,0,0,0,0,1,0,0,0,0,1,0,0,0,0,1,1,1,1,1
,0],

[0,0,1,1,1,0,0,0,0,0,0,1,0,0,0,0,0,1,0,0,0,0,1,0,0,0,0,0,0,1,0,0,0,0,0,1,0,0,0,1,1
,0],

[0,0,0,0,0,0,0,0,0,1,1,0,0,0,1,0,1,0,0,1,0,0,1,0,0,1,1,1,1,0,0,0,0,0,1,0,0,0,0,0,0
,0],

[0,1,1,1,1,0,1,0,0,0,0,0,1,0,0,0,0,0,0,1,1,1,1,0,0,0,0,0,0,1,0,0,0,0,0,1,0,0,1,1,1
,0],

[0,0,0,0,0,0,1,0,0,0,0,0,1,0,0,0,0,0,0,1,1,1,1,1,0,1,0,0,0,0,1,1,0,0,0,0,1,0,1,1,1
,0],

[0,0,1,1,1,0,0,0,0,0,0,1,0,0,0,0,0,1,0,0,0,1,1,0,0,0,0,1,0,0,0,0,0,1,0,0,0,0,0,1,0
,0],

[0,1,1,1,1,0,1,0,0,0,0,1,1,0,0,0,1,1,1,1,1,1,1,1,0,0,0,0,1,1,0,0,0,0,1,1,0,1,1,1,1
,0],
```

```
[0,1,1,1,1,0,1,0,0,0,0,1,1,0,0,0,0,1,0,1,1,1,1,0,0,0,0,0,0,1,0,0,0,0,0,1,0,0,0,0,0
,0],

[1,1,1,1,1,1,1,0,0,0,0,1,1,0,0,0,0,1,1,0,0,0,0,1,1,0,0,0,0,1,1,0,0,0,0,1,0,0,0,0,1
,0]])

for i in xrange(1):
    z = np.dot(X, w1) + 0.15
    h1 = sigmoide(z)
    z3 = np.dot(h1, w2) + 0.25
    o = sigmoide(z3)

print("\nla nueva salida es : \n%s" % (o))

#plot
ylim([0,0.5]); xlim([0,ciclos]);plt.xlabel("Iteraciones"); plt.ylabel("Error")
plt.plot(error_plot, color='r');plt.title("Perceptron Multicapa")
plt.grid(True); plt.show()
```

B Código Python Perceptrón Simple

Listado B-1

```
#----------perceptron simple------------
import numpy as np
import matplotlib.pyplot as plt
from pylab import plot, ylim, xlim
np.set_printoptions (precision = 4)

def escalon(x):
    return np.where(x >= 0.0, 1, 0)

##                   x1   x2  bias
entrada = np.array([[0,   0,   1],
                    [0,   1,   1],
                    [1,   0,   1],
                    [1,   1,   1]])

sd = np.array([[0],
               [0],
               [0],
               [1]])

tasa = 1; ciclos = 20; muestras = 1;
error_plot = [];

##               w1    w2    w0
w1 = np.array([[-0.5], [1],[-1.5]])
print("\nSalida deseada: \n%s \nPesos Iniciales \n    w1   w2  w0\n%s\n"
% (sd, w1.T))

for iteracion in xrange(ciclos):
    z = np.dot(entrada, w1)
    out = escalon(z) ##salida de la red
    error = sd - out
    z_error = (np.mean(np.abs(error)))
    error_plot.append(z_error)

    ##actualizacion de los pesos
    if z_error == 0 :
        pass
    else:
        if error[0] == 1:
            error = np.array([[1],[0],[0],[0]])
        else:
            pass
        if error[1] == 1:
            error = np.array([[0],[1],[0],[0]])
        else:
            pass
        if error[2] == 1:
            error = np.array([[0],[0],[1],[0]])
        else:
```

```
            pass
        if error[3] == 1:
                error = np.array([[0],[0],[0],[1]])
        else:
            pass ##pass #aqui poner la condicion de termino
        w1 += tasa * (np.dot(entrada.T, error))
    if (iteracion% muestras) == 0 :
        print("{Iteracion    %d}    z:%s-->f():%s-->error%s-->w%s"    %
(iteracion + 1, z.T,out.T,error.T,w1.T))
    if z_error == 0 :
        break

print("\nPesos Adaptados:\n    w1  w2  w0 \n%s\n \nSalida de la red :
\n%s" % (w1.T, out))
```

C Código Ejemplo Perceptrón Multicapa

Listado C-1

```
#ejemplo perceptron multicapa


import numpy as np
import matplotlib.pyplot as plt
from pylab import plot, ylim, xlim
np.set_printoptions(precision=6)

entrada = np.array([[0.01, 0.1]])
salida_deseada = np.array([[0.01]])

tasa = 1; ciclos = 3000; muestras = 1; n_oculta = 2
error_plot = []; bias1 = 0.35; bias2 = 0.4

#funcion sigmoide y derivada
def sigmoide (x):
    return 1 / ( 1 + np.exp(-x))
def d_sigmoide(x):
    return x * (1 - x)

w1 = np.array ([[0.1,0.2],[0.3,0.4]])
w2 = np.array([[0.5],[0.6]])

#iteraciones
for iteracion in xrange(ciclos):

#propagacion hacia adelante
    entrada_h = np.dot(entrada, w1) + bias1
    salida_h = sigmoide(entrada_h)
    entrada_o = np.dot(salida_h, w2) + bias2
    salida_o = sigmoide(entrada_o) ## salida de la red
    ##error cuadratico medio = 1/2 *(salida obtenida - salida deseada)^2
    error_salida_o = 0.5 * (np.power((salida_o - salida_deseada),2))
    suma_error = (np.mean(np.abs(error_salida_o)))
    error_plot.append(suma_error)

#propagacion hacia atras
    # capa de salida
    ## (dE/dw2) = (dE/ds_o)(ds_o/de_o)(de_o/dw2)
    d_error_salida_o = salida_o - salida_deseada ## (dE/ds_o)
    d_salida_derivada = d_sigmoide(salida_o) ## (ds_o/de_o)
    d_pesos = salida_h ##(de_o/dw2)
    ##(dE/dw2) resultado final de la capa de salida
    d_capa_salida      =      np.dot(d_pesos.T,(d_error_salida_o      *
d_salida_derivada))
        # capa oculta
    ##(dE/dw1)=(dE/ds_h)(ds_h/de_h)(de_h/dw1)
    d_salida_h  =  np.dot(d_error_salida_o  *  d_salida_derivada,  w2.T)
##(dE/ds_h)
    d_entrada_h = d_sigmoide(salida_h) ## (ds_h/de_h)
    d_entrada = entrada ##(de_h/dw1)
```

```
    ## (dE/dw1) resultado final de la capa oculta
    d_capa_oculta = np.dot(d_entrada.T, d_salida_h * d_entrada_h)

#actualizacion de los pesos capa oculta y salida
    ## w1 = w1 - tasa * dE/dw
    w1 -= tasa * d_capa_oculta
    w2 -= tasa * d_capa_salida
    if (iteracion% muestras) == 0:
        print("Iteracion   (%s)--->Error:   %s--->   salida   %s"   %
(iteracion+1, suma_error, salida_o))

#plot
ylim([0,0.3]); xlim([0,ciclos]); plt.xlabel("Iteraciones")
plt.ylabel("Error"); plt.plot(error_plot, color='r')
plt.title("Perceptron Multicapa"); plt.grid(True); plt.show()
```