UNIVERSIDAD DE SONORA

UNIDAD REGIONAL SUR

DIVISION DE CIENCIAS E INGENIERIA

DEPARTAMENTO DE FISICA, MATEMATICAS E INGENIERIA

APRENDIZAJE DE UNA RED NEURONAL ARTIFICIAL MEDIANTE EL ALGORITMO DE RETROPROPAGACION UTILIZANDO EL SOFTWARE MATLAB

T E S I S

QUE PARA OBTENER EL TITULO DE

INGENIERO INDUSTRIAL Y DE SISTEMAS

PRESENTA

BUITIMEA ZAZUETA OMAR ALEJANDRO

NAVOJOA, SONORA **MARZO DE 2012**

Universidad de Sonora

Repositorio Institucional UNISON

UNIVERSIDAD DE SONORA

UNIDAD REGIONAL SUR

DIVISIÓN DE CIENCIAS E INGENIERÍA

DEPARTAMENTO DE FÍSICA, MATEMÁTICAS E INGENIERÍA

Los miembros del comité revisor, recomendamos que este trabajo profesional sea aceptado como requisito parcial para la obtención del título de Ingeniero Industrial y de Sistemas.

COMITÉ REVISOR

DIRECTOR: _____

Dr. GÓMEZ ALDAMA ÓSCAR RUBÉN.

ASESOR: Viridiana Gómez B.

M.C. GÓMEZ BARRÓN VIRIDIANA

SINODAL: _____

M.I. GARCÍA OCHOA JUAN JOSÉ

SINODAL: _____

Dr. CASTRO ARCE LAMBERTO

Cd. Navojoa, Sonora; México. Marzo de 2012.

AGRADECIMIENTOS

Primeramente se agradece a Dios por permitirme llegar hasta este momento.

A LA UNIVERSIDAD DE SONORA. Por la formación académica que me brindó en el transcurso de mis estudios.

A mis padres, familia y hermanos por su apoyo y comprensión.

CON ESPECIAL RECONOCIMIENTO

A MI DIRECTOR DE TESIS Dr. ÓSCAR RUBÉN GÓMEZ ALDAMA; por su tiempo, dedicación y esfuerzo en la dirección de este trabajo profesional.

DEDICATORIA

El autor dedica esta tesis a sus padres: Benigno Buitimea Acosta y Balvina Zazueta González, hermanos: Benigno, Juan Carlos, José Cruz y Felipe, y a todos mis seres queridos familiares y amigos(as).

ÍNDICE

LISTA DE FIGURAS

LISTA DE TABLAS

LISTA DE ANEXOS

RESUMEN

Las redes neuronales artificiales (RNA) son un paradigma para hacer cómputo y para la detección de patrones basado en la interconexión paralela de neuronas artificiales. Son un modelo basado en los complejos sistemas nerviosos de los animales y seres humanos con su gran cantidad de interconexiones y paralelismo. Las RNA son modelos computacionales que tratan de replicar, de manera simplificada, el complejo funcionamiento del cerebro humano. Su capacidad de aprendizaje a través de ensayos repetidos, las ha hecho muy populares en una amplia variedad de aplicaciones en todas las ciencias. En el capítulo primero se definen los principios de la metodología de las redes neuronales, además del objetivo, alcances y limitaciones, se inicia con las redes neuronales biológicas y

los procesos que estos involucran, así como su relación con las artificiales. En el capítulo segundo se analizan las facetas del proceso de aprendizaje y sus características, así como el algoritmo de aprendizaje de retropropagación, también se estudia el perceptrón en su forma más simple y del perceptrón multicapa, entrenado con el algoritmo de retropropagación. En el capítulo tercero se aplica el algoritmo de retropropagación a una RNA por medio del software de Matlab con diferentes funciones de activación y factor de aprendizaje. En el capítulo cuarto se analizan los resultados obtenidos del algoritmo de aprendizaje de retropropagación. En el capítulo quinto se concluye que el objetivo planteado se satisface, ya que se obtuvieron los valores óptimos del factor de aprendizaje, así como, la función de activación que en conjunto minimizan el error.

INTRODUCCIÓN

En las últimas décadas las redes neuronales artificiales (RNA) han recibido un interés particular como una tecnología para modelar de manera efectiva y eficiente problemas grandes y complejos. Las RNA tienen muchas aplicaciones en ingeniería industrial, (Muñoz, 1996) realizó en su tesis doctoral la aplicación de técnicas de redes neuronales artificiales al diagnóstico de procesos industriales. (Paternina, Lerin y Márceles, 2001) realizaron un enfoque de redes neuronales hacia el control de calidad para procesos multivariados. Los modelos de RNA son dirigidos a partir de los datos, es decir, son capaces de encontrar relaciones (patrones) de forma inductiva por medio de los algoritmos de aprendizaje basados en los datos existentes más que requerir de ayuda de un modelo para especificar la forma funcional y sus interacciones.

Las RNA son un método para resolver problemas, de forma individual o combinada con otros métodos, para aquellas tareas de clasificación, identificación, diagnóstico, optimización o predicción en las que el balance de datos-conocimiento se inclina hacia los datos y donde, adicionalmente, existe la necesidad de aprendizaje en tiempo de ejecución y de cierta tolerancia a fallos. ***En estos casos las RNA se adaptan dinámicamente reajustando constantemente los "pesos" de sus interconexiones.***

Las redes neuronales se componen de elementos simples que funcionan en paralelo. Estos elementos se inspiran en los sistemas biológicos nerviosos. Como en la naturaleza, la función de red es determinada en gran medida por las conexiones entre los elementos. Usted puede entrenar una red neuronal en matrix-laboratory (Matlab) para realizar una función determinada mediante el ajuste de los valores de las conexiones (pesos) entre los elementos.

Redes neuronales comúnmente se ajustan, o la formación, de manera que una entrada en particular conduce a una de salida de destino específico. La red se ajusta, sobre la base de una comparación de la salida y el valor deseado, hasta que la salida de la red coincide con la objetivo. Por lo general muchos de estos datos de entrada son necesarios para capacitar a una red.

Las redes neuronales han sido entrenadas para realizar funciones complejas en diversos ámbitos, incluyendo el reconocimiento de patrones, identificación, clasificación, el habla, la visión y el control sistemas.

Actualmente, las redes neuronales son entrenadas para resolver problemas que son difíciles para los convencionales ordenadores o seres humanos. Durante todo el énfasis se coloca en caja de herramientas neuronales paradigmas de la red que se acumulan para sí mismos o son utilizados en la ingeniería, financieros y otras aplicaciones prácticas.

Las RNA se basan en la analogía que existe en el comportamiento y función del cerebro humano, en particular del sistema nervioso, el cual está compuesto por

redes de neuronas biológicas que poseen bajas capacidades de procesamiento, sin embargo toda su capacidad cognitiva se sustenta en la conectividad de éstas. La unidad de una RNA es un procesador elemental llamado neurona que posee la capacidad limitada de calcular, en general, una suma ponderada de sus entradas y luego le aplica una función de activación para obtener una señal que será transmitida a la próxima neurona. Estas neuronas artificiales se agrupan en capas o niveles y poseen un alto grado de conectividad entre ellas, conectividad que es ponderada por los pesos. A través de un algoritmo de aprendizaje supervisado o no supervisado, las RNA ajustan su arquitectura y parámetros de manera de poder minimizar alguna función de error que indique el grado de ajuste a los datos y la capacidad de generalización de las RNA.

I.1 Planteamiento del problema.

Las redes neuronales y el software de matrix laboratory (Matlab) son herramientas de gran uso en ingeniería industrial, es por eso que se modelará el algoritmo de retropropagación a través de RNA utilizando Matlab para determinar el valor óptimo del factor de aprendizaje de una red.

I.2 Justificación.

El algoritmo de retropropagación se modelará a través de una RNA y se resolverá usando el software de Matlab. La RNA se utilizan para resolver problemas que son difíciles para los convencionales ordenadores o seres humanos, donde se incluye el reconocimiento de patrones, identificación, clasificación, el habla, la visión y el control de sistemas, entre otros, aumentando la eficiencia y eficacia .

I.3 Objetivo.

Aplicar el algoritmo de aprendizaje de retropropagación en una RNA utilizando el software de Matlab con la finalidad de determinar el factor de aprendizaje óptimo.

I.4 Alcances.

Comparar los diferentes factores de aprendizaje de una RNA utilizando Matlab.

I.4.1 Limitaciones del estudio.

Este estudio se limita a la aplicación del algoritmo de aprendizaje de retropropagación a una RNA utilizando el software Matlab.

II MARCO TEÓRICO

II.1 Redes neuronales biológicas.

Una célula es una unidad o componente básico de los sistemas biológicos. Sus componentes básicos son las neuronas ya que estas forman parte del sistema nervioso, incluyendo al cerebro, son células que presentan una forma especial; tienen tres partes principales: las dendritas, el cuerpo de la célula o soma y el axón, como se muestra en la Figura 1.

(Ramón y Cajal, 1888), demuestran que el sistema nervioso está compuesto por una red de células individuales, las neuronas, ampliamente interconectadas entre sí.

La información fluye desde las dendritas hacia el axón atravesando el soma.

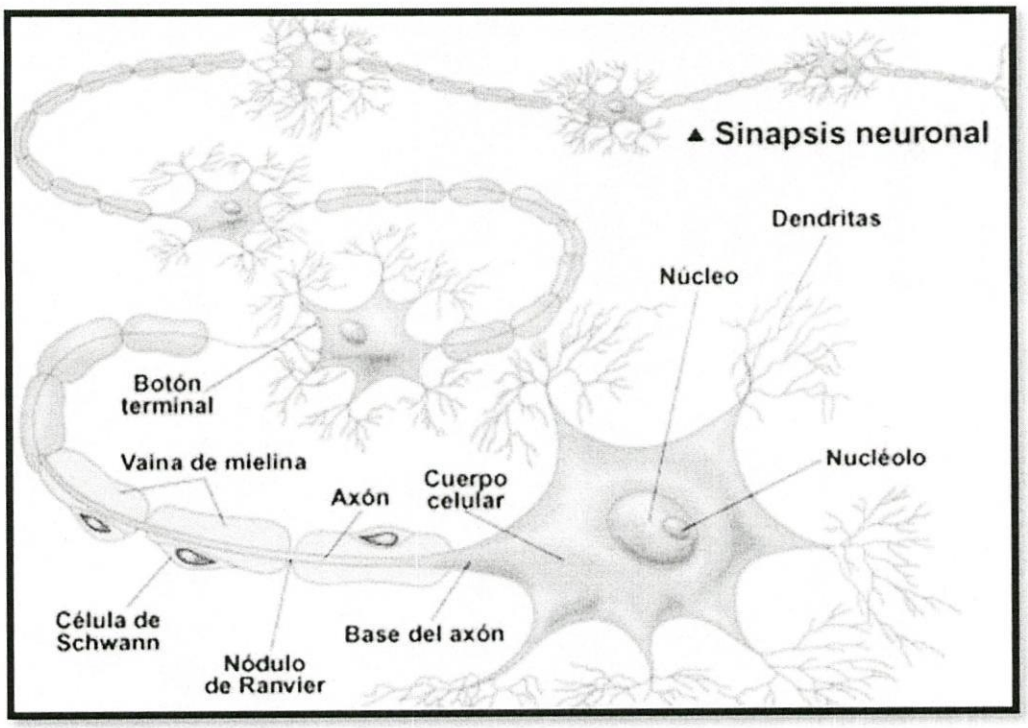

Figura 1. Red neuronal biológica.

El cerebro humano promedio cuenta con aproximadamente 10^{11} neuronas y 10^{15} interconexiones, con una longitud de 10 a 80 micras. Del soma surge un denso árbol de ramificaciones (árbol dendrítico) formado por las dendritas. Del soma parte una fibra tabular denominada axón que mide de 100 micras hasta un metro de longitud. El axón se ramifica en su extremo final para conectarse con otras neuronas. Así mismo, durante las sinapsis (uniones entre neuronas) cada una de estas neuronas recibe en promedio alrededor de 10,000 estímulos de entrada y genera alrededor de 10,000 estímulos de salida. En este sentido, la principal ventaja del cerebro humano promedio recae en su conectividad, interpretada como la capacidad del mismo para realizar diferentes procedimientos lógicos a la vez. Aunque cabe mencionar que algunas neuronas reciben información directamente del exterior.

El objetivo de las redes neuronales de tipo biológico se constituye en desarrollar un elemento sintáctico que permita verificar las hipótesis correspondientes a los demás sistemas biológicos. Es decir, las redes neuronales de tipo biológico deben recibir y procesar información de otros sistemas biológicos y devolver una respuesta de acción efectiva.

II.1.1 Sinapsis.

La membrana de la neurona genera impulsos eléctricos y transfiere la información a otras neuronas por medio de la sinapsis, la información viaja a lo largo de los axones en breves impulsos eléctricos (Stevens, 1979). Los axones tienen una cubierta que se denomina vaina de mielina, la cual funciona como una capa aislante; ésta es interrumpida en varios puntos por los nodos de Ranvier (Simpson, 1990).

El estímulo es creado por la información que llega a través de las dendritas. La apertura del canal se ve como un cambio de conducta de la membrana. El impulso se propaga a lo largo del axón por el cambio de conductancia de la membrana, que abre y cierra canales a los iones de sodio y potasio. Los potenciales de acción alcanzan una amplitud máxima de unos 100 mV y duran aproximadamente 1 ms. La membrana en reposo mantiene un gradiente de potencial eléctrico de -70 mV.

Las fibras nerviosas en si son malos conductores. La transmisión del potencial de acción a los largo del axón es resultado de una serie de despolarizaciones que tienen lugar en los nodos de Ranvier. Se llaman nodos de Ranvier a las interrupciones que ocurren a intervalos regulares a lo largo de la longitud del axón en la vaina de mielina que lo envuelve. Son pequeñísimos espacios, de un micrómetro de longitud, que exponen a la membrana del axón al líquido extracelular. Cuando uno de los nodos se despolariza, se desencadena la despolarización del siguiente nodo. El potencial de acción viaja a lo largo de la fibra en forma discontinua de un nodo a otro. Una vez que un potencial de acción ha pasado por un cierto punto, este no puede volver a ser excitado durante

aproximadamente 1 ms, que es el tiempo que tarda en volver su potencial de reposo. Este periodo refractario limita la frecuencia de trasmisión de los impulsos nerviosos.

La comunicación entre las neuronas tiene lugar cuando el impulso llega al botón sináptico, lo que provoca la liberación de neurotransmisores por parte de la célula presináptica y la absorción de esto por la célula postsináptica, lo que tiene lugar en la hendidura sináptica. Así la sinapsis convierte una señal eléctrica de la neurona pre-sináptica en un proceso químico en la sinapsis, que después se reconvierte en una señal eléctrica en la neurona post-sináptica.

Existen dos tipos de neurotransmisores: los excitatorios, que incrementan el potencial de membrana, y los inhibitorios, que decrementan dicho potencial.

II.2 Red neuronal artificial

Las RNA son modelos simplificados de las redes neuronales biológicas. Tratan de extraer las excelentes capacidades del cerebro para resolver ciertos problemas complejos, como: visión, reconocimiento de patrones o control moto-sensorial. En la Figura 2 se muestra la representación esquemática de una RNA. Estas también llamadas neuro-computadoras, red conexionista, procesador paralelo distribuido, entre otras. Es un procesador paralelo distribuido y masivamente interconectado que almacena conocimiento experimental (Haykin, 1999). Las RNA presentan las siguientes características:

- El conocimiento es adquirido experimentalmente.
- Los pesos (ganancias) de interconexión (sinapsis) varían constantemente.

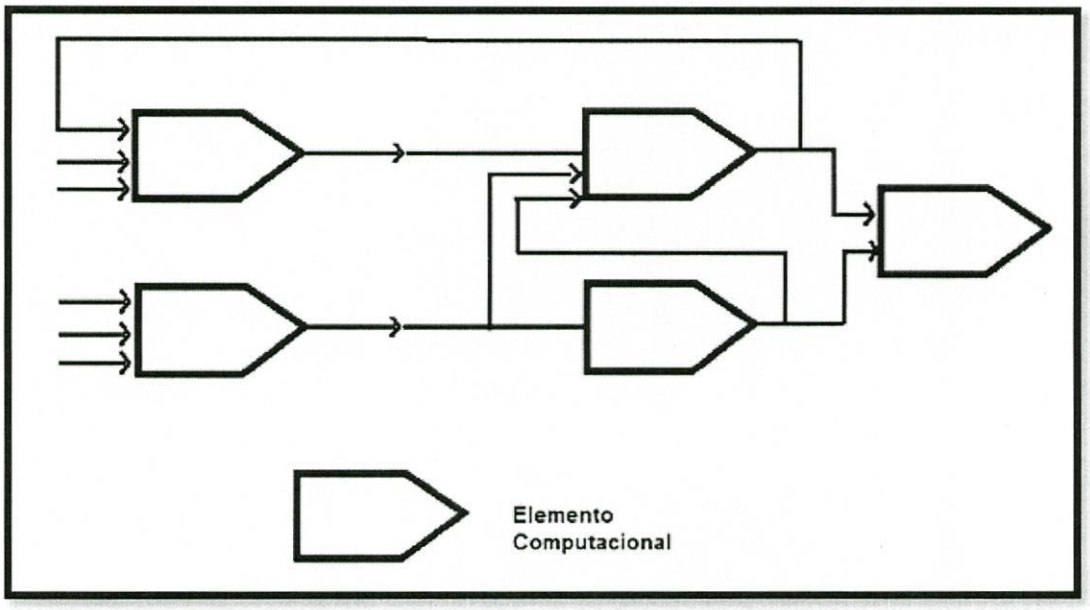

Figura 2. Representación de una red neuronal artificial.

II.2.1 Ventajas de las RNA.

Las RNA ofrecen ventajas como:

- **No linealidad.** El procesador neuronal es básicamente no lineal y, por consecuencia, la red neuronal también.
- **Transformación entrada-salida.** El proceso de aprendizaje consiste básicamente en presentar a la red un ejemplo y modificar sus pesos sinápticos de acuerdo con su respuesta. Aprende, por lo tanto, una transformación entrada/salida.
- **Adaptabilidad.** La red tiene la capacidad de adaptar sus parámetros, aun en tiempo real.
- **Tolerancia a fallas.** Debido a la interconexión masiva, la falla de un procesador no altera seriamente la operación.
- **Uniformidad en el análisis y diseño.** Esto permite garantizar características precisas.

- **Analogía con las redes biológicas.** Esto permite la utilización mutua del conocimiento de las dos áreas.

A partir de la siguiente sección se omitirá la especificación de la palabra artificial ya que se tratara exclusivamente este tipo de neuronas.

II.3 Modelo de una RNA.

La neurona es la unidad de proceso de información fundamental en una red neuronal (Haykin, 1999). En la Figura 3 se muestra el modelo de una neurona; éste es el elemento básico de una RNA.

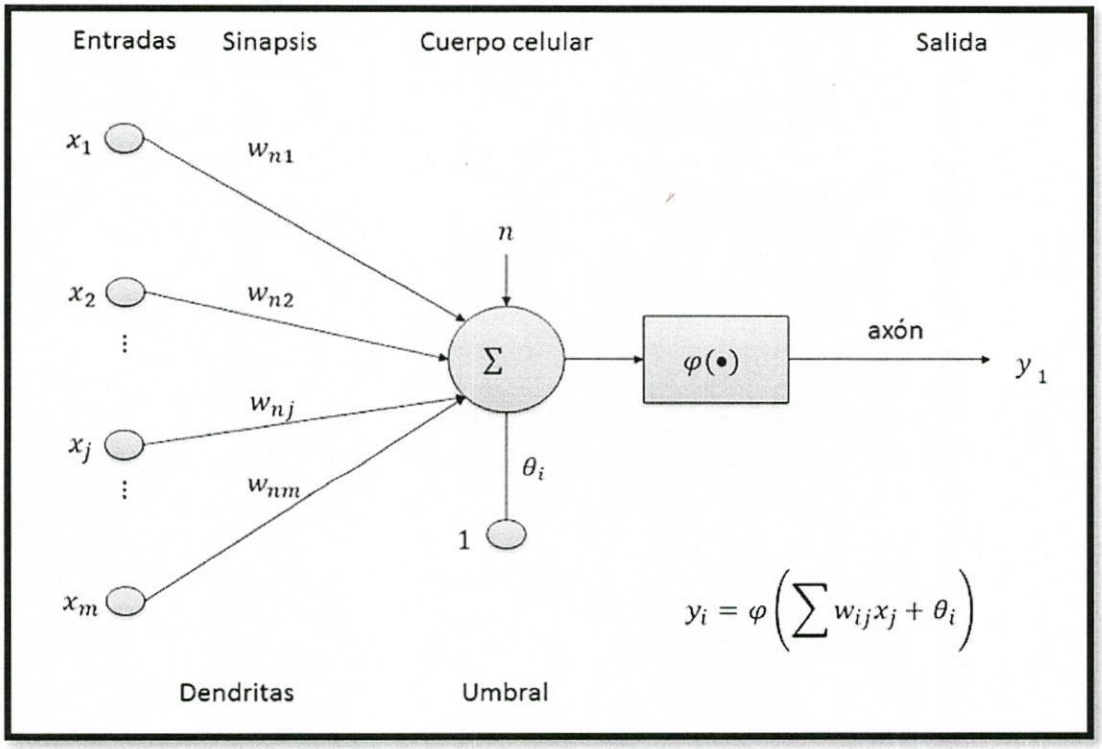

Figura 3. Modelo de una neurona artificial.

En el modelo de una neurona como se muestra en la Figura 3 se identifican cuatro elementos:

a) Enlaces de conexión. Parametrizados por los pesos sinápticos w_{nj}. Es importante notar que el primer subíndice corresponde a la neurona receptora, mientras que el segundo subíndice corresponde a la neurona emisora. Si $w_{nj}>0$, entonces la conexión es excitadora; así mismo, si $w_{nj}<0$, la conexión es inhibidora.

b) Sumador ($\sum$). Suma los componentes de las señales de entrada multiplicada por w_{nj}.

c) Función de activación (φ). Transformación no lineal.

d) Umbral o polarización(θ_i). Desplaza la entrada.

II.3.1 Modelo matemático.

En términos matemáticos, es posible describir la neurona n como se muestra en la Figura 3, por el siguiente par de ecuaciones:

$$u_n = \sum_{j=1}^{m} w_{nj} x_j \tag{1}$$

y

$$y_n = \varphi(u_n + b_n) \tag{2}$$

donde $x_1, x_2, \ldots, x_m$ son señales de entrada $w_{n1}, w_{n2} \ldots, w_{nm}$ son los pesos sinápticos de la neurona n; u_n es la combinación lineal de las entradas ponderadas por los pesos sinápticos; b_n es la polarización o umbral; $\varphi(\bullet)$ es la función de activación; y, finalmente, y_n es la señal de salida de la neurona n.

La polarización es un parámetro externo de la neurona n, pero es posible considerarla como parte de las señales de entrada, de tal forma que si se combinan las ecuaciones 1 y 2 se tiene:

$$v_n = \sum_{j=0}^{m} w_{nj} x_j \tag{3}$$

y

$$y_n = \varphi(v_n) \tag{4}$$

a v_n se le denomina potencial de activación. En la ecuación 3 se ha agregado una nueva sinapsis. Su entrada es:

$$x_0 = +1$$

y el peso correspondiente es:

$$w_{n0} = b_n$$

De tal forma que el modelo neuronal presentado como se muestra en la Figura 4. Estos modelos tienen una apariencia diferente, pero matemáticamente son iguales. Hasta este punto solo se ha nombrado a la función de activación; sin embargo, no se ha definido formalmente. A continuación se analizaran algunas de las funciones de activación más utilizadas en redes neuronales.

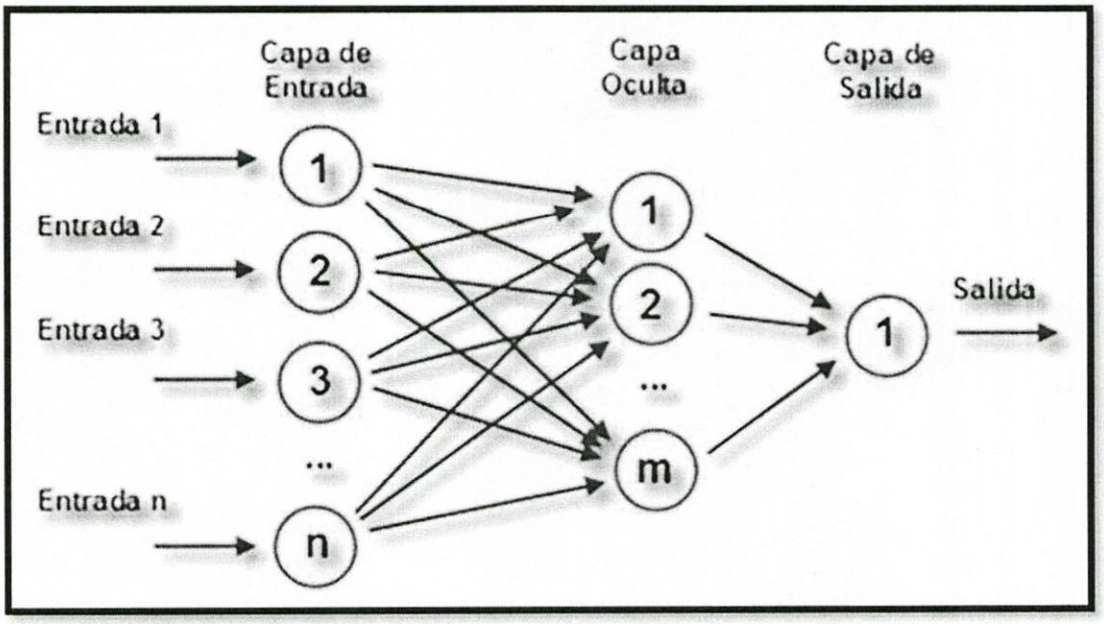

Figura 4. Modelo neuronal.

II.4 Tipos de funciones de activación.

Las funciones de activación, denotadas por $\varphi(v)$, definen la salida de la neurona en función del potencial de activación v. Se incluyen tres de los tipos básicos de funciones de activación:

1. Función escalón o umbral. Para este tipo de función de activación, como se muestra en la Figura 5a, se tiene:

$$\varphi(v) = \begin{cases} 1 \text{ si } v \geq 0 \\ 0 \text{ si } v < 0 \end{cases} \tag{5}$$

Correspondientemente, la salida de la neurona n empleando esta función de activación queda expresada como:

$$y_n = \begin{cases} 1 \text{ si } v_n \geq 0 \\ 0 \text{ si } v_n < 0 \end{cases} \tag{6}$$

Con v_n dada por la ecuación 3.

2. Función lineal a tramos. Para este tipo de función, presentada en la Figura 5b, se tiene:

$$\varphi(v) = \begin{cases} 1 & v \geq +\frac{1}{2} \\ v & +\frac{1}{2} > v > -\frac{1}{2} \\ 0 & v \leq -\frac{1}{2} \end{cases} \tag{7}$$

donde el factor de amplificación dentro de la región de operación se supone igual a la unidad.

3. Función sigmoidal. Esta es la función más comúnmente utilizada en RNA. Es estrictamente creciente, con un comportamiento asintótico. Un ejemplo de la función sigmoidal es la función logística:

$$\varphi(v) = \frac{1}{1+e^{-av}} \tag{8}$$

donde a es el parámetro que determina la pendiente de la función sigmoidal como se muestra en la Figura 5c.

Para la forma correspondiente de la función sigmoidal se puede usar la función tangente hiperbólica, definida por:

$$\varphi(v) = \tanh(v) \tag{9}$$

Las funciones de activación hasta aquí descritas toman valores en el intervalo cerrado $[0,1]$. Sin embargo, también se puede permitir que estos tomen los valores en el intervalo cerrado $[-1,1]$; en ese caso la función signo queda definida por:

$$\varphi(v) = \begin{cases} 1 & si\ v > 0 \\ 0 & si\ \ v = 0 \\ -1 & si\ v < 0 \end{cases} \tag{10}$$

Y su gráfica se muestra en la Figura 6.

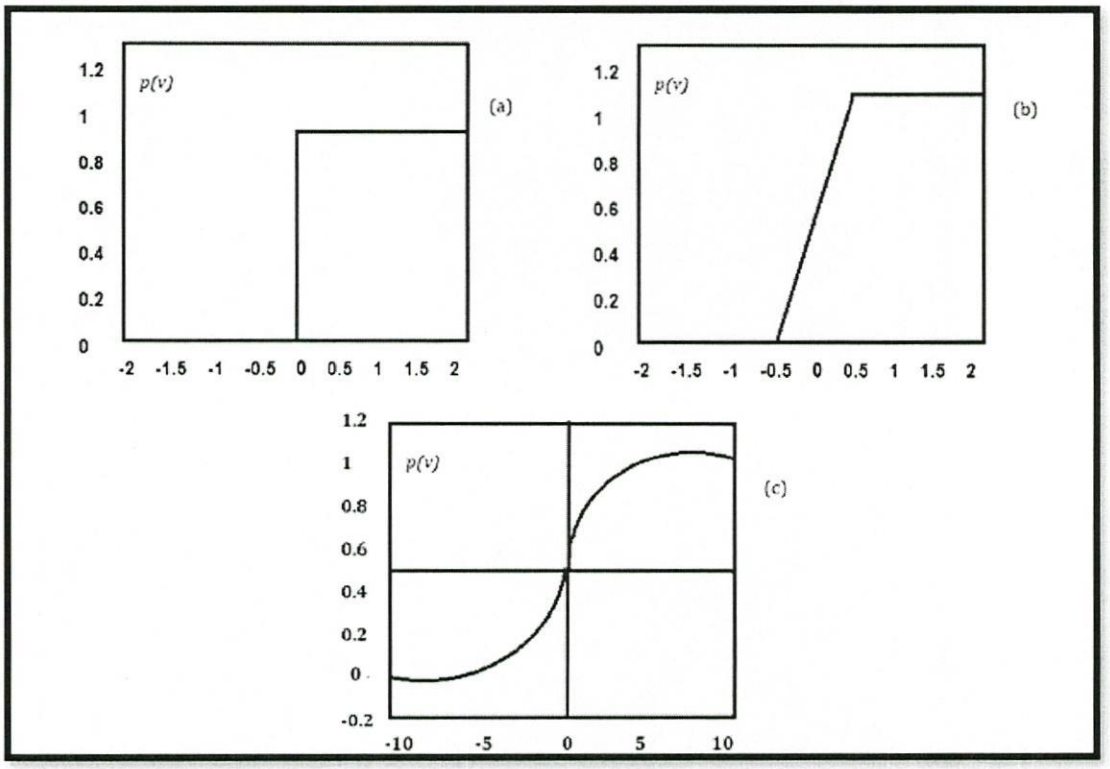

Figura 5. Funciones de activación: (a) Función escalón. (b) Función lineal a tramos. (c) Función sigmoidal.

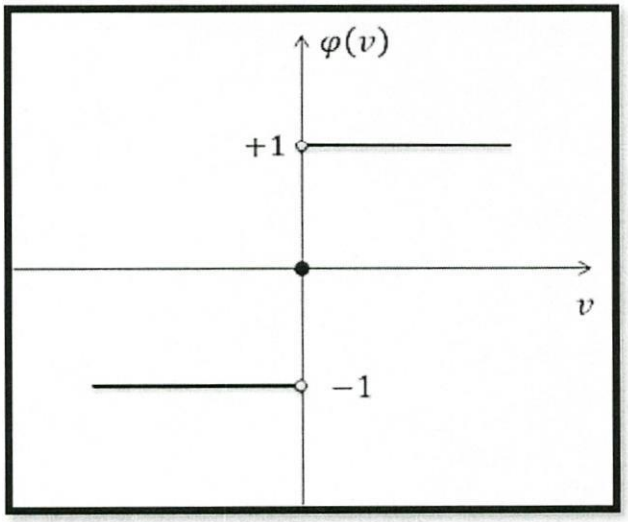

Figura 6. Función signo.

II.5 Grafos orientados aplicados a redes neuronales.

Gráfica de flujo de señal (Mason, 1953, 1956). Es una red de enlaces orientados, interconectados en puntos llamados nodos. Un nodo j tiene asociada una señal de nodo (x_j). Los enlaces orientados se inician en un nodo j y terminan en un nodo n y tienen asociada una función de transferencia o transmitancia T_{nj}. que especifica como x_n depende de x_j. Como se muestra en la Figura 7.

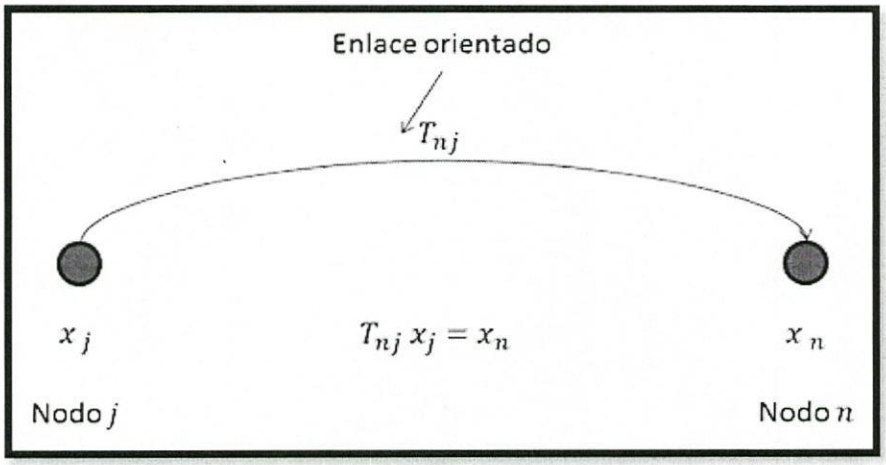

Figura 7. Componentes de una gráfica de flujo de señal.

II.5.1 Extensión a redes neuronales.

Los flujos de señales en las diferentes partes del gráfico se rigen por tres reglas básicas:

Regla 1. Una señal fluye en el sentido de la flecha. Existen dos tipos de enlaces:

a) Sinápticos. Relación lineal $y_n = w_{nj}x_j$. como se muestra en la Figura 8a.
b) De activación. Relación no lineal $y_n = \varphi(x_j)$ como se muestra en la Figura 8b.

Regla 2. Una señal de nodo es igual a la suma algebraica de las señales de los enlaces de entrada (convergencia sináptica *fan-in*). La regla de la convergencia sináptica se muestra en la Figura 8d.

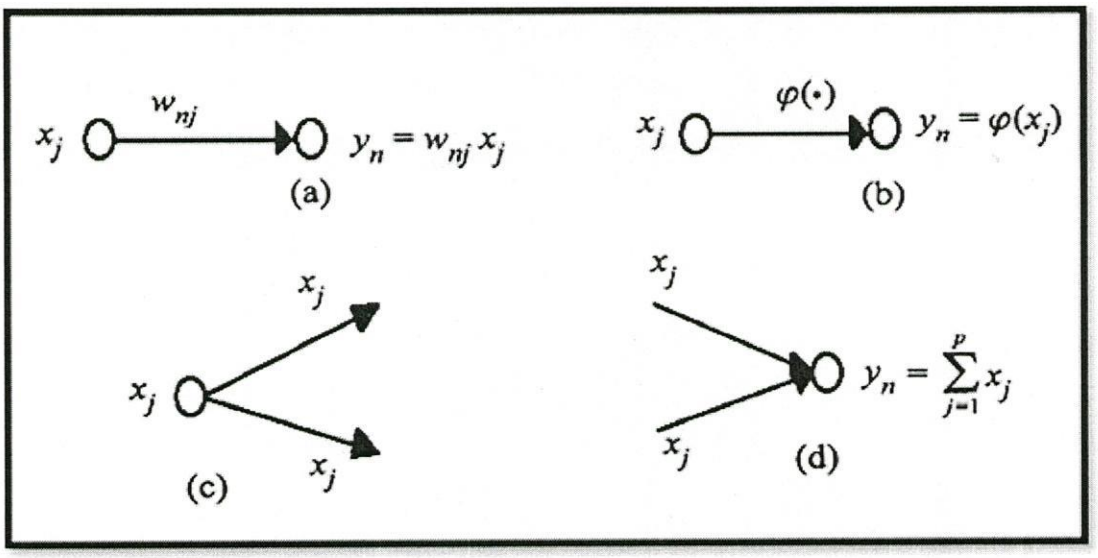

Figura 8. Ilustración de las reglas básicas para la construcción de grafos dirigidos.

Regla 3. La señal de un nodo trasmitida por todos los enlaces de salida (divergencia sináptica fan-out) como se muestra en la Figura 8c.

Los esquemas que se muestran en las Figuras 3 y 4 respectivamente no son la mejor forma de representar gráficamente una neurona. Una representación más útil es la que se muestra en la Figura 9, que se basa en la teoría de grafos orientados presentada en esta sección. También se le llama gráfica incompleta ya que solo representa el flujo de información inter-neuronal (Haykin, 1999).

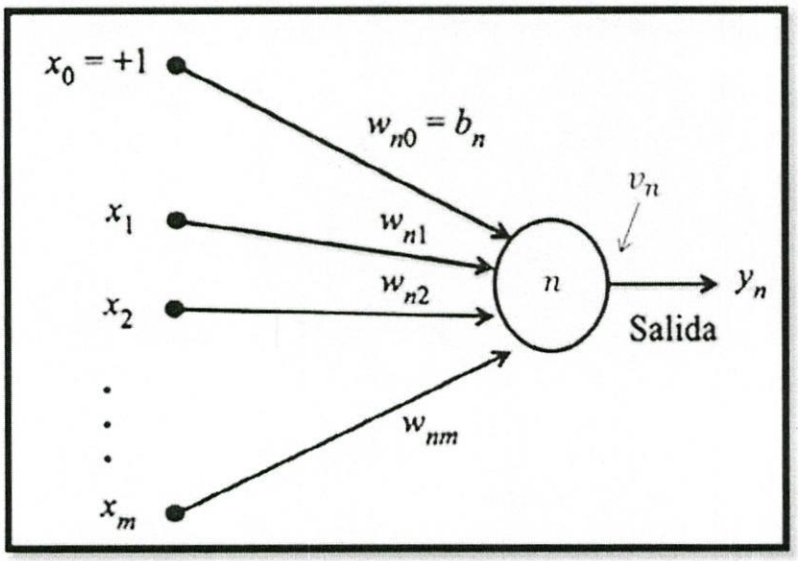

Figura 9. Grafo orientado de una neurona.

Propiedades:

- Cada neurona es representada por un conjunto de enlaces sinápticos, un enlace de polarización y un enlace de activación.
- Los enlaces sinápticos ponderan la señal de entrada, multiplicándola por los pesos sinápticos.
- La suma de las señales pesadas define el potencial de activación de la neurona (v_n).
- El enlace de activación transforma el potencial de activación en la señal de salida (y_n).

II.6 Retroalimentación.

En un sistema existe retroalimentación si al menos una de sus salidas influye en al menos una de sus entradas, creándose un lazo cerrado. Este es un concepto importante en las redes neuronales recurrentes (Haykin, 1999).

Ejemplo 1. En la Figura 10, se muestra el grafo orientado de un sistema con un lazo de retroalimentación, donde la señal de entrada $x_j\ (k)$, la señal interna $x'_j\ (k)$ y la señal de salida $y_{nj}\ (k)$ son funciones de la variable discreta en el instante de muestreo k.

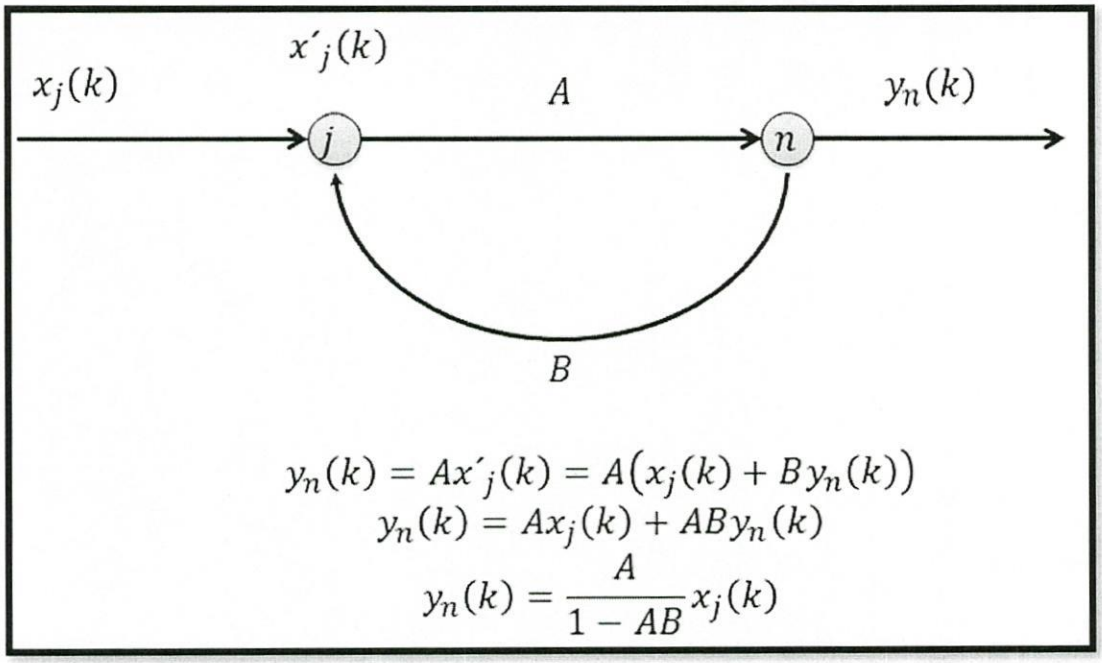

Figura 10. Grafo orientado de un sistema con lazo de retroalimentación.

El comportamiento dinámico del sistema está determinado por el peso w. En particular se distinguen dos casos específicos:

Si $|w|<1$, entonces la señal de salida $y_n\ (k)$ es exponencialmente convergente, por lo cual el sistema es estable como se muestra en la Figura 11a;

Si $|w| \geq 1$, entonces la señal de salida $y_n(k)$ es divergente (inestable). Si $|w| = 1$, la divergencia es lineal como se muestra en la Figura 11b; pero, si $|w|>1$, la divergencia es exponencial como se muestra en la Figura 11c.

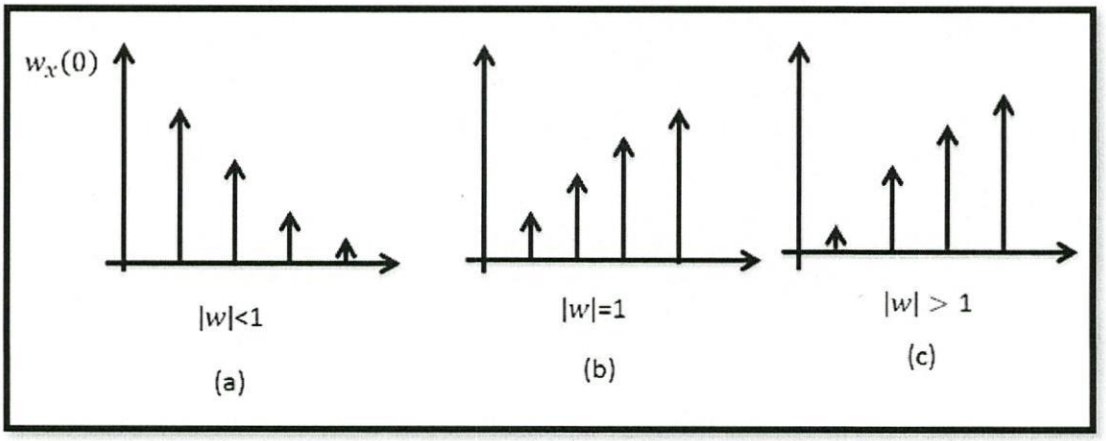

Figura 11. Respuesta en el tiempo del sistema: (a) Estable. (b) Divergencia lineal. (c) Divergencia exponencial.

II.7 Arquitecturas neuronales.

En general, es posible identificar tres clases diferentes de arquitecturas neuronales. Estas se describen a continuación:

II.7.1 Redes neuronales unicapa.

Generalmente, en las redes neuronales, las neuronas están organizadas por capas. En el caso más simple, una red neuronal unicapa, la capa de entrada se conecta directamente a la capa de neuronas de salida por medio de la sinapsis. Como se muestra en la Figura 12a.

II.7.2 Redes neuronales multicapa.

Las redes neuronales multicapa se distinguen por tener una o más capas de neuronas ocultas, cuyos nodos computacionales se denominan neuronas ocultas o unidades ocultas. La red neuronal se muestra en la Figura 12b, se dice totalmente conectada, en el sentido de que todos los nodos en cada capa de la

red están conectados a todos los otros nodos en la siguiente capa. En el caso de que falte alguna de las sinapsis de la red, entonces se dice que la red es parcialmente conectada.

II.7.3 Redes neuronales recurrentes.

Las redes neuronales recurrentes se distinguen de las anteriores en que por lo menos tienen un lazo de retroalimentación. Por ejemplo, una red recurrente puede consistir en una sola capa oculta con cada una de sus neuronas retroalimentando las señales de salida a la entrada de otras neuronas, como se muestra en la Figura 12c, donde se introduce el operador de retardo unitario q^{-1} en el lazo de retroalimentación. En la estructura mostrada no existen lazos de retroalimentación en la red. La red neuronal que se muestra en la Figura 12c, no tiene neuronas ocultas. En la Figura 12d se muestra otra clase de red neuronal recurrente con neuronas ocultas.

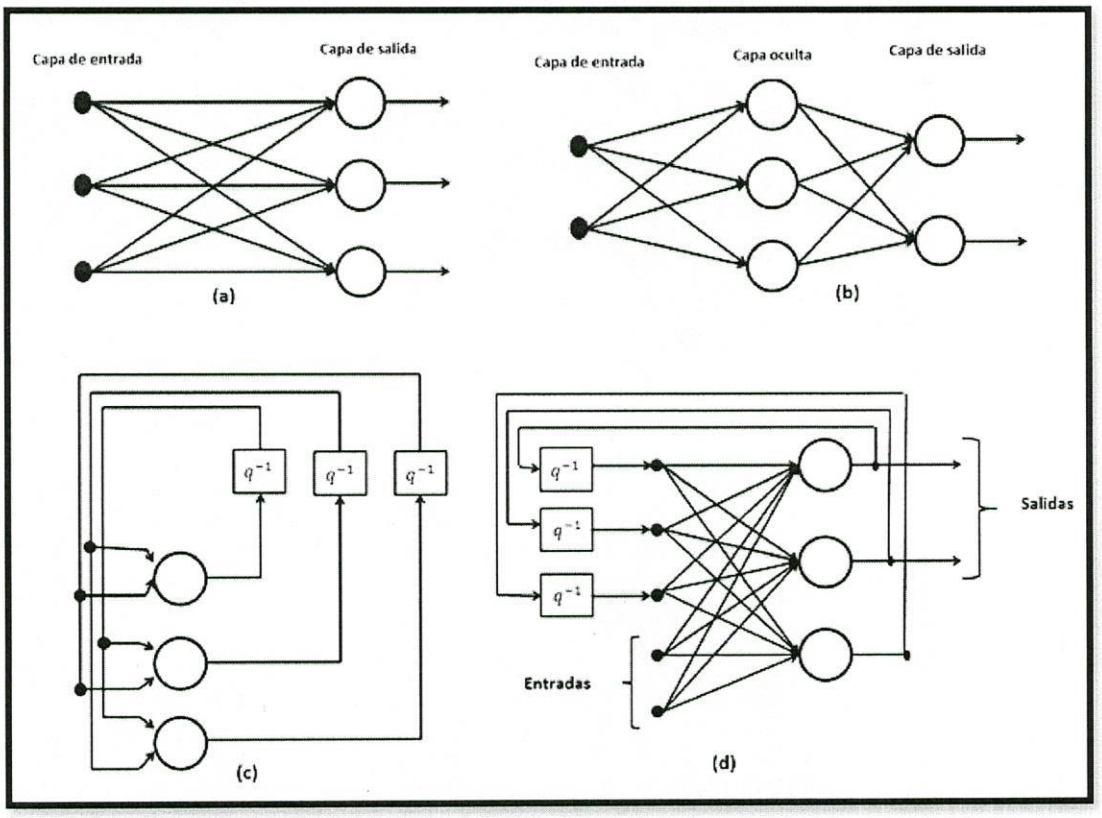

Figura 12. Estructuras neuronales: (a) Red neuronal unicapa. (b) Red neuronal multicapa. (c) Red neuronal recurrente sin auto lazos. (d) Red neuronal recurrente con neuronas ocultas.

II.7.4 Representación del conocimiento en redes neuronales.

El conocimiento se refiere a:

- La información acumulada.
- Los modelos y procedimientos, utilizados por seres humanos o máquinas para interpretar, predecir y responder apropiadamente a estímulos del medio ambiente.

La Figura 13 muestra esquemáticamente estos componentes.

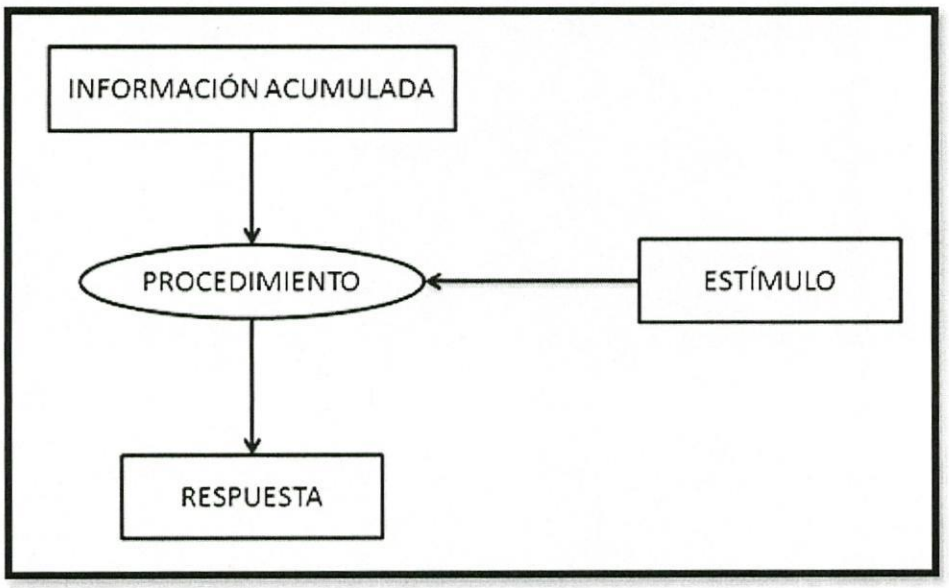

Figura 13. Representación del conocimiento.

En la relación o la representación del conocimiento se considera la información que contiene y como está codificada. La red neuronal construye un modelo de la interacción con el medio ambiente, por medio del aprendizaje de relaciones de entrada-salida. Este modelo debe ser compatible con el mundo real. La información puede ser adquirida por conocimiento a priori o con mediciones obtenidas por sensores. Usualmente la utilización de la red neuronal se divide en:

a) Aprendizaje por ejemplos (entrenamiento).
b) Aplicación.

II.8 Diseño de una red neuronal.

Éste consiste en:

- Definir tipo de red y su estructura.
- Dimensión de la entrada y de la salida (proceso heurístico).
- Entrenamiento de la red.

- Presentación de los ejemplos.
- Aplicación (generalización).

Existe una gran diferencia en relación con otras formas de adquisición de conocimiento. La red aprende directamente de ejemplos, pero no codifica el conocimiento en un modelo específico. Por lo contrario, este está distribuido en la estructura de la red. Sin embargo, la red debe cumplir las cuatro reglas básicas de representación del conocimiento:

Regla 1. Entradas similares, pertenecientes a clases similares, deben producir dentro de la red representaciones similares y ser clasificadas como pertenecientes a la misma categoría.

Entrada:

$$x = \begin{pmatrix} x_1 \\ \cdot \\ \cdot \\ x_p \end{pmatrix} \tag{11}$$

Distancia euclidiana:

$$d_{xy} = \| x - y \|_2 = \sqrt{\sum_{i=1}^{p} (x_i - y_i)^2} \tag{12}$$

Similitud = $1/d_{xy}$

Similitud usando producto interno.

Si $// x - y // \to 0$, entonces $x \to y$. Esto implica mayor similitud lo que supone mayor producto interno. Si la similitud es alta entonces pertenece a la misma categoría.

Regla 2. Ejemplos a categorizarse separadamente deben tener una representación muy diferente en la red neuronal, como se muestra en la Figura 14.

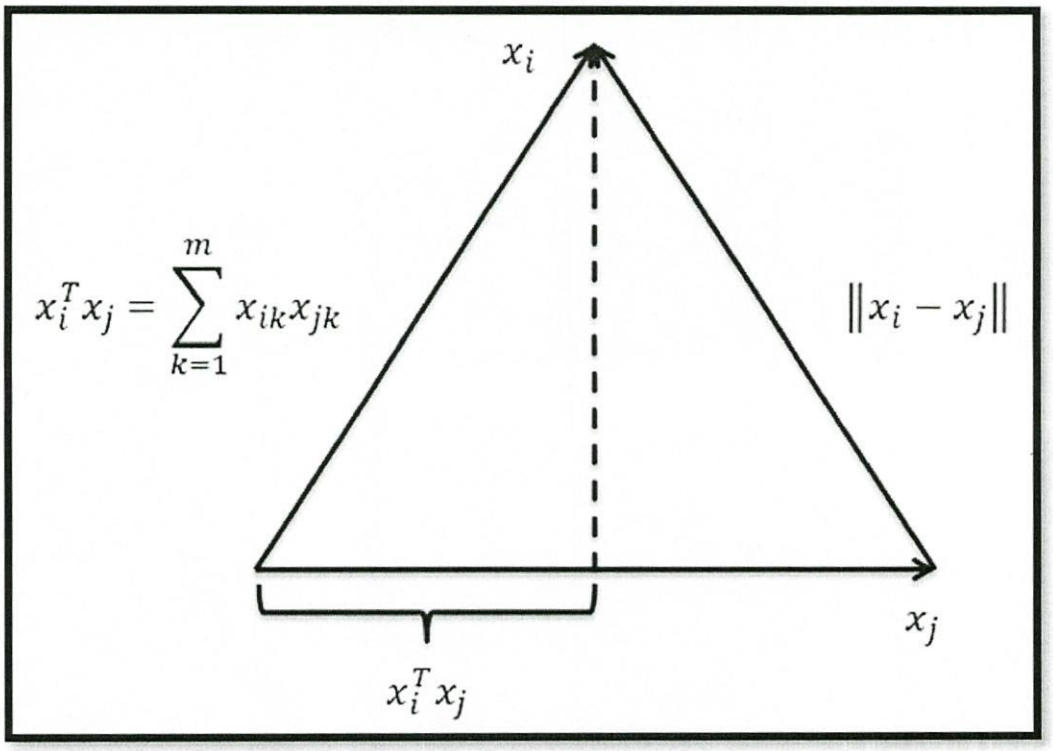

Figura 14. Relación entre el producto y la distancia Euclidiana.

Regla 3. Si una característica es importante, debe existir un gran número de neuronas dedicadas a su representación.

Regla 4. Información a priori e invariancias deben incorporarse a la estructura de la red. La regla 4 implica la presencia de una estructura especializada.

Las ventajas de incluir información a priori son:

- Evidencia biológica.
- Menos parámetros a ajustar, lo que implica reducción de la complejidad (número de ejemplos y tiempo) del aprendizaje.
- Aceleración del proceso de la información.
- Reducción de los costos de implementación.

La información a priori se incorpora de la forma siguiente:

- Seleccionando la estructura de la red con conexiones locales (parcialmente conectadas).
- Restringiendo la selección de pesos.
- Por procesamiento para extraer características.

Invariancia: determinar las características que no varían o varían pero en las diferentes representaciones.

La invariancia se puede incorporar por medio de:

1) **La estructura.** Seleccionar la estructura de la red de tal forma que diferentes representaciones del mismo objeto den la misma salida.

   Ejemplo: Clasificación de imágenes invariantes a rotación.

   w_{ji}= Peso sináptico que conecta el pixel 'i' con la neurona 'j'.

   Entonces, si $w_{jk} = w_{jk}$ para todos los pixeles tales que la distancia al centro sea la misma, se obtiene la invariancia a la rotación.

   Desventajas: duplicación de pesos e incremento excesivo del número de pesos.

2) **Por entrenamiento.** Se utilizan pares de entrenamiento con diferentes representaciones.

   Desventajas: no se garantiza la clasificación de diferentes objetos con diferentes representaciones además de alta complejidad de entrenamiento.

3) **Características invariantes.** Estas se obtienen por procesamiento.

Ventajas: el número de características invariantes es reducido a un valor suficiente; el diseño del clasificador neuronal se simplifica; la invariancia se garantiza para todos los ejemplos, como se muestra en la Figura 15.

Desventaja: requiere información a priori; desafortunadamente no existe una teoría bien desarrollada para optimizar la estructura de una red neuronal.

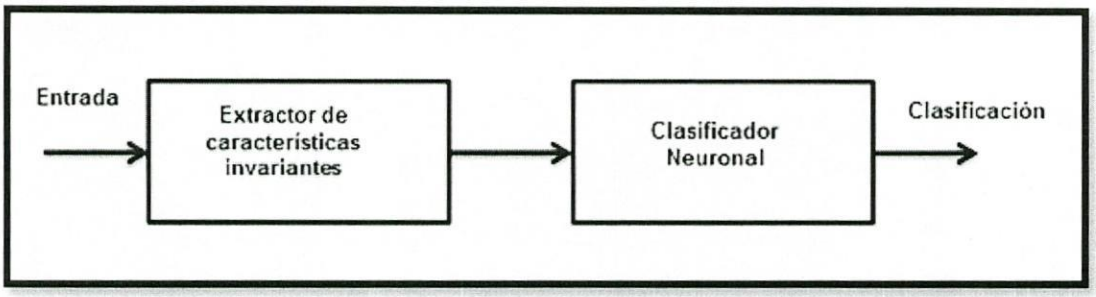

Figura 15. Diagrama a bloques de un sistema del tipo de espacio de características invariantes.

II.9 Proceso de aprendizaje.

Desde el punto de vista de aplicaciones en ingeniería, las características más importantes de las redes neuronales son: a) aprendizaje por interacción con el medio ambiente y b) mejoramiento de su desempeño por medio de este aprendizaje.

¿Cómo aprende?

La red neuronal aprende variando sus parámetros, en particular los pesos sinápticos y el umbral o polarización. El aprendizaje es un proceso por el cual los parámetros se adaptan, por la interacción continua con el medio ambiente (Haykin, 1999). El tipo de aprendizaje está determinado por la forma en que se realiza dicha adaptación. Este proceso implica la siguiente secuencia de eventos:

- La red neuronal es estimulada por el medio ambiente.

- La red neuronal ajusta sus parámetros.
- La red neuronal genera una nueva respuesta.

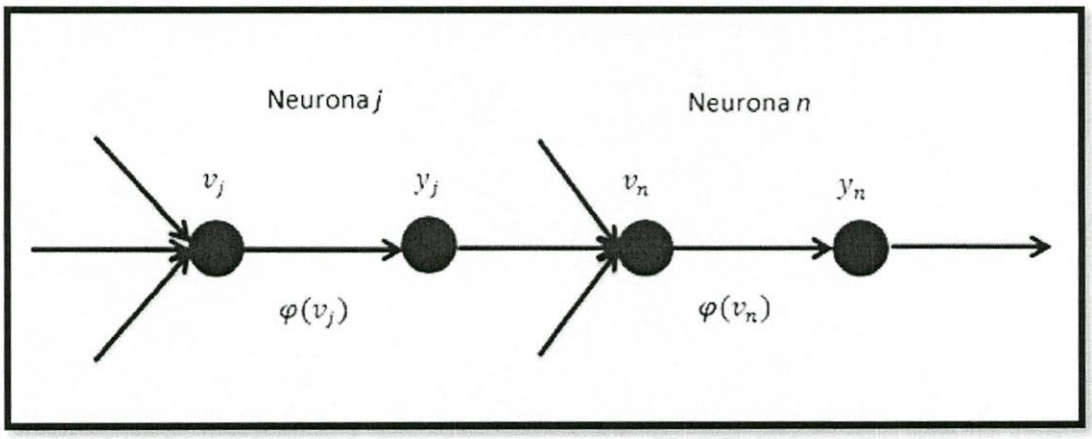

Figura 16. Conexión de dos neuronas.

Mecanismo de ajuste. Los pesos de la red neuronal de la Figura 16 se ajustan por el siguiente mecanismo:

$$w_{nj}\,(k+1) = w_{nj}(k) + \Delta w_{nj}\,(k) \tag{13}$$

Donde k es la iteración respectiva.

Algoritmo de aprendizaje. Es el conjunto de reglas para resolver el problema de aprendizaje. Difieren en como determinar $\Delta w_{nj}(k)$. Los algoritmos de aprendizaje más importantes son:

- Corrección del error.
- Boltzman.
- Ley de efectos de Thorndike.
- Regla de Hebb (delta).
- Competitivo.

Paradigmas de aprendizaje. Es la forma como la red neuronal interactúa con el medio ambiente. Los principales paradigmas de aprendizaje son:

- paradigmas de aprendizaje supervisado.
- aprendizaje por reforzamiento.
- aprendizaje no supervisado.

II. 9.1 Algoritmos de aprendizaje

II. 9.1.1 Corrección del error.

Considérese la $n - ésima$ neurona en la iteración k; $d_n(k)$ es la repuesta deseada para la neurona n; y_n es la respuesta de esta neurona; $x(k)$ es el vector estímulo del ambiente, y $\{x(k), d_n(k)\}$ es el par de entrenamiento, como se muestra en la Figura 17.

La presencia de un ambiente aleatorio implica la necesidad de utilizar una descripción probabilística. Considérese la siguiente señal de error:

$$e_n(k) = d_n(k) - y_n(k) \tag{14}$$

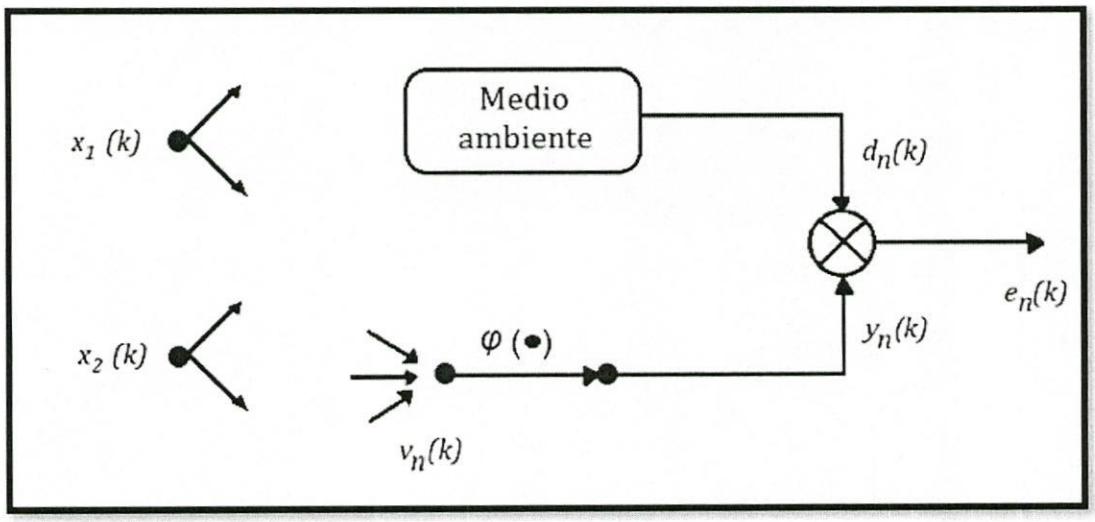

Figura 17. Ilustración de la regla de corrección del error.

El objetivo es minimizar la función de costo (criterio), que considera este error (Haykin, 1999). Una vez seleccionado el criterio, el problema de aprendizaje por corrección de error se convierte en uno de optimización.

Criterio de mínimos cuadrados.

$$J = E\left\{\frac{1}{2}\sum_{n=0}^{I} e_n^2\ (k)\right\} \tag{15}$$

Donde E es el operador de esperanza estadística o valor esperado. Este es un criterio ampliamente utilizado. Para resolverlo es muy común utilizar el gradiente descendiente como se muestra en la Figura 18, puesto que el descenso más rápido hacia el mínimo es colineal al gradiente.

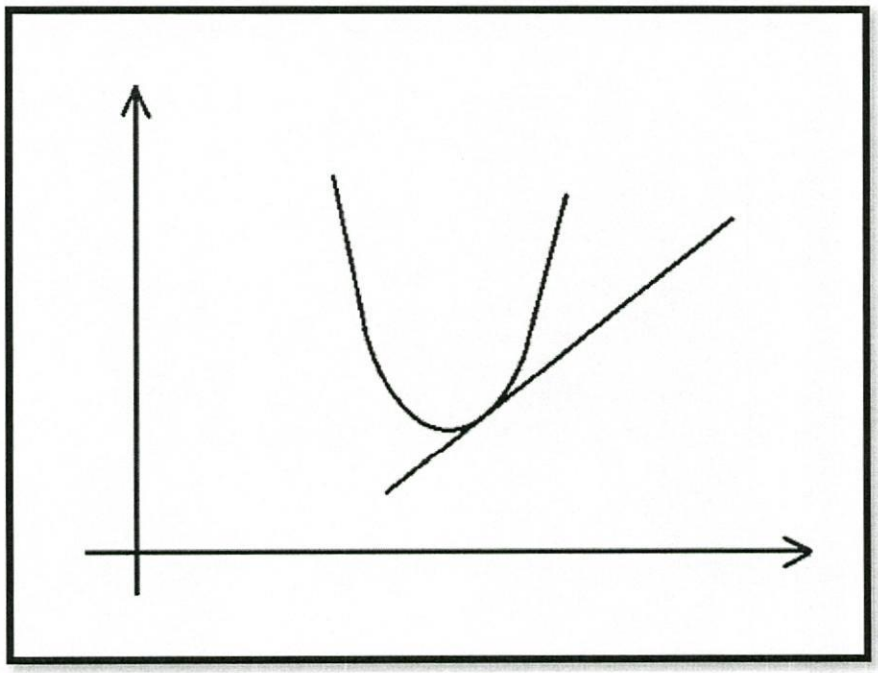

Figura 18. Método del gradiente descendiente.

Para minimizar el criterio de la ecuación 15 se debe conocer la caracterización probabilística. Una solución aproximada es tomar como criterio el valor instantáneo del error.

$$J(k) = \frac{1}{2}\sum_{n=1}^{I} e_n^2(k) \tag{16}$$

La regla de aprendizaje de los pesos es:

$$\Delta w_{nj}(k) = \eta e_n(k) x_j(k) \tag{17}$$

donde $\eta > 0$ es el factor de aprendizaje, y debe seleccionarse cuidadosamente para evitar inestabilidades. Seleccionar η pequeño implica aprendizaje suave pero lento, mientras que la selección de η grande implica mayor velocidad de aprendizaje pero riesgo de inestabilidad (Widrow et al, 1960).

II.9.1.2 Regla de Hebb.

Fue el primer postulado de cómo aprenden las neuronas (Hebb, 1949): cuando el axón de una neurona *A* esta lo suficientemente cerca de una neurona *B*, y repetida y persistentemente ayuda a su disparo, se produce un proceso de crecimiento o cambio de metabolismo en una o ambas neuronas, de tal manera que la eficiencia de *A*, como una de las neuronas que disparan a *B*, se incrementa como se muestra en la Figura 19.

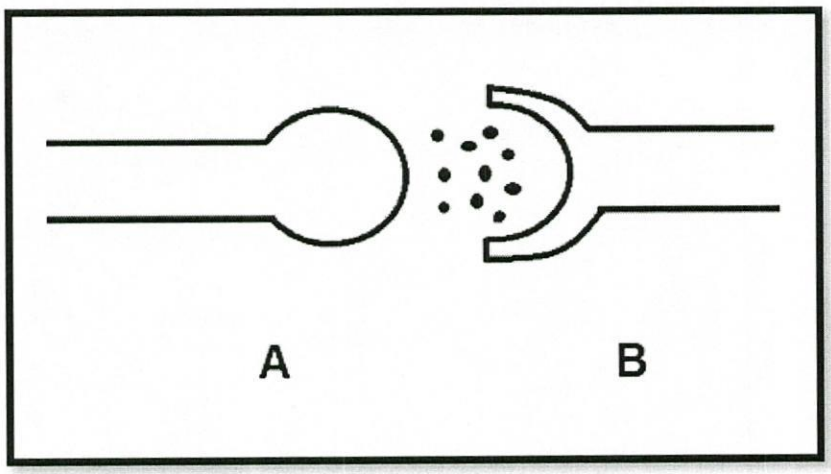

Figura 19. Regla de Hebb.

La regla de Hebb se puede replantear en los siguientes términos:

Si *A* y *B* son activadas simultáneamente (sincrónicamente), el peso sináptico que las une se incrementa.

Si son activadas asincrónicamente, entonces este peso se decrementa o desaparece.

Definición formal.

La regla de Hebb es un mecanismo variante en el tiempo, local y altamente interactivo, para incrementar la eficiencia de la sinapsis como una función de la correlación entre las actividades pre y postsináptica.

Modelo matemático.

Para formular la regla de aprendizaje de Hebb en términos matemáticos considérese un peso sináptico w_{nj} de la neurona n con señales pre y postsinápticas x_j y y_n respectivamente como se muestra en la Figura 20. El ajuste aplicado al peso sináptico w_{nj} en la iteración k es expresado en forma general por:

$$\Delta w_{nj}(k) = F(y_n(k), x_j(k)) \tag{18}$$

Donde F (•,•) es una función de las señales pre y postsináptica.

Regla de activación producto.

La expresión siguiente muestra la forma más simple de escribir la regla de aprendizaje de Hebb en términos matemáticos:

$$\Delta w_{nj}(k) = \eta y_n(k) x_j(k) \tag{19}$$

Donde η es una constante positiva que determina la razón de aprendizaje; esta expresión claramente pone de relieve la naturaleza correlacional de la sinapsis hebbiana, como se muestra en la Figura 20.

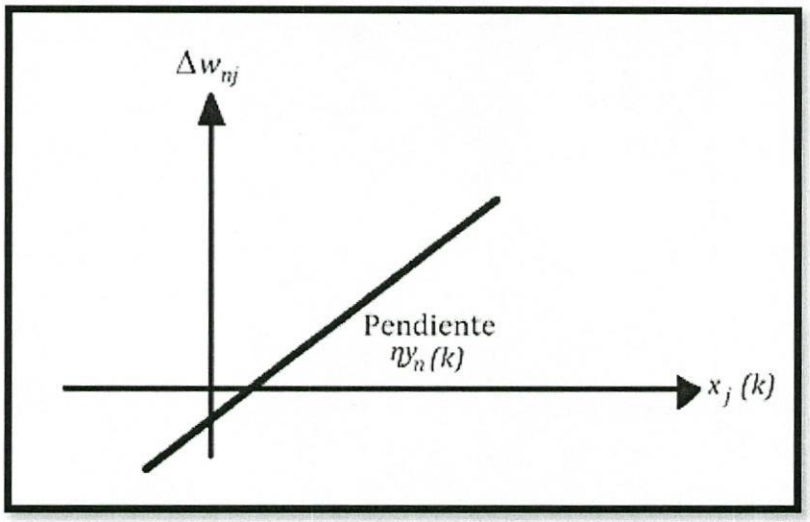

Figura 20. Incremento en los pesos.

Regla de activación producto generalizada.

Para eliminar el crecimiento sin límite de los pesos, se agrega un factor de olvido; por lo tanto, la descripción matemática de la regla de aprendizaje de Hebb queda expresada como:

$$\Delta w_{nj}(k) = \eta y_n(k)x_j(k) - \alpha y_n(k)w_{nj}(k)$$

$$\Delta w_{nj}(k) = \alpha y_n(k)\left[cx_j(k) - w_{nj}(k)\right]; c = \frac{\eta}{\alpha} \qquad (20)$$

$$sí\ x_j(k) < \frac{w_{nj}(k)}{c}\ entoces\ \Delta w_{nj} < 0.$$

II.9.2 Paradigmas de aprendizaje.

II.9.2.1 Paradigma de aprendizaje supervisado.

Conceptualmente, el maestro tiene conocimiento del medio ambiente, representado en un conjunto de ejemplos entrada-salida, mientras que el medio ambiente es desconocido para la red neuronal (Haykin, 1999). Los parámetros de la red son ajustados como función del vector de entrenamiento y de la señal de error tal como se muestra en la Figura 21.

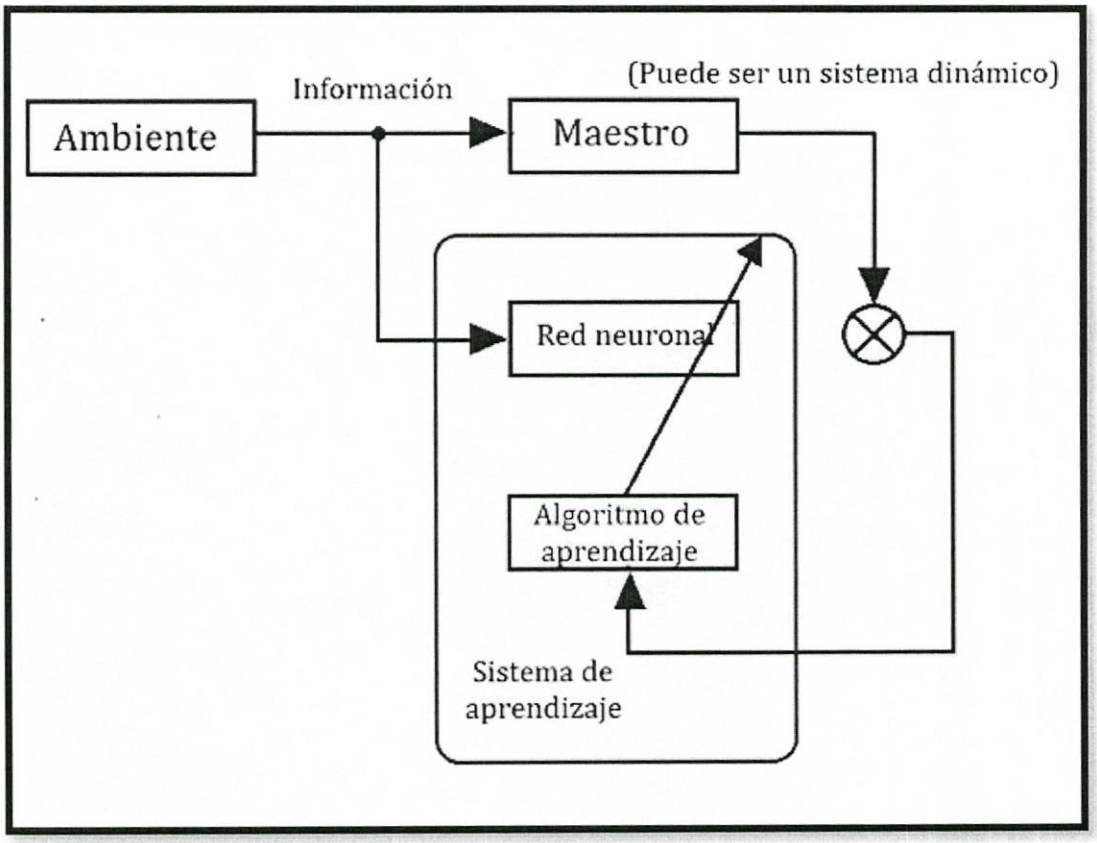

Figura 21. Esquema del paradigma de aprendizaje supervisado.

Se define J como el criterio de desempeño (mínimos cuadrados), el cual es una superficie multidimensional.

Los parámetros del sistema se desplazan hacia la solución mínima de J; tal desplazamiento puede hacerse por el método del gradiente.

Aplicaciones:

- Clasificación de patrones
- Aproximación de funciones
- Identificación (construcción de modelos de sistemas)

II.9.2.4 Paradigma de aprendizaje no supervisado.

Se define una medida independiente de las tareas, para medir la calidad de la representación que el sistema de aprendizaje debe aprender como se muestra en la Figura 22. Los parámetros se ajustan para optimizar esta medida. Este aprendizaje puede ser implementado por aprendizaje competitivo. Puede ser más rápido que el aprendizaje supervisado. La red neuronal desarrolla la habilidad para formar representaciones de las características de las entradas y, por lo tanto, crear nuevas clases automáticamente.

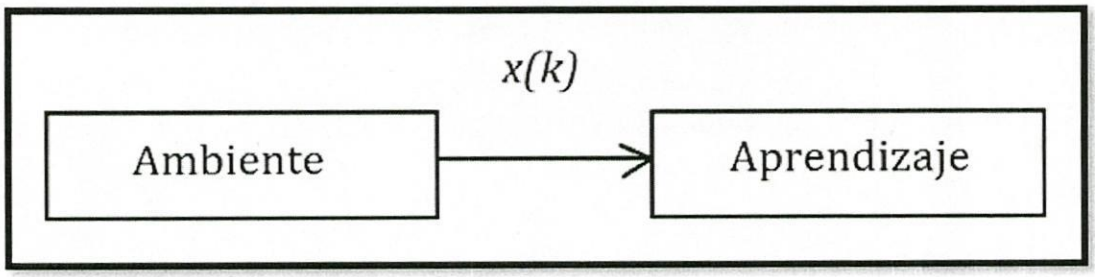

Figura 22. Estructura del paradigma de aprendizaje no supervisado.

II.10 El perceptrón.

El perceptrón es la forma más simple de una red neuronal utilizada para la clasificación de patrones linealmente separables (patrones que se localizan en los lados opuestos de un hiperplano, también conocido como dicotomía). Considérese

en una sola neurona con pesos ajustables y un umbral, como se muestra en la Figura 23 el algoritmo utilizado para ajustar los parámetros libres de esta red surgió como un procedimiento de aprendizaje de Rosenblatt (Rosenblat, 1958) para su modelo del cerebro, el perceptrón. La prueba de convergencia de dicho algoritmo es conocida como el teorema de convergencia del Perceptrón.

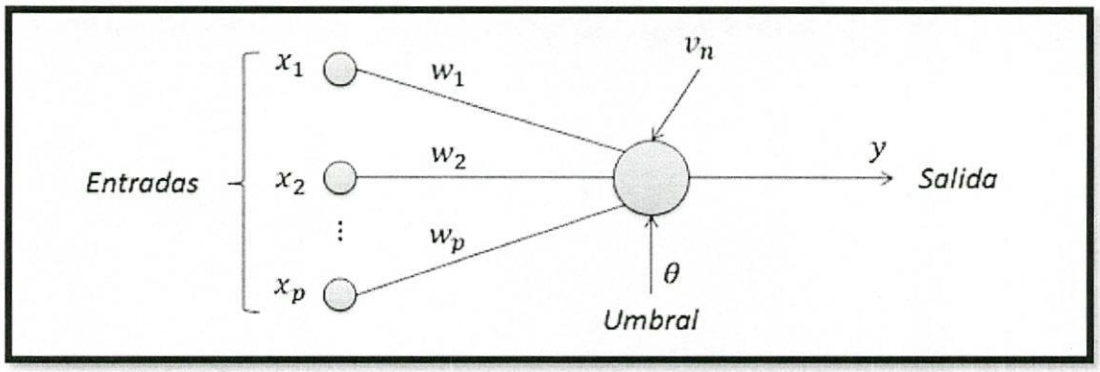

Figura 23. El perceptrón.

II.10.1 Consideraciones básicas.

Considerese el perceptrón que se muestra en la Figura 24 con una función de activación tipo signo como se muestra en la Figura 6; el modelo matemático está dado por las ecuaciones 21 y 22:

$$y = \varphi(v) = \begin{cases} 1 \; si \; v > 0 \\ 0 \; si \; v = 0 \\ -1 \; si \; v < 0 \end{cases} \tag{21}$$

con v definida como:

$$v = \sum_{i=1}^{P} w_i \; x_i - \theta \tag{22}$$

El objetivo es clasificar $y = 1$ si:

$$x = \begin{pmatrix} x_1 \\ x_2 \\ . \\ . \\ . \\ x_p \end{pmatrix} \in C_1$$

o $y = -1$ para:

$$x = \begin{pmatrix} x_1 \\ x_2 \\ . \\ . \\ . \\ x_p \end{pmatrix} \in C_2$$

La condición es que $C_1 \cap C_2 = \emptyset$ y además $C_1 \cap C_2$ sean separables por un hiperplano.

II.10.1.1 Algoritmo de aprendizaje.

Para deducir el algoritmo de aprendizaje por correción de error, resulta conveniente trabajar con el perceptrón que se muestra en la Figura 24, en el cual se considerará al umbral como un peso sináptico conectado a una entrada fija igual a + 1. Se define el vector de entradas por:

$$x(k)^T = \left(+1, x_1(k), x_2(k), \dots, x_p(k)\right) \tag{23}$$

donde p es el número de entradas y $x(k) \in \Re^{(p+1)}$. Así mismo, se define el vector de pesos como:

$$w(k)^T = \left(\theta, w_1(k), w_2(k), \dots, w_p(k)\right) \tag{24}$$

donde k denota la iteración tal que $k = 1, \ldots, N$, siendo N el número de pares de entrenamiento donde $d(k)$ es la salida deseada en la iteración k. Una vez definido esto, el potencial de activación para el perceptrón como se muestra en la Figura 24, queda determinado por la acuación 25 y la salida respectiva definida por:

$$v(k) = w(k)^T x(k) \tag{25}$$

$$y(k) = \varphi\big(v(k)\big) = sign\big(v(k)\big) \tag{26}$$

En la Figura 24 se muestra el hiperplano para un problema de clasificación de patrones bidimensional. Así mismo, en la Figura 25a se muestra un par de patrones linealmente separables, mientras que en la Figura 25b se muestra un par de patrones que no son linealmente separables.

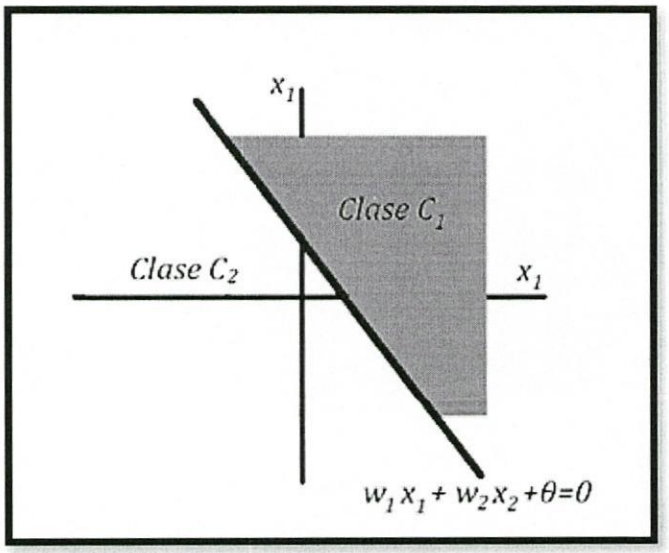

Figura 24. Hiperplano separador.

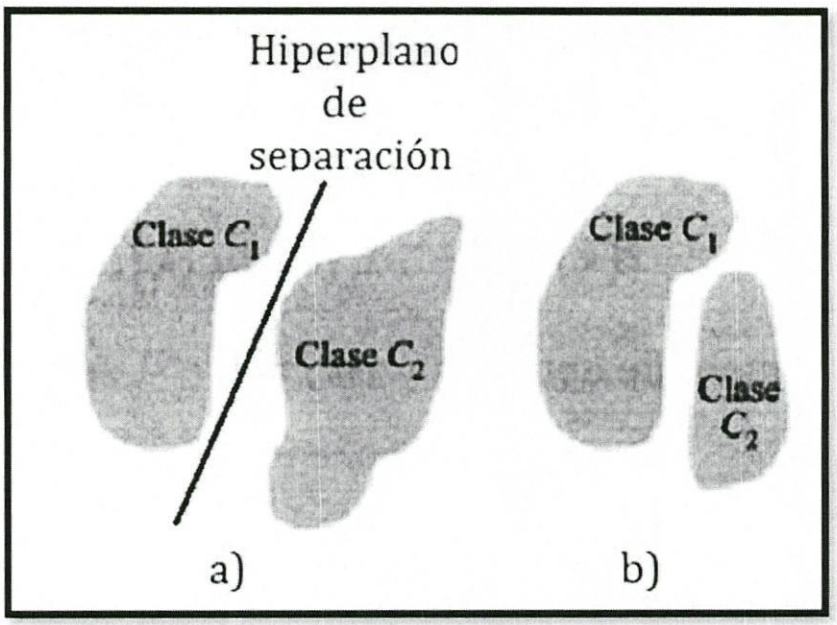

Figura 25. a) Patrones linealmente separables. b) Patrones linealmente no separables.

Supongase que las variables de entrada al perceptrón están originadas por dos clases linealmente separables. Sea x_1 el subconjunto de vectores de entrenamiento $x_1(1), x_1(2), \ldots$que pertenecen a C_1, y sea x_2 el subconjunto de vectores de entrenamiento $x_2(1), x_2(2), \ldots$ que pertenecen a C_2; la unión de x_1 y x_2 es el conjunto de entrenamiento completo. Dado los dos conjuntos de vectores x_1 y x_2 para el entrenamiento del clasificador, el proceso de entrenamiento involucra el ajuste del vector de pesos w de tal modo que las dos clases C_1 y C_2 sean separables por un hiperplano. Entonces existe un vector de pesos w tal que:

$$w^T x > 0 \; para \; cualquier \; vector \; de \; entrada \; x_1 \; que \; pertenezca \; a \; la \; clase \; C_1$$

$$w^T x \leq 0 \; parta \; cualquier \; vector \; de \; entrada \; x_2 \; que \; pertenezca \; a \; la \; clase \; C_2$$

En la expresión anterior arbitrariamente se especifica que si una entrada x que produce $w^T x = 0$ entonces dicha entrada pertenece a la clase C_2; sin embargo se elige de manera distinta el algoritmo también trabaja adecuadamente.

Para la adaptación de los pesos sinápticos del perceptrón se puede utilizar la regla de corrección de error propuesta por Rosenblantt (Rosenblatt, 1958) como sigue:

Sí $$w^T x \geq 0, \quad \forall x \in C_1$$

o sí $$w^T x < 0, \quad \forall x \in C_2$$

$$w(k+1) = w(k) - \eta(k)x(k) \text{ si } w^T x \geq 0 \text{ y } x \in C_2 \tag{27}$$

$$w(k+1) = w(k) + \eta(k)x(k) \text{ si } w^T x < 0 \text{ y } x \in C_1$$

donde $\eta(k)$ es el factor de aprendizaje; C_1 y C_2 son clases linealmente separables. Considerando la salida del perceptrón y la salida deseada, es posible definir el error como:

$$e(k) = d(k) - y(k) \tag{28}$$

Con:

$$d(k) = \begin{cases} +1 \; x(k) \epsilon \; C_1 \\ -1 \; x(k) \epsilon \; C_2 \end{cases} \tag{29}$$

Donde $d(k)$ es la salida deseada y $x(k)$ es el vector de entradas. Entonces la ley de adaptación de pesos queda como:

$$w(k+1) = w(k) + \eta \; e(k)x(k) \tag{30}$$

II.10.2 Problema de optimización.

Considérese una función $\varepsilon(w)$, la cual es una función continuamente diferenciable de algún vector de pesos (parámetros) w. La función $\varepsilon(w)$ transforma los elementos de w en números reales. Se desea encontrar una solución óptima w^* que satisfaga la condición:

$$\varepsilon(w^*) \leq \varepsilon(w) \tag{31}$$

Entonces es necesario resolver un problema de optimización sin restricciones, planteado como sigue:

"Minimizar la función de costo $e(w)$ con respecto al vector de pesos w"

La condición necesaria para optimalidad es:

$$\nabla\varepsilon(w^*) = 0 \tag{32}$$

Donde ∇ es el operador gradiente definido como:

$$\nabla = \left[\frac{\partial}{\partial w_1}, \frac{\partial}{\partial w_2}, \ldots, \frac{\partial}{\partial w_m}\right]^T \tag{33}$$

Una clase de algoritmos de optimización sin restricciones particularmente adecuado está basado en la idea de descendencia iterativa. Iniciando con una condición inicial $w(0)$, se genera una secuencia $w(1), w(2), \ldots$, tal que la función de costo $e(w)$ disminuye en cada iteración del algoritmo como sigue:

$$e\big(w(k+1)\big) < e\big(w(k)\big) \tag{34}$$

Donde $w(k)$ es el valor anterior del vector de pesos y $w(k+1)$ es el valor actualizado. Es deseable que el algoritmo converja eventualmente a la solución óptima. Se dice que es lo deseable porque finalmente el algoritmo puede divergir si no se toman las precauciones pertinentes. A continuación se describe una técnica de optimización sin restricción bajo la idea de descendencia iterativa.

II.10.2.1 Algoritmo de mínimos cuadrados.

El algoritmo de mínimos cuadrados está basado en el uso de los valores instantáneos de la función de costo:

$$\varepsilon(w) = \frac{1}{2}e^2(k) \tag{35}$$

Donde $e(k)$ es la señal de error medida en la iteración k. Diferenciando la ecuación 35 con respecto al vector de pesos w, se tiene:

$$\frac{\partial}{\partial w}\varepsilon(w) = e(k)\frac{\partial e(k)}{\partial w} \tag{36}$$

Cuando este algoritmo se utiliza para una neurona lineal, se puede expresar la señal de error como:

$$e(k) = d(k) - x^T(k)w(k) \tag{37}$$

con $d(k)$ como la salida deseada y $x(k)$ el vector de entrada. Entonces:

$$\frac{\partial e(k)}{\partial w} = -x(k)$$

y

$$\frac{\partial \varepsilon(w)}{\partial w(n)} = -x(k)e(k)$$

Usando este resultado como un estimado del vector de gradiente:

$$\hat{\mathrm{g}}(k) = -x(k)e(k) \tag{38}$$

Finalmente, se tiene:

$$\widehat{w}(k+1) = \widehat{w}(k) + \eta x(k)e(k) \tag{39}$$

donde η es el parámetro de la velocidad de aprendizaje. Al combinar la ecuación 38 y 39 es posible expresar la evolución del vector de pesos en el algoritmo de mínimos cuadrados como:

$$\widehat{w}(k+1) = \widehat{w}(k) + \eta x(k)[d(k) - x^T(k)\widehat{w}(k)] \tag{40}$$

$$= [I - \eta x(k)x^T(k)]\widehat{w}(k) + \eta x(k)d(k)$$

donde I es la matriz identidad. Al utilizar el algoritmo de mínimos cuadrados, se debe recordar que:

$$\hat{w}(k) = q^{-1}[\hat{w}(k+1)] \tag{41}$$

donde q^{-1} es el vector de retardo unitario, usando las ecuaciones 40 y 41, es posible representar el algoritmo de mínimos cuadrados como se muestra en la Figura 26.

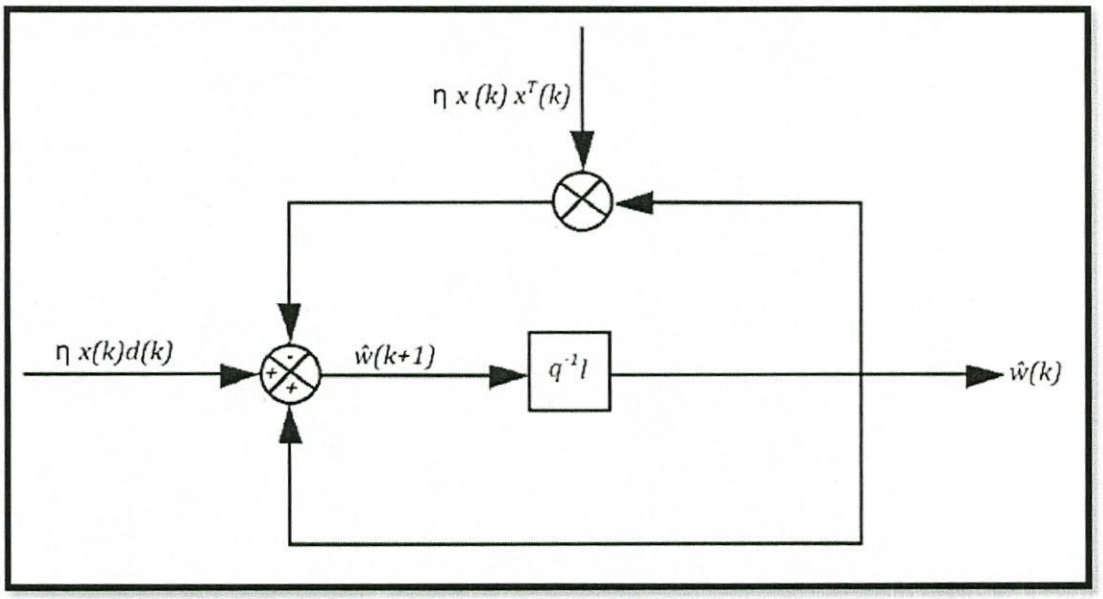

Figura 26. Algoritmo de mínimos cuadrados representado por una gráfica de flujo de señal.

II.10.3 El perceptrón multicapa.

(Minsky y Petert, 1969) publicaron su libro Perceptrons: an introduction to computacional geometry, el cual causó un desaceleramiento en el desarrollo de RNA. En él, se presentaba un análisis detallado del perceptrón, en términos de sus capacidades y limitaciones, en especial en cuanto a las restricciones que existen para las redes tipo perceptrón; la mayor desventaja de este tipo de redes es su incapacidad para resolver problemas de clasificación que no sean linealmente separables (Haykin, 1999). El perceptrón multicapa, inicialmente desarrollado por (Werbos, 1974), permite resolver este problema. Posee una estructura con almenos una capa oculta y su algoritmo de entrenamiento es del

tipo corrección de error. Se basa en el cálculo del gradiente distribuido en los diferentes componentes de la red.

Las redes neuronales tipo perceptrón multicapa (MLP), han sido aplicadas satisfactoriamente para resolver muy diversos y difíciles problemas por medio del algoritmo conocido como retropropagación; este algoritmo consta de dos etapas:

1. Etapa hacia adelante.
   Se fijan los parámetros de la red y se presenta una señal de entrada a la red, que se propaga hacia adelante para producir la salida.
2. Etapa hacia atrás.
   El error entre la salida deseada y la red se propaga hacia atrás. Los parámetros de la red se modifican para mínimizar el cuadrado del error.

   El MLP tiene tres características distintivas:
   1. El modelo de cada neurona en la red incluye una función de activación no lineal. Lo importante aquí es que la no linealidad es suave (en cualquier punto existen todas sus derivadas).
   2. La red contiene una o más capas ocultas que no son parte de las entradas o las salidas de las red. Estas neuronas ocultas permiten que la red aprenda tareas complejas por la extracción progresiva de las características principales de los patrones de entrada.
   3. La red presenta altos grados de conectividad, determinados por las sinapsis de la propia red.

La combinación de estas características junto con la habilidad de aprender de la experiencia a través del entrenamiento del MLP dan como resultado un gran potencial de computación.

II.10.4 El algoritmo de retropropagación.

La estructura del perceptrón multicapa se muestra en la Figura 27.

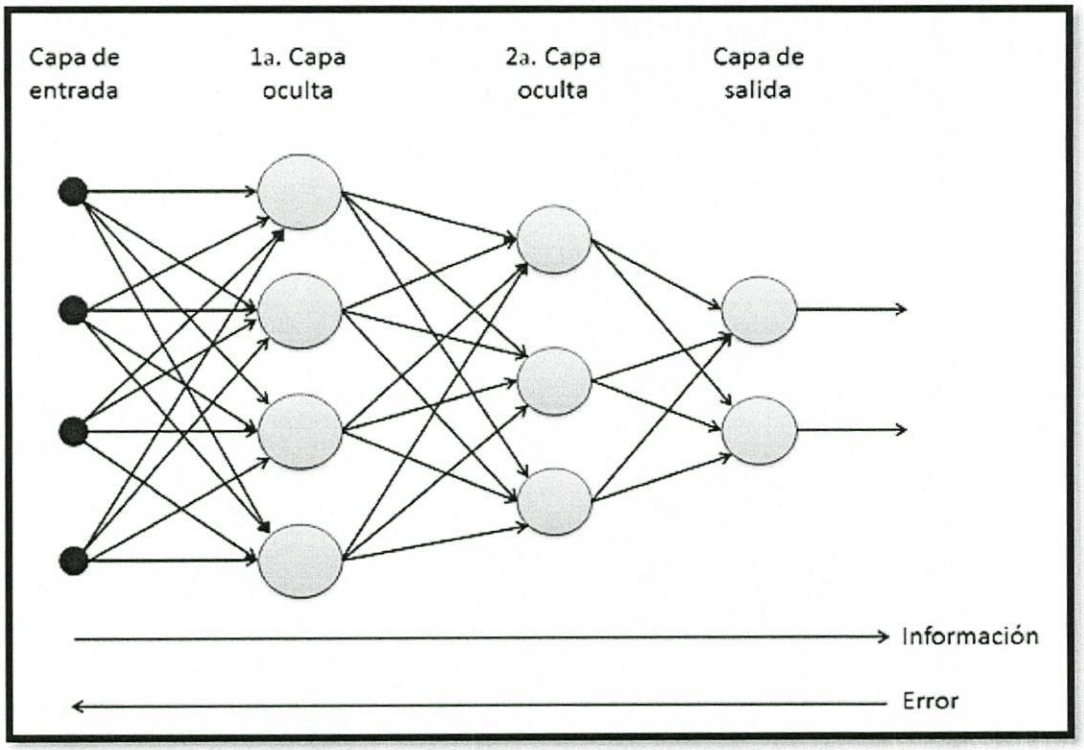

Figura 27. Red neuronal tipo MLP.

- Señales
    1. Señal de información. Constituida por el flujo de información, desde la capa de entrada hacia la capa de salida.
    2. Señal de error. Constituida por el flujo de error, desde la capa de salida hacia la capa de entrada.

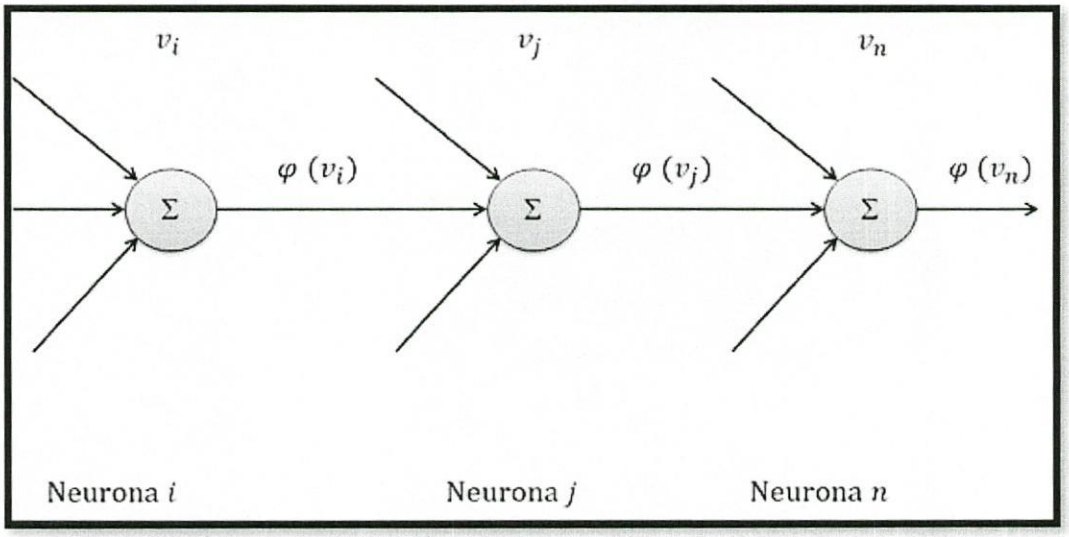

Figura 28. Elementos de una red neuronal tipo MLP.

En la Figura 28 se muestran algunos elementos que conforman la RNA:

* Notación.
  - Los subíndices i, j, n identifican las diferentes capas de la red.
  - k identifica la iteración. En la iteración k se presenta el patrón $\{x(k), d(k)\}$.
  - $e(k)$ es la suma instantánea del error al cuadrado.
  - $d_l(k)$ es la respuesta deseada de la neùrona l, de la capa de salida.
  - $y_l(k)$ representa la salida de la neurona l, de la capa de salida.
  - $e_l(k)$ es el error de la iteración k de la neurona l.
  - $w_{ij}(k)$ representa el peso que conecta la neurona i a la neurona $j\big(\Delta w_{ij}(k)\ \text{incremento respectivo}\big)$.
  - v_j es el potencial de activación de la neurona j.
  - $\varphi_j(\bullet)$ es la función de activación de la neurona j.
  - θ_j es el umbral de la neurona j.
  - $x_i(\bullet)$ representa el i-ésimo elemento de x.
  - η es la razón de aprendizaje.

II.10.4.1 Desarrollo del algoritmo de retropropagación.

Considerando la red que se muestra en la Figura 29 y la notación descrita anteriormente el j-ésimo componente del error de aproximación se puede expresar como:

$$e_j(k) = d_j(k) - y_j(k) \tag{42}$$

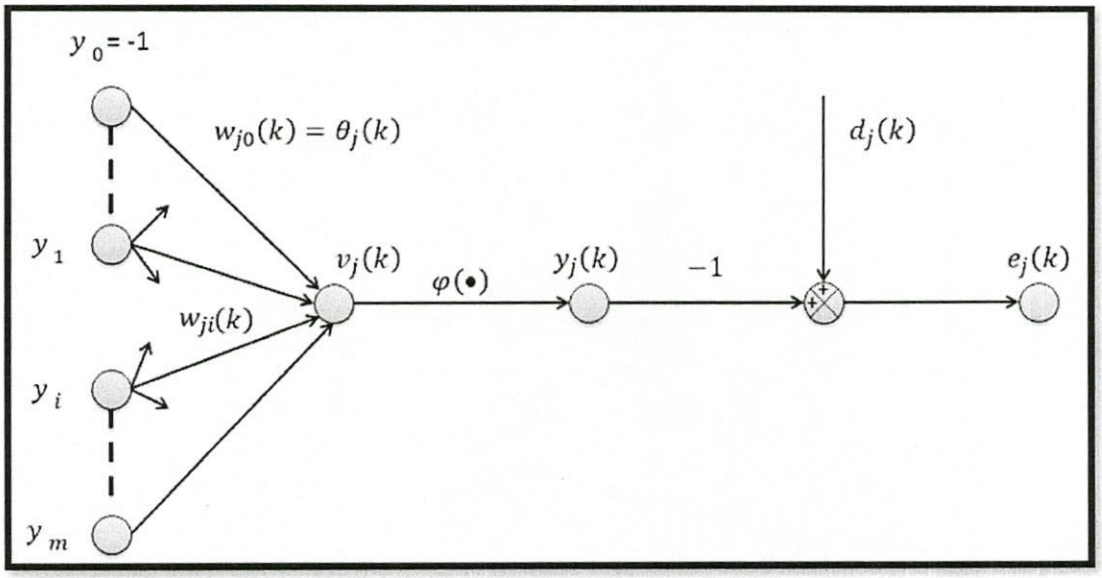

Figura 29. Diagrama del flujo de señal de informacion en la neurona j de la capa de salida.

Entonces la suma de los errores instantáneos al cuadrado se establece como:

$$\varepsilon(w) = \frac{1}{2}\sum_{j\in C}^{l} e_j^2\,(k) \tag{43}$$

donde C es el conjunto de las neuronas de salida, $C = \{1, 2, \ldots, l\}$.

$Error\ promedio\ (e_{av})$. Se obtiene de promediar los errores instantáneos correspondientes a los N pares de entrenamiento.

$$\varepsilon_{av} = \frac{1}{N}\sum_{k=1}^{N} e\,(k) \tag{44}$$

Objetivo: Minimizar e_{av} con respecto a los pesos.

Solución aproximada.

Interesa calcular $\Delta w_{ji}(k)$.

$$\frac{\partial \varepsilon(k)}{\partial w_{ji}(k)} \tag{45}$$

Por la regla de la cadena se tiene:

$$\frac{\partial \varepsilon(k)}{\partial w_{ji}(k)} = \frac{\partial \varepsilon(k)}{\partial e_j(k)}\frac{\partial e_j(k)}{\partial y_j(k)}\frac{\partial y_j(k)}{\partial v_j(k)}\frac{\partial v_j(k)}{\partial w_{ji}(k)} \tag{46}$$

De la Figura 29 que se presenta en la pag. Anterior, se tiene:

$$v_j(k) = \sum_{p=0}^{m} w_{jp}(k) y_p\,(k) \tag{47}$$

Con:

$$y_j(k) = \varphi\left(v_j(k)\right) \tag{48}$$

Considerando las ecuaciones 42, 43, 47 y 48, se definen los siguientes componentes:

$$\frac{\partial \varepsilon(k)}{\partial e_j(k)} = e_j(k)$$

$$\frac{\partial e_j(k)}{\partial y_j(k)} = -1 \tag{49}$$

$$\frac{\partial y_j(k)}{\partial v_j(k)} = \acute{\varphi}_j\left(v_j(k)\right)$$

$$\frac{\partial v_j(k)}{\partial w_{ji}(k)} = y_i(k)$$

Sustituyendo la ecuación anterior en la ecuación 46, se tiene:

$$\frac{\partial \varepsilon(k)}{\partial w_{ji}(k)} = -e_j(k)\phi_j\left(v_j(k)\right)y_i(k) \tag{50}$$

Lo cual implica que la corrección aplicada al vector de pesos está determinada por la regla delta, como sigue:

$$\Delta w_{ji}(k) = -\eta\frac{\partial \varepsilon(k)}{\partial w_{ji}(k)} = \eta e_j(k)\phi_j\left(v_j(k)\right)y_i(k) \tag{51}$$

Definiendo el gradiente local como:

$$\delta_j(k) = \frac{\partial \varepsilon(k)}{\partial v_j(k)} = -\frac{\partial \varepsilon(k)}{\partial e_j(k)}\frac{\partial e_j(k)}{\partial y_j(k)}\frac{\partial y_j(k)}{\partial v_j(k)} = e_j(k)\phi_j\left(v_j(k)\right) \tag{52}$$

entonces la regla delta se puede reformular como:

$$\Delta w_{ji} = \eta\delta_j(k)y_i(k) \tag{53}$$

De manera similar al algoritmo de mínimos cuadrados, el algoritmo de retropropagación aplica una corrección $\Delta w_{ji}(k)$ al peso sináptico $\Delta w_{ji}(k)$, la cual es proporcional al valor del gradiente local.

A partir de estas dos últimas ecuaciones se observa que el cálculo del ajuste $\Delta w_{ji}(k)$ involucra a la señal del error $e(k)$ de la neurona de salida j. En este contexto se identifican dos casos distintos, dependiendo de la ubicación de la neurona: 1) cuando la neurona es un nodo de salida y 2) cuando la neurona está situada en alguna capa oculta.

Caso 1. La neurona j es un nodo de salida. Como la neurona está en la salida, entonces es posible determinar el valor de $e(k)$ a partir de la ecuación 36 y, consecuentemente, el valor del gradiente local se determina directamente de la ecuación 53.

Caso 2. La neurona *j*está situada en alguna capa oculta. Cuando la neurona j se localiza en alguna de las capas ocultas, el valor deseado para la salida de dicha neurona no está disponible. Por lo tanto, la respectiva señal de error se debe determinar de manera recursiva en función de la señal de error de las neuronas de salida. Considérese la situación que se muestra en la Figura 31, en la cual la neurona j es una neurona oculta. De acuerdo con la ecuación 52 se puede determinar el gradiente local $\delta_j(k)$ para una neurona oculta como:

$$\delta_j(k) = \frac{\partial \varepsilon(k)}{\partial v_j(k)} = -\frac{\partial \varepsilon(k)}{\partial y_j(k)} \frac{\partial y_j(k)}{\partial v_j(k)} = -\frac{\partial \varepsilon(k)}{\partial y_j(k)} \phi_j\left(v_j(k)\right) \tag{54}$$

De acuerdo con la ecuación 43 se tiene:

$$\varepsilon(k) = \frac{1}{2} \sum_{n \in C} e_n^2(k) \tag{55}$$

Donde j es la neurona oculta y n es la neurona de salida; por lo tanto:

$$\frac{\partial \varepsilon(k)}{\partial y_j(k)} = \sum_n e_n(k) \frac{\partial e_n}{\partial y_j(k)} \tag{56}$$

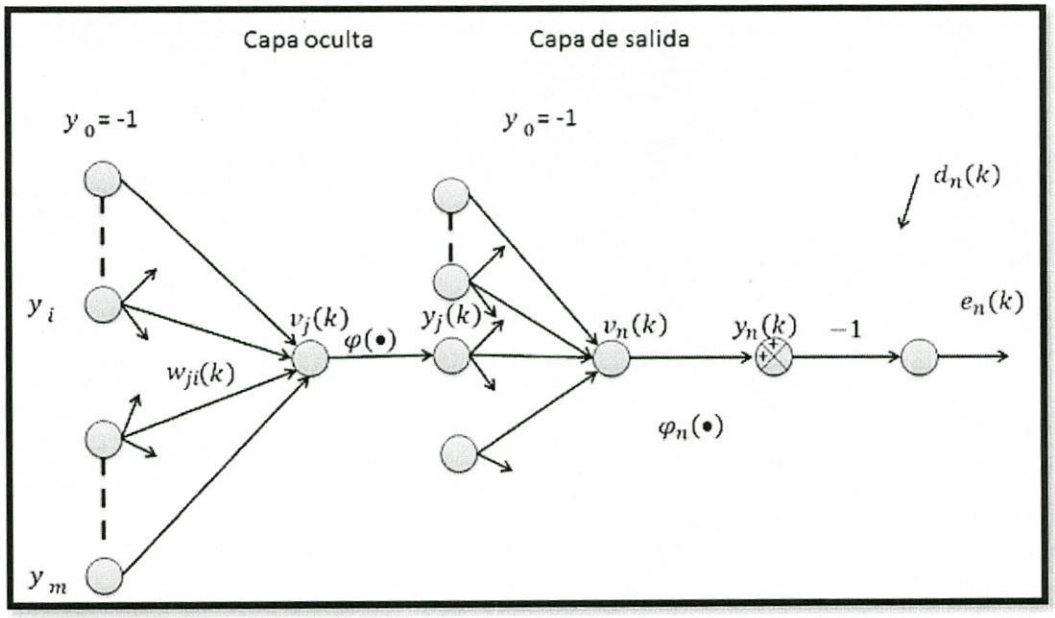

Figura 30. Diagrama del flujo de señal de información en la neurona j de la capa oculta.

Aplicando la regla de la cadena:

$$\frac{\partial \varepsilon(k)}{\partial y_j(k)} = \sum_n e_n(k) \frac{\partial e_n}{\partial v_n(k)} \frac{\partial v_n(k)}{\partial y_j(k)} \tag{57}$$

y considerando:

$$e_n(k) = d_n(k) - y_n(k) = d_n(k) - \varphi_n\big(v_n(k)\big) \tag{58}$$

por lo tanto:

$$\frac{\partial e_n}{\partial v_n(k)} = \acute{\varphi}_n\big(v_n(k)\big) \tag{59}$$

Por lo tanto:

$$v_n(k) = \sum_{j=0}^{q} w_{nj}\,(k) y_j(k) \tag{60}$$

lo cual implica:

$$\frac{\partial v_n(k)}{\partial y_j(k)} = w_{nj}(k) \tag{61}$$

Reemplazando el valor de estas derivadas en la ecuación 57, se tiene:

$$\frac{\partial \varepsilon(k)}{\partial y_j(k)} = -\sum_n e_n\,(k) \acute{\varphi}_n\big(v_n(k)\big) w_{nj}(k) \tag{62}$$

o equivalentemente:

$$\frac{\partial \varepsilon(k)}{\partial y_j(k)} = -\sum_n \delta_n\,(k) w_{nj}(k) \tag{63}$$

Finalmente:

$$\delta_j(k) = \acute{\varphi}_n\big(v_n(k)\big) \sum_n \delta_n\,(k) w_{nj}(k) = \acute{\varphi}_n\big(v_n(k)\big) \sum_n w_{nj}(k) \delta_n\,(k) \tag{64}$$

La Tabla 1, muestra un resumen del gradiente local, para los casos 1 y 2.

Tabla 1.Gradiente local.

Ubicación	Valor del gradiente local
Capa de salida	$\delta_j(k) = e_j(k)\acute{\varphi}_j\big(v_n(k)\big) = \acute{\varphi}_j\big(v_n(k)\big)e_j(k)$
Capa oculta	$\delta_j(k) = \acute{\varphi}_j\big(v_n(k)\big)\sum_n w_{nj}(k)\delta_n\,(k)$

II.10.4.2 Las dos etapas de computación.

En la aplicación del algortimo de retropropagación, se distinguen dos etapas de computación:

1. Etapa hacia adelante.

   Los pesos permanecen fijos. Para cada neurona en cada capa se calcula:

$$y_j(k) = \varphi\left(v_j(k)\right)$$

$$v_j(k) = \sum_{i=0}^{P} w_{ji}(k)y_i\,(k) = \sum_{i=0}^{P} w_{ji}(k)x_i\,(k)$$

   Conexión entre neuronas:

   a) Capa oculta: $y_j(k) = x_j(k)$

   b) Capa de salida: $O_j(k) = y_j(k)$

2. Etapa hacia atrás.

Las salidas de las neuronas permanecen fijas:

$$\Delta w_{ji} = \eta \delta_j(k) y_i(k)$$

$$e_j(k) = d_j(k) - O_j(k)$$

a) Capa oculta: $\delta_j(k) = \acute{\varphi}_j\big(v_n(k)\big) \sum_n w_{nj}(k)\delta_n(k)$

b) Capa de salida: $\delta_j(k) = e_j(k)\acute{\varphi}_j\Big(v_j(k)\Big)$

c) Es importante señalar que para las dos etapas, tanto la señal de entrada como la salida deseada permanecen constantes.

II.10.4.3 Función de activación.

El cálculo del gradiente local δ para cada neurona del perceptrón multicapa requiere el conocimiento de la derivada de la función de activación $\varphi(\bullet)$ asociada con la neurona en cuestión. Para que esa derivada exista es necesario que $\varphi(\bullet)$ sea diferenciable. Una función de activación comúnmente utilizada en el perceptrón multicapa es la no lienealidad, presentada en sus dos formas:

1. La función logística. Descrita en su forma general por:

$$\varphi\Big(v_j(k)\Big) = \frac{1}{1 + e^{-av_{j(k)}}}, \qquad -\infty < v_j(k) < \infty, \qquad a > 0 \qquad (65)$$

Por lo tanto, y de acuerdo con la ecuación 65, se tiene:

$$\acute{\varphi}_j\big(v_n(k)\big) = a y_j(k)\big[1 - y_j(k)\big]$$

Esta derivada se anula para $y_j(k) = 0$ o 1, y presenta un máximo en $v_j(k) = 0$.

El gradiente local para una neurona de salida con esta función de activación es:

$$\delta_j(k) = e_j(k)\acute{\varphi}_j\Big(v_j(k)\Big)$$
$$= a\big[d_j(k) - y_j(k)\big]y_j(k)\big[1 - y_j(k)\big] \qquad (66)$$

y para una neurona oculta:

$$\delta_j(k) = \acute{\varphi}_j\left(v_j(k)\right)\sum_n \delta_n\,(k)w_{nj}(k)$$

$$= ay_j(k)[1 - y_j(k)]\sum_n \delta_n\,(k)w_{nj}(k) \tag{67}$$

Conviene operar la neurona en su región central $v_j(k) \cong 0$; en este caso $y_j(k) \cong 0{,}5$ puesto que, si se satura $\delta_j(k) \to 0$, los pesos ya no se actualizan adecuadamente.

2. La función tangente hiperbólica. Descrita en su forma general por:

$$\varphi_j\left(v_j(k)\right) = a\,\tanh\left(bv_j(k)\right), \qquad a, b > 0 \tag{68}$$

donde a y b son constantes. Su derivada con respecto a $v_j(k)$ esta dada por:

$$\acute{\varphi}_j\left(v_j(k)\right) = ab\,\mathrm{sech}^2\left(bv_j(k)\right)$$

$$= ab\left(1 - \tanh^2\left(bv_j(k)\right)\right)$$

$$= \frac{b}{a}[a - y_j(k)][a - y_j(k)]$$

Para la neurona j localizada en la capa de salida, el gradiente local es:

$$\delta_j(k) = e_j(k)\acute{\varphi}_j\left(v_j(k)\right) = \frac{b}{a}[d_j(k) - y_j(k)][a - y_j(k)][a + y_j(k)] \tag{69}$$

y para la neurona j en una capa oculta se tiene:

$$\delta_j(k) = \acute{\varphi}_j\left(v_j(k)\right)\sum_n \delta_n\,(k)w_{nj}(k) \tag{70}$$

$$= \frac{b}{a}[a - y_j(k)][a - y_j(k)]\sum_n \delta_n\,(k)w_{nj}(k)$$

II.10.5 Aprendizaje con término de momento.

Si se selecciona un parámetro de aprendizaje η pequeño, entonces el ajuste a los pesos sinápticos que se da de iteración a iteración será pequeño: por lo tanto, el aprendizaje es lento. De lo contrario, si η se selecciona grande, entonces el aprendizaje será rápido, pero existe la posibilidad de que la red se vuelva inestable (se presentan oscilaciones). Una forma sencilla de incrementar el parámetro de la velocidad de aprendizaje de modo que se evite la posibilidad de inestabilidad consistente en modificar la regla delta mostrada en la ecuación 63, incluyendo un término de momento (Rumelhart *et. al.* 1986).

$$\Delta w_{ji}(k) = \alpha \Delta w_{ji}(k-1) + \eta \delta_j(k) y_j(k) \tag{71}$$

Donde α es la constante de momento, y usualmente es un número positivo. Esta expresión también se puede representar como un gráfico de flujo de señales, tal y como se muestra en la Figura 31. Si en la ecuación 71 $\alpha = 0$, entonces se tiene la ecuación 89; por ello a la ecuación 71 se le llama regla delta generalizada.

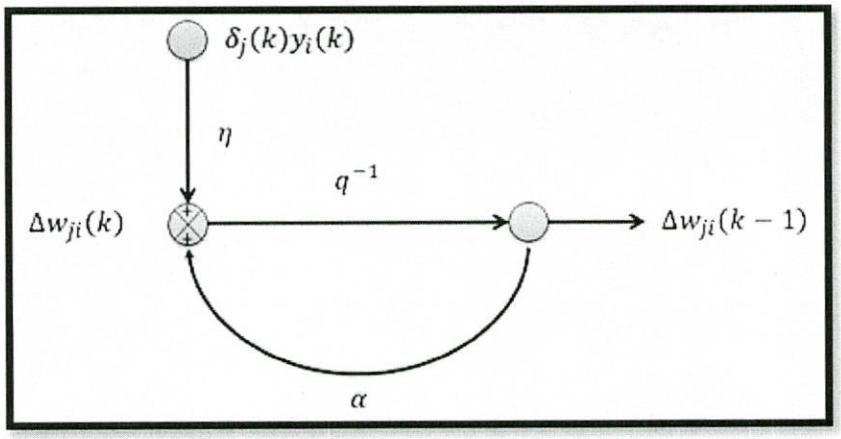

Figura 31. Efecto de la constante de momento α.

La ecuación71 se reescribe como una serie de tiempo:

$$\Delta w_{ji}(k) = \eta \sum_{i=0}^{k} \alpha^{k-1} \delta_j(l) y_i(l) \tag{72}$$

Tomando en cuenta lasecuaciones 50 y 52, la ecuación 72 se puede reformular como:

$$\Delta w_{ji}(k) = \eta \sum_{i=0}^{k} \alpha^{k-1} \frac{\partial \varepsilon(l)}{\partial w_{ji}(l)} \tag{73}$$

Basándose en la función anterior, se hacen los siguientes comentarios:

- Para que la serie de tiempo converja es necesario que la constante de momento se restrinja a: $0 \leq \alpha < 1$. Cuando $\alpha = 0$, el algoritmo de retropropagación opera sin momento.
- Cuando el signo de $\frac{\partial \varepsilon(l)}{\partial w_{ji}(l)}$ no cambia en iteraciones consecutivas, entonces $w_{ji}(k)$ es cada vez mayor.
- Cuando $\frac{\partial \varepsilon(l)}{\partial w_{ji}(l)}$ cambia de signo en iteraciones consecutivas, entonces $w_{ji}(k)$ se decrementa.

También es posible realizar otras modificaciones:

- Es posible que el parámetro de velocidad de aprendizaje η sea dependiente de la conexión y de la iteración $\eta_{ji}(k)$.
- Para el caso de que se tengan algunos pesos fijos, entonces el parametro de velocidad de aprendizaje para dichos pesos se puede tomar como $\eta_{ji}(k) = 0$.

II.10.5.1 Criterios de finalización.

En general, no se puede demostrar que el algoritmo de retropropagación converge, ni existe un criterio bien definido para finalizar. Sin embargo, algunos de los criterios más utilizados son:

1. Sea $g(w)$ el gradiente de la función de $e(w)$

$$\|g(w)\|_2^2 < \varepsilon_1$$

2. $e_{av}(w)$ estacionario

$$|\varepsilon_{av}(w)| < \varepsilon_2$$

3. Combinacion

$$\|g(w)\|_2^2 < \varepsilon_1 \cup |\varepsilon_{av}(w)| < \varepsilon_2$$

II.10.5.2 Inicialización.

Inicialización de los pesos: una vez definida la estructura de la red, se deben inicializar los pesos, si existe información inicial, ésta se debe usar; si se tiene una mala inicialización, entonces se puede presentar una saturación prematura de $\varphi(\bullet)$, lo cual no es adecuado.

Para la inicialización se deben de tomar en cuenta los siguientes aspectos:

- Distribuir aleatoriamente, con función uniforme, los pesos dentro de un rango pequeño,
- Tener un número de neuronas ocultas bajo.
- Operar las neuronas en la zona lineal.

II.10.6 Cotas en el error de aproximación.

(Barrón, 1993) estableció las propiedades de aproximación de un perceptrón multicapa, suponiendo que la red tiene sólo una capa oculta con funciones de activación sigmoidales para las neuronas ocultas y una neurona de salida con función de activación lineal. Esta propiedad queda establecida como:

$$\varepsilon_0 < \frac{mp}{N}$$

Donde ε_0 es el valor medio cuadrático del error de estimación, m es el número de neuronas ocultas, p es el número de entradas a la red neuronal y N es el número de ejemplos de entrenamiento.

El algoritmo de retropropagación se ha establecido como uno de los algoritmos más populares para el entrenamiento supervisado de redes neuronales tipo perceptrón multicapa.

II.11 Aplicaciones de redes neuronales artificiales con Matlab.

Reconocimiento de patrones mediante redes neuronales (Aldabas, 2002), aplicaciones típicas de las redes neuronales artificiales ANN (Artificial Neural Network). En ella, un perceptrón multinivel MLP (Multilayer Perceptrón) se usa para el reconocimiento óptico de caracteres OCR (Optical Character Recognition). Por último, se simula una red neuronal en el entorno Matlab, entrenándola mediante el conocido algoritmo back propagation BP.

Facultad de ingeniería, departamento Ing. eléctrica, Universidad de Antioquia. Diagnóstico de fallas en motores de inducción mediante la aplicación de redes neuronales artificiales (Villada, Cadavid, 2007): en este trabajo se presentó un nuevo método para diagnosticar fallas en el estator de motores de inducción utilizando RNA. Para tal efecto, se implementó en Matlab y se verificó experimentalmente un modelo de motor de inducción que permite simular el comportamiento del motor bajo diferentes condiciones de carga y desequilibrio de tensión. Los resultados experimentales demuestran la fortaleza del método al poder detectar de forma confiable fallas incipientes, aún en los casos donde dichas fallas evolucionen lentamente.

Implementación mediante Matlab de un sistema de reconocimiento de patrones por redes neuronales para la inspección visual de gajos de mandarinas (Roldán, 2006): reconocimiento de patrones utilizando como herramienta las redes neuronales, para solventar los problemas que padecen los sistemas en el apartado de la visión artificial, donde tienen un importante cuello de botella. El trabajo se ayuda de una herramienta versátil como Matlab para desarrollar toda una aplicación que pueda servir como clasificador de patrones. Estos patrones serán imágenes extraídas de una cinta transportadora industrial de gajos de mandarinas, y el objetivo primario del proyecto es hacer una distinción entre dos clases en principio fácilmente separables.

Implementación de redes neuronales artificiales, retropropagación con Matlab (Nazarí, Okan, 1992). La Universidad de Purdue Escuela de Ingeniería Eléctrica.
La red neuronal artificial llevara a cabo el algoritmo de retropropagación en el leguaje de Matlab. Esta implementación se compara con varios paquetes de software.
El efecto es reducir el número de iteraciones en el desempeño del algoritmo estudiado. La velocidad del programa de retropropagación, mkckpmp, escrito en lenguaje de Matlab se compara con la velocidad de otros programas de retropropagación que están escritas en el lenguaje C. La velocidad del programa Matlab mbackpmp, también en comparación con el quickpmp programa C que es una variante de la parte posterior del algoritmo. Se demuestra que el programa Matlab mbackpmp es 4.5 a 7 veces más rápido que los programas de C.

III MATERIALES Y MÉTODOS

III.1 Introducción al Matlab.

Matrix laboratory (Matlab) es un entorno de computación y desarrollo de aplicaciones totalmente integrado y orientado para llevar a cabo proyectos en donde se encuentren implicados elevados cálculos matemáticos y la visualización gráfica de los mismos. Matlab integra análisis numérico, cálculo matricial, proceso de señal y visualización gráfica en un entorno completo donde los problemas y sus soluciones son expresados del mismo modo en que se escribirían tradicionalmente, sin necesidad de hacer uso de la programación tradicional.

Matlab dispone también en la actualidad de un amplio abanico de programas de apoyo especializados, denominados toolboxes, que extienden significativamente el número de funciones incorporadas en el programa principal. Estos toolboxes cubren en la actualidad prácticamente casi todas las áreas principales en el mundo de la ingeniería y la simulación, destacando entre ellos el 'toolbox' de proceso de imágenes, señal, control robusto, estadística, análisis financiero, matemáticas simbólicas, redes neurales, lógica difusa, identificación de sistemas, simulación de sistemas dinámicos, etc. es un entorno de cálculo técnico, que se ha convertido en estándar de la industria, con capacidades no superadas en computación y visualización numérica.

De forma coherente y sin ningún tipo de fisuras, integra los requisitos claves de un sistema de computación técnico: cálculo numérico, gráficos, herramientas para aplicaciones específicas y capacidad de ejecución en múltiples plataformas. Esta familia de productos proporciona al estudiante un medio de carácter único, para resolver los problemas más complejos y difíciles.

III.2 Origen del Matlab.

Matlab nace como una solución a la necesidad de mejores y más poderosas herramientas de cálculo para resolver problemas de cálculo complejos en los que es necesario aprovechar las amplias capacidades de proceso de datos de grandes computadores.

El nombre Matlab viene de "matrix laboratory" (laboratorio matricial). Matlab fue originalmente escrito para proveer acceso fácil al software matricial desarrollado por los proyectos LINPACK y EISPACK, que juntos representan ***el estado del arte*** de software para computación matricial. Hoy Matlab es usado en una gran variedad de áreas de aplicación incluyendo procesamiento de señales e imágenes, diseño de sistemas de control, ingeniería financiera e investigación médica. La arquitectura abierta facilita usar Matlab y los productos que lo acompañan para explorar datos y

crear herramientas personalizadas que proveen visiones profundas tempranas y ventajas competitivas.

III.3 Iniciación al Matlab.

El Lenguaje de computación técnica Matlab es un ambiente de computación técnica integrada que combina computación numérica, gráficos y visualización avanzada y un lenguaje de programación de alto nivel.

Sea cual fuere el objetivo, un algoritmo, análisis, gráficos, informes o simulación, Matlab lo lleva allí. El lenguaje flexible e interactivo de Matlab permite a ingenieros y científicos expresar sus ideas técnicas con simplicidad. Los poderosos y amplios métodos de cómputo numérico y graficación permiten la prueba y exploración de ideas alternativas con facilidad, mientras que el ambiente de desarrollo integrado facilita producir resultados prácticos fácilmente.

Matlab es la fundación numérica y gráfica para todos los productos de The MathWorks. Matlab combina computación numérica, gráficos 2D y 3D y capacidades de lenguaje en un único ambiente fácil de usar.

Con un amplio rango de herramientas para modelar sistemas de control, análisis, simulación y procesamiento de prototipos, Matlab es el sistema ideal para desarrollar sistemas avanzados de control. Se puede modelar un sistema de control usando las cajas de herramientas para el diseño de controles avanzados de Matlab control system, robust control, μ-analysis and synthesis, model predictive control, QTF control design y LMI control. Posteriores análisis y refinamientos pueden ser efectuados estableciendo una simulación interactiva en simulink, y luego sintonizar automáticamente los parámetros usando el nonlinear control design blockset. Finalmente, se genera en código C para correr en controladores incrustados con real time workshop.

Combinando Matlab con signal processing toolbox, wavelet toolbox y un conjunto de herramientas complementarias - tales como image processing, neural network, fuzzy

logic, statistics y otras, se puede crear un ambiente de análisis personalizado de señales y desarrollo de algoritmos DSP. Para simulación y desarrollo de prototipos se agrega simulink y el DSP blockset para modelar y simular sistemas DSP, y luego usar real-time workshop para generar un código C para el hardware designado.

III.4 Características del entorno.

Características de Matlab:

- Cálculos intensivos desde un punto de vista numérico.
- Gráficos y visualización avanzada.
- Lenguaje de alto nivel basado en vectores, arrays y matrices.
- Colección muy útil de funciones de aplicación.

Las poderosas capacidades de cálculo técnico de Matlab se ponen a la disposición de los estudiantes, aunque limita el tamaño de las matrices 2^{13} elementos, la edición de estudiante mantiene toda la potencia de la versión profesional de Matlab 4.0, en una forma diseñada para que los estudiantes puedan ejecutarlo en sus propios ordenadores personales bajo windows.

Toolbox especiales:

Se incluyen el toolbox de señales y sistemas (un conjunto de herramientas para el procesamiento de señal y para el análisis de sistemas de cuadro) y el toolbox symbolyc math (herramienta de cálculo simbólico basada en Maple V).

III.5 Salidas o presentaciones.

Matlab provee acceso inmediato a las características gráficas especializadas requeridas en ingeniería y ciencias. Potente graficación orientada a objetos gráficos

permite graficar los resultados de su análisis, incorporar gráficos en modelos de sistemas, rápidamente presenta los complejos 3-D objetos, y crear resultados de presentación, entre lo cual se destaca:

- Representaciones 2-D y 3-D, incluyendo datos triangulados y reticulados
- Representaciones 3-D quiver, ribbon, y stem
- Control de fuentes, letras Griegas, símbolos, subíndices y superíndices
- Selección expandida de símbolos marcadores de curvas
- Gráficos de torta, de barras 3-D y de barras horizontales
- Gráficos 3-D y sólido modelado
- Representación de imágenes y archivos I/O
- Gráficos comentados
- Leer/Escribir archivos de datos Hierarchical Data Format (HDF)
- Presentación de OpenGL software y hardware
- Animación
- Display de buffer x rápido y exacto
- Soporte de colores verdaderos (24-bit RGB)
- Fuentes múltiples de luz para superficies coloreadas
- Vista basada en cámara y control de perspectiva
- Iluminación Plana, Gouraud y Phong
- Soporte eficiente de imagen de datos de 8-bit
- Control de eje y cámara
- Propiedades de superficie y patch
- Modelos de iluminación
- Control gráfico de objetos
- Impresión y representación de copias
- Formatos gráficos exportables
- Soporte de publicación de escritorio

III.6 Funciones de Matlab.

Manipulación y reducción de datos.

Matlab tiene un rango completo de funciones para preprocesar datos para análisis, incluyendo:

- y decimando
- secciones de datos
- y promediando
- y procesando umbrales
- y filtrando

Numerosas operaciones para manipular arreglos multidimensionales, incluyendo reticulación e interpolación de datos, están también disponibles.

Descriptivos gráficos para explorar y presentar sus datos gráficos de propósitos generales y de aplicación específica permiten visualizar al instante señales, superficies paramétricas, imágenes y más. Todos los atributos de los gráficos de Matlab son personalizables, desde los rótulos de ejes al ángulo de la fuente de luz en las superficies 3-D. Los gráficos están integrados con las capacidades de análisis, de modo que usted puede mostrar gráficamente cualquier conjunto de datos sin editar, ecuación o resultado funcional.

Se ingresa y saca datos de file Matlab rápidamente. Las funciones están disponibles para leer y escribir archivos de datos formateados en Matlab, llamados archivos mat. Funciones adicionales ejecutan programas ASCII o binario de bajo nivel desde los archivos de programas M, C, y Fortran, permitiéndole trabajar con todos los formatos de datos. Matlab también incluye soporte incorporado para formatos populares de archivos estándar.

Computación simbólica integrada: Integrando el motor simbólico Maple V con Matlab, los symbolic math toolboxes le permiten mezclar libremente computación simbólica y numérica una sintaxis simple e intuitiva.

Análisis de datos confiable, rápido y exacto: Los métodos usados comúnmente para análisis de datos multidimensional generalizados 1-D, 2-D están incorporados en Matlab. Interfaces gráficas fáciles de usar, específicas para aplicaciones, la línea de comando interactiva y herramientas de programación estructuradas le permiten elegir el mejor camino para sus tareas de análisis.

Análisis de datos para DSP: Matlab ofrece muchas herramientas para realizar la funcionalidad indispensable en procesamiento de señales, tales como transformadas rápidas de fourier y transformadas rápidas inversas de fourier. La visualización de datos de procesamiento de señales está soportada por funciones tales como gráficos stem y periodogramas. El lenguaje de Matlab, inherentemente orientado a matrices hace que la expresión de coeficientes de filtros y demoras de buffers sean muy simples de expresar y comprender.

Análisis de datos en aplicaciones de imágenes: Matlab y la image processing toolbox ofrece un amplio conjunto de herramientas que le permite fácilmente manipular, procesar y analizar datos de imágenes, interactivamente mostrar pantallas de imágenes 2-D o 3-D, visualizar datos temporarios cuando es necesario, y comentar sus resultados para publicaciones técnicas. La orientación basada en matrices del lenguaje de Matlab le permite expresar en forma compacta operaciones matemáticas de forma similar a cómo las expresaría sobre papel. Como resultado, es fácil e intuitivo efectuar procesamiento de imágenes y operaciones de análisis tales como FFTs, filtrado 2-D, morfología binaria, manipulación geométrica, conversión de espacios de colores, compresión, análisis de componentes conectados y más.

Algorithm development (desarrollo de algoritmos) Sea que se estén usando los algoritmos del sistema o esté inventando los propios, Matlab provee un ambiente en el que se puede experimentar. A diferencia de C y C++, Matlab permite desarrollar algoritmos desde cero o trabajar con interfaces complicadas a bibliotecas externas. La poderosa fundación de computación, el lenguaje técnico, y cientos de funciones en cajas de herramientas (toolboxes) convierten a Matlab en lo más adecuado para aplicaciones matemáticamente intensivas que requieran análisis de datos,

procesamiento de señales e imágenes, modelado de sistemas o técnicas numéricas avanzadas.

III.7 Como se define una función en Matlab.

Construcción de funciones en Matlab presenta algunas diferencias respecto de la elaboración de programas. La relevante es la siguiente: una función incluye en la primera línea del fichero, una cabecera su nombre, cuales y cuantos argumentos tiene, y cuales y cuantos valores devuelve. Dicha cabecera se identifica además, por la palabra function, tal y como se demuestra en el siguiente ejemplo:

Function a=producto(x, y)

Donde se define una función de nombre producto, que tiene dos argumentos (x e y) y devuelve un valor.

En el cuerpo de la función contiene las sentencias que sean necesarias para calcular los valores que la función va a devolver. Para calcular dichos valores se utilizan tanto los argumentos de la función como todas aquellas variables auxiliares que sean precisas. Hay que tener en cuenta que las variables que no sean argumentos locales a la función (es decir, no toman valores desde el espacio de trabajo de Matlab), y por lo tanto deberán ser inicializadas en esta.

III.8 Aplicación del algoritmo de retropropagación en una RNA.

Como se mencionó en el capítulo anterior el algoritmo de retropropagación se ha establecido como uno de los algoritmos más populares para el entrenamiento supervisado de redes neuronales tipo perceptrón multicapa, a continuación se modelará una RNA utilizando Matlab para funciones de activación logística con tangente hiperbólica en las capas ocultas y, diferentes parámetros de velocidad de

aprendizaje η, con la finalidad de determinar el aprendizaje de la RNA. Realizando varias iteraciones hasta alcanzar un error absoluto menor al 10%.

III.8.1 Combinación de funciones de activación logística con tangente hiperbólica.

En la Figura 32, se muestra la RNA donde se aplicará el algoritmo de retropropagación, considerando en las capas ocultas las funciones no lineales de la función logística con tangente hiperbólica, y en la Tabla 2 se presentan los valores iniciales de la red.

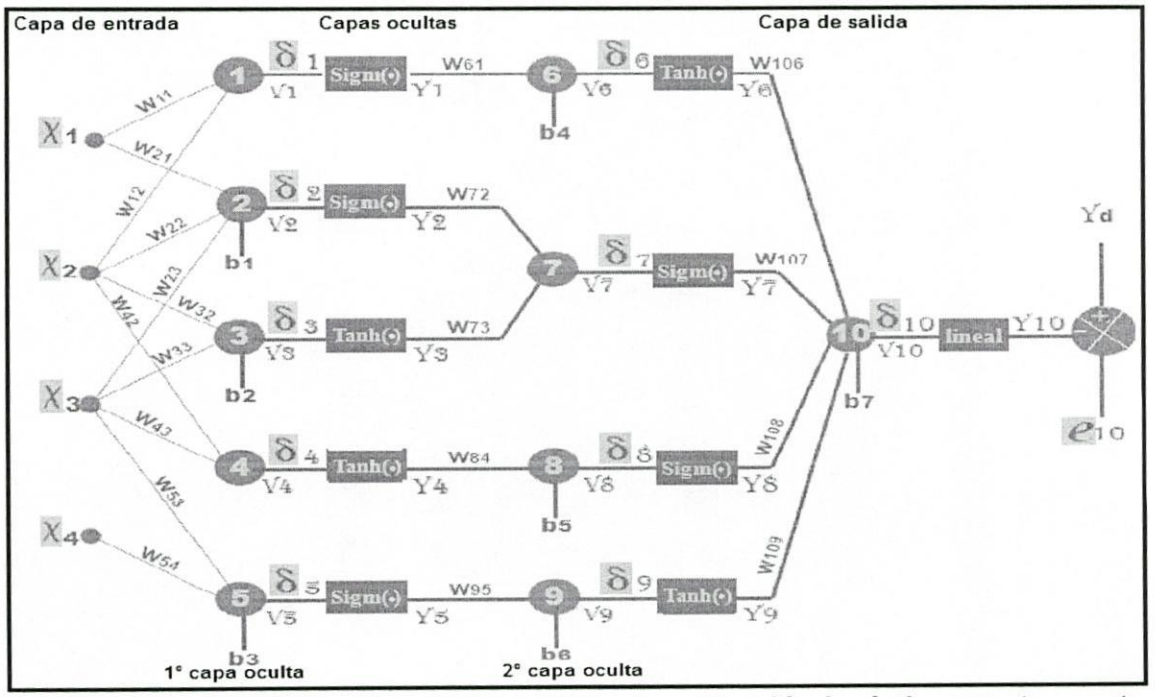

Figura 32. RNA Combinación de funciones de activación logística con tangente hiperbólica, salida lineal y η=0.2.

Tabla 2. Entrada de datos en la RNA tipo función logística con tangente hiperbólica, salida lineal y $\eta = 0.2$.

Entrada	Pesos sinápticos					Polarización	Valor deseado	Factor de aprendizaje
$x_1 = 1$	$w_{11} = -0.1$	$w_{21} = 0.2$	$w_{32} = 0.1$	$w_{42} = 0.2$	$w_{53} = -0.1$	$b_1 = -0.16$	$y_d = 1$	$\eta = 0.2$
$x_2 = -1$	$w_{12} = -0.2$	$w_{22} = -0.15$	$w_{33} = 0.2$	$w_{43} = -0.1$	$w_{54} = -0.2$	$b_2 = 0.1$		
$x_3 = 2$	$w_{61} = 0.3$	$w_{23} = 0.16$	$w_{73} = -0.15$	$w_{84} = 0.1$	$w_{95} = 0.16$	$b_3 = -0.2$		
$x_4 = -2$	$w_{106} = -0.1$	$w_{72} = -0.2$		$w_{108} = -0.2$	$w_{109} = 0.1$	$b_4 = -0.1$		
		$w_{107} = -0.7$				$b_5 = 0.2$		
						$b_6 = 0.15$		
						$b_7 = 0.1$		

En la Tabla 3 se muestra que el valor obtenido del error es mayor al 10%.

Tabla 3. Cálculo de la salida (y) y el error (e) en la RNA tipo logística con tangente hiperbólica, salida lineal y $\eta = 0.2$ en la 1ra. iteración.

$k = 1$	v	y	$e = y_d - y_{10}$
1	0.1	0.5250	1.3095
2	0.51	0.6248	
3	0.4	0.3799	
4	-0.4	-0.3799	
5	0	0.5	
6	0.057494	0.05743	
7	-0.1819	0.4546	
8	0.16201	0.54041	
9	0.23	0.22603	
10	-0.30947	-0.30947	

Lo anterior implica la necesidad de actualizar los pesos sinápticos y la polarización para continuar realizando los entrenamientos. El objetivo de aprendizaje de la RNA se alcanzó en el entrenamiento $k = 4$, utilizando un factor de aprendizaje de $\eta = 0.2$, como se muestra en la Tabla 4.

Tabla 4. Cálculo de la salida (y) y el error (e) en la RNA tipo logística con tangente hiperbólica, salida lineal y $\eta = 0.2$ en la 4ta. iteración.

$k = 4$	v	y	$e = y_d - y_{10}$
1	0.0898	0.522	0.0534
2	0,5515	0.6345	
3	0.4848	0.4581	
4	-0.4079	-0.3867	
5	0.0409	0.51023	
6	-0.0325	-0.0325	
7	-0.2641	0.4344	
8	0.1394	0.5348	
9	0.3616	0.3466	
10	0.9466	0.9466	

En la Figura 33, se muestra la RNA donde se aplicará el algoritmo de retropropagación, considerando en las capas ocultas la combinación de las funciones no lineales de sigmoidal y tangente hiperbólica con una salida lineal multiplicada por ½ y $\eta = 0.2$, y en la Tabla 5 se presentan los valores iniciales de la red.

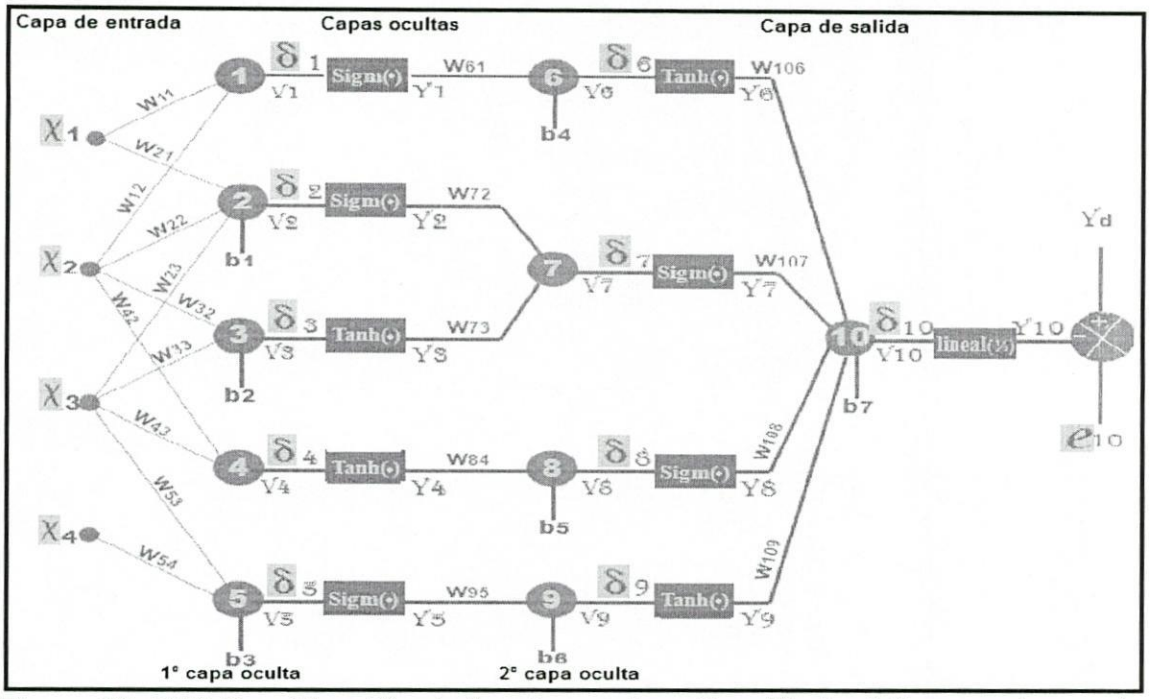

Figura 33. RNA Combinación de funciones de activación logística con tangente hiperbólica, salida lineal multiplicada por ½ y $\eta = 0.2$.

Tabla 5. Entrada de datos en la RNA tipo función logística con tangente hiperbólica, salida lineal multiplicada por ½ y $\eta = 0.2$.

Entrada	Pesos sinápticos					Polarización	Valor deseado	Factor de aprendizaje
$x_1 = 1$	$w_{11} = -0.1$	$w_{21} = 0.2$	$w_{32} = 0.1$	$w_{42} = 0.2$	$w_{53} = -0.1$	$b_1 = -0.16$	$y_d = 1$	$\eta = 0.2$
$x_2 = -1$	$w_{12} = -0.2$	$w_{22} = -0.15$	$w_{33} = 0.2$	$w_{43} = -0.1$	$w_{54} = -0.2$	$b_2 = 0.1$		
$x_3 = 2$	$w_{61} = 0.3$	$w_{23} = 0.16$	$w_{73} = -0.15$	$w_{84} = 0.1$	$w_{95} = 0.16$	$b_3 = -0.2$		
$x_4 = -2$	$w_{106} = -0.1$	$w_{72} = -0.2$		$w_{108} = -0.2$	$w_{109} = 0.1$	$b_4 = -0.1$		
		$w_{107} = -0.7$				$b_5 = 0.2$		
						$b_6 = 0.15$		
						$b_7 = 0.1$		

En la Tabla 6 se muestra que el valor obtenido del error es mayor al 10%.

Tabla 6. Cálculo de la salida (y) y el error (e) en la RNA tipo logística con tangente hiperbólica, salida lineal multiplicada por ½ y $\eta = 0.2$ en la 1ra. iteración.

$k = 1$	v	y	$e = y_d - y_{10}$
1	0.1	0.5250	1.1547
2	0.51	0.6248	
3	0.4	0.3799	
4	-0.4	-0.3799	
5	0	0.5	
6	0.0575	0.0574	
7	-0.1819	0.4546	
8	0.1620	0.5404	
9	0.23	0.2260	
10	-0.3095	-0.1547	

Lo anterior implica la necesidad de actualizar los pesos sinápticos y la polarización para continuar realizando los entrenamientos. El objetivo de aprendizaje de la RNA se alcanzó en el entrenamiento $k = 9$, utilizando un factor de aprendizaje de $\eta = 0.2$, como se muestra en la Tabla 7.

Tabla 7. Cálculo de la salida (y) y el error (e) en la RNA tipo logística con tangente hiperbólica, salida lineal multiplicada por ½ y $\eta = 0.2$ en la 9na. iteración.

$k = 9$	v	y	$e = y_d - y_{10}$
1	0.0843	0.5211	0.042
2	0.5656	0.6378	
3	0.5228	0.4799	
4	-0.386	-0.3679	
5	0.1426	0.5379	
6	-0.0903	-0.0901	
7	-0.2862	0.4289	
8	0.2015	0.5502	
9	0.5988	0.5362	
10	1.9159	0.958	

En la Figura 34, se muestra la RNA donde se aplicará el algoritmo de retropropagación, considerando en las capas ocultas la combinación de las funciones no lineales de sigmoidal y tangente hiperbólica con una función de salida no lineal de tangente hiperbólica y $\eta = 0.2$, y en la Tabla 8 se presentan los valores iniciales de la red.

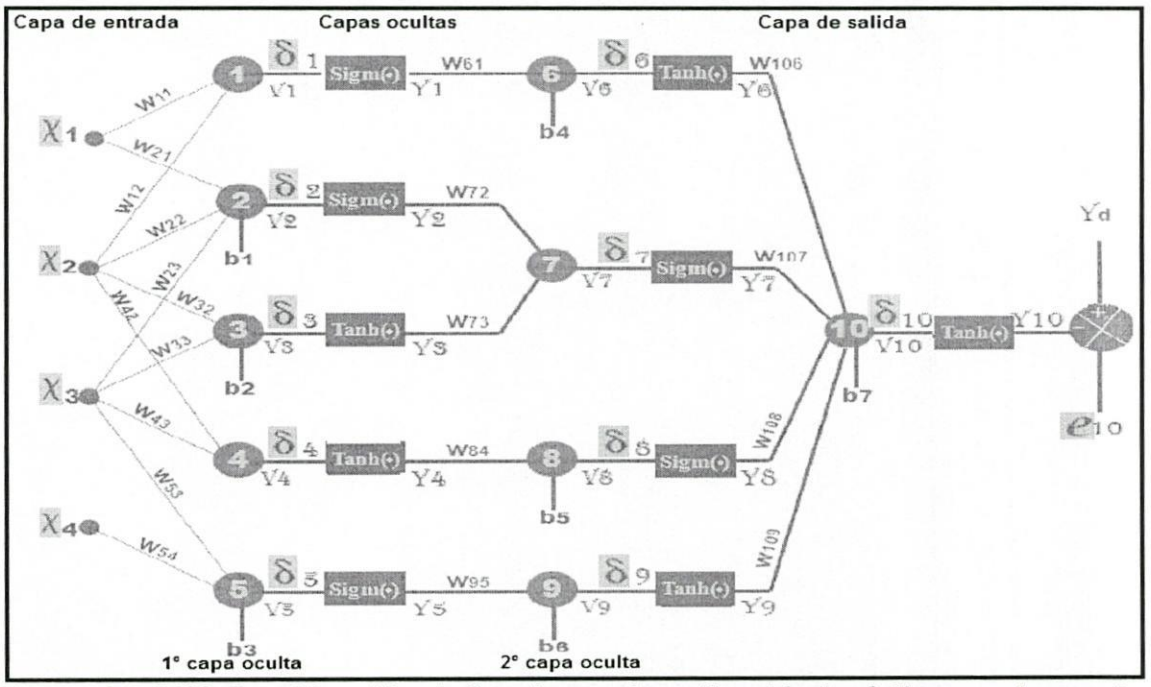

Figura 34. RNA Combinación de funciones de activación logística con tangente hiperbólica, salida no lineal de tangente hiperbólica y $\eta = 0.2$.

Tabla 8. Entrada de datos en la RNA tipo función logística con tangente hiperbólica, salida no lineal de tangente hiperbólica y $\eta = 0.2$.

Entrada	Pesos sinápticos					Polarización	Valor deseado	Factor de aprendizaje
$x_1 = 1$	$w_{11} = -0.1$	$w_{21} = 0.2$	$w_{32} = 0.1$	$w_{42} = 0.2$	$w_{53} = -0.1$	$b_1 = -0.16$	$y_d = 1$	$\eta = 0.2$
$x_2 = -1$	$w_{12} = -0.2$	$w_{22} = -0.15$	$w_{33} = 0.2$	$w_{43} = -0.1$	$w_{54} = -0.2$	$b_2 = 0.1$		
$x_3 = 2$	$w_{61} = 0.3$	$w_{23} = 0.16$	$w_{73} = -0.15$	$w_{84} = 0.1$	$w_{95} = 0.16$	$b_3 = -0.2$		
$x_4 = -2$	$w_{106} = -0.1$	$w_{72} = -0.2$		$w_{108} = -0.2$	$w_{109} = 0.1$	$b_4 = -0.1$		
		$w_{107} = -0.7$				$b_5 = 0.2$		
						$b_6 = 0.15$		
						$b_7 = 0.1$		

En la Tabla 9 se muestra que el valor obtenido del error es mayor al 10%.

Tabla 9. Cálculo de la salida (y) y el error(e) en la RNA tipo logística con tangente hiperbólica, salida no lineal de tangente hiperbólica y $\eta = 0.2$ en la 1ra. iteración.

$k = 1$	v	y	$e = y_d - y_{10}$
1	0.1	0.5250	1.3
2	0.51	0.6248	
3	0.4	0.3799	
4	-0.4	-0.3799	
5	0	0.5	
6	0.0575	0.0574	
7	-0.1819	0.4546	
8	0.1620	0.5404	
9	0.23	0.2260	
10	-0.3095	-0.3	

Lo anterior implica la necesidad de actualizar los pesos sinápticos y la polarización para continuar realizando los entrenamientos. El objetivo de aprendizaje de la RNA se alcanzó en el entrenamiento $k = 7$, utilizando un factor de aprendizaje de $\eta = 0.2$, como se muestra en la Tabla 10.

Tabla 10. Cálculo de la salida (y) y el error (e) en la RNA tipo logística con tangente hiperbólica, salida no lineal de tangente hiperbólica y $\eta = 0.2$ en la 7ma. iteración.

$k = 7$	v	y	$e = y_d - y_{10}$
1	0.0867	0.5217	0.0929
2	0.5620	0.6369	
3	0.5163	.4748	
4	-0.3972	-0.3776	
5	0.0898	0.5224	
6	-0.0639	-0.0639	
7	-0.2819	0.43	
8	0.1684	0.542	
9	0.4879	0.45	
10	1.5108	0.9071	

En la Figura 35, se muestra la RNA donde se aplicará el algoritmo de retropropagación, considerando en las capas ocultas las funciones no lineales de la función logística con tangente hiperbólica, salida lineal y $\eta = 0.05$, y en la Tabla 11 se presentan los valores iniciales de la red.

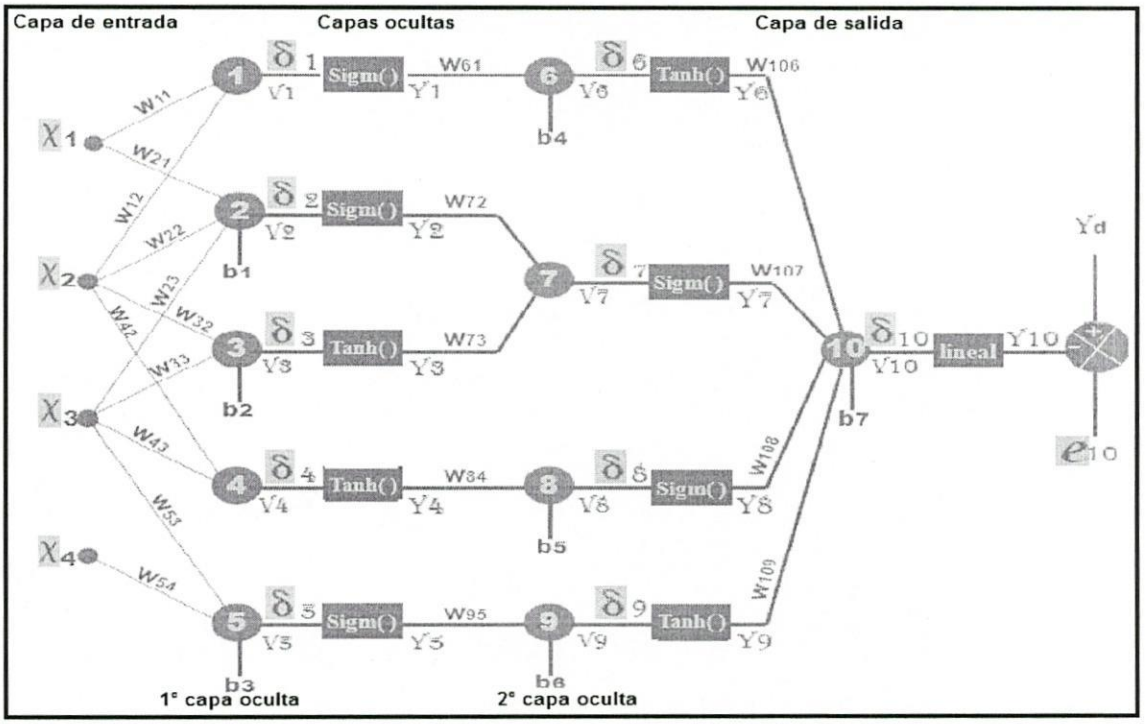

Figura 35. RNA Combinación de funciones de activación logística con tangente hiperbólica, salida lineal y $\eta = 0.05$.

Tabla 11. Entrada de datos en la RNA tipo función logística con tangente hiperbólica, salida lineal y $\eta = 0.05$.

Entrada	Pesos sinápticos					Polarización	Valor deseado	Factor de aprendizaje
$x_1 = 1$	$w_{11} = -0.1$	$w_{21} = 0.2$	$w_{32} = 0.1$	$w_{42} = 0.2$	$w_{53} = -0.1$	$b_1 = -0.16$	$y_d = 1$	$\eta = 0.05$
$x_2 = -1$	$w_{12} = -0.2$	$w_{22} = -0.15$	$w_{33} = 0.2$	$w_{43} = -0.1$	$w_{54} = -0.2$	$b_2 = 0.1$		
$x_3 = 2$	$w_{61} = 0.3$	$w_{23} = 0.16$	$w_{73} = -0.15$	$w_{84} = 0.1$	$w_{95} = 0.16$	$b_3 = -0.2$		
$x_4 = -2$	$w_{106} = -0.1$	$w_{72} = -0.2$		$w_{108} = -0.2$	$w_{109} = 0.1$	$b_4 = -0.1$		
		$w_{107} = -0.7$				$b_5 = 0.2$		
						$b_6 = 0.15$		
						$b_7 = 0.1$		

En la Tabla 12 se muestra que el valor obtenido del error es mayor al 10%.

Tabla 12. Cálculo de la salida (y) y el error (e) en la RNA tipo logística con tangente hiperbólica, salida lineal y $\eta = 0.05$ en la 1ra. iteración.

$k = 1$	v	y	$e = y_d - y_{10}$
1	0.1	0.5250	1.3095
2	0.51	0.6248	
3	0.4	0.3799	
4	-0.4	-0.3799	
5	0	0.5	
6	0.057494	0.05743	
7	-0.1819	0.4546	
8	0.16201	0.54041	
9	0.23	0.22603	
10	-0.30947	-0.30947	

Lo anterior implica la necesidad de actualizar los pesos sinápticos y la polarización para continuar realizando los entrenamientos. El objetivo de aprendizaje de la RNA se alcanzó en el entrenamiento $k = 16$, utilizando un factor de aprendizaje de $\eta = 0.05$, como se muestra en la Tabla 13.

Tabla 13. Cálculo de la salida(y) y el error(e) en la RNA tipo logística con tangente hiperbólica, salida lineal y $\eta = 0.05$ en la 16va. iteración.

$k = 16$	v	y	$e = y_d - y_{10}$
1	0.0906	0.5226	0.0847
2	0.5478	0.6336	
3	0.4848	0.45	
4	-0.4015	-0.3812	
5	0.0539	0.5135	
6	-0.0277	-0.0277	
7	-0.2534	0.4369	
8	0.1578	0.5394	
9	0.3915	0.3727	
10	0.9153	0.9153	

En la Figura 36, se muestra la RNA donde se aplicará el algoritmo de retropropagación, considerando en las capas ocultas las funciones no lineales de la función logística con tangente hiperbólica y con salida lineal multiplicada por ½ y $\eta = 0.05$, y en la Tabla 14 se presentan los valores iniciales de la red.

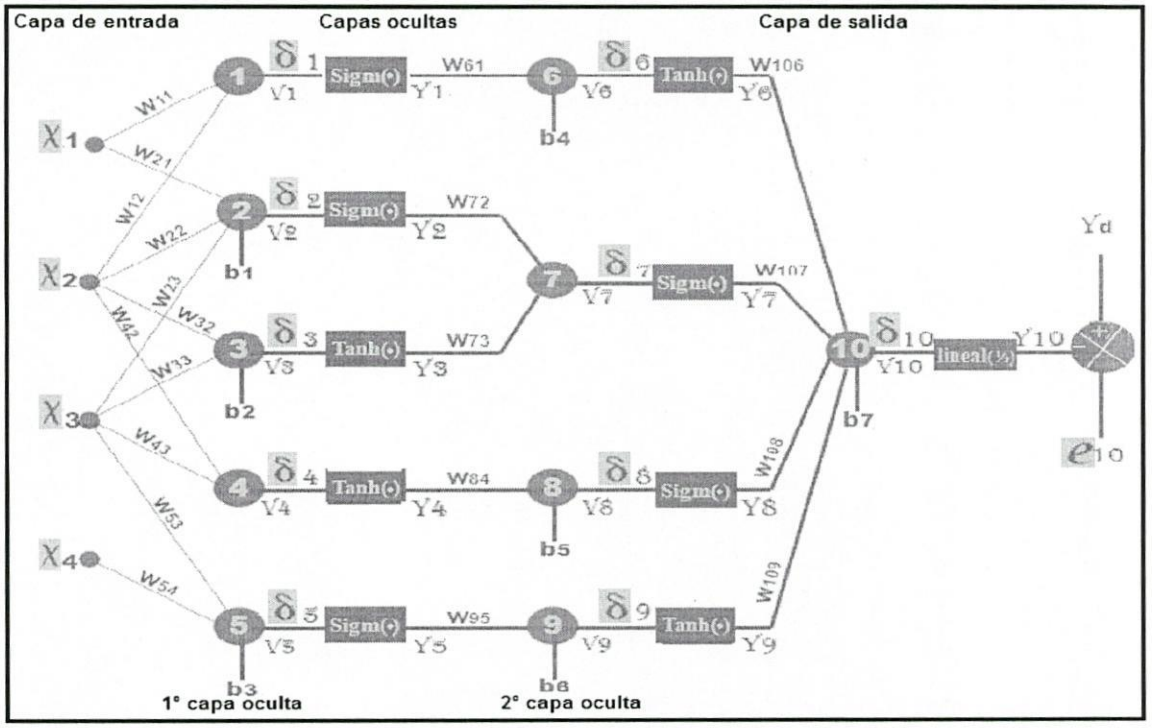

Figura 36. RNA Combinación de funciones de activación logística con tangente hiperbólica, salida lineal multiplicada por ½ y $\eta = 0.05$.

Tabla 14. Entrada de datos en la RNA tipo función logística con tangente hiperbólica, salida lineal multiplicada por ½ y $\eta = 0.05$.

Entrada	Pesos sinápticos					Polarización	Valor deseado	Factor de aprendizaje
$x_1 = 1$	$w_{11} = -0.1$	$w_{21} = 0.2$	$w_{32} = 0.1$	$w_{42} = 0.2$	$w_{53} = -0.1$	$b_1 = -0.16$	$y_d = 1$	$\eta = 0.05$
$x_2 = -1$	$w_{12} = -0.2$	$w_{22} = -0.15$	$w_{33} = 0.2$	$w_{43} = -0.1$	$w_{54} = -0.2$	$b_2 = 0.1$		
$x_3 = 2$	$w_{61} = 0.3$	$w_{23} = 0.16$	$w_{73} = -0.15$	$w_{84} = 0.1$	$w_{95} = 0.16$	$b_3 = -0.2$		
$x_4 = -2$	$w_{106} = -0.1$	$w_{72} = -0.2$		$w_{108} = -0.2$	$w_{109} = 0.1$	$b_4 = -0.1$		
		$w_{107} = -0.7$				$b_5 = 0.2$		
						$b_6 = 0.15$		
						$b_7 = 0.1$		

En la Tabla 15 se muestra que el valor obtenido del error es mayor al 10%.

Tabla 15. Cálculo de la salida (y) y el error (e) en la RNA tipo logística con tangente hiperbólica, salida lineal multiplicada por ½ y $\eta = 0.05$ en la 1ra. iteración.

$k = 1$	v	y	$e = y_d - y_{10}$
1	0.1	0.5250	1.1547
2	0.51	0.6248	
3	0.4	0.3799	
4	-0.4	-0.3799	
5	0	0.5	
6	0.0575	0.0574	
7	-0.1819	0.4546	
8	0.1620	0.5404	
9	0.23	0.2260	
10	-0.3095	-0.1547	

Lo anterior implica la necesidad de actualizar los pesos sinápticos y la polarización para continuar realizando los entrenamientos. El objetivo de aprendizaje de la RNA se alcanzó en el entrenamiento $k = 32$, utilizando un factor de aprendizaje de $\eta = 0.05$, como se muestra en la Tabla 16.

Tabla 16. Cálculo de la salida (y) y el error (e) en la RNA tipo logística con tangente hiperbólica, salida lineal multiplicada por ½ y $\eta = 0.05$ en la 32va. iteración.

$k = 32$	v	y	$e = y_d - y_{10}$
1	0.0845	0.5211	0.0849
2	0.5618	0.6369	
3	0.5143	0.4733	
4	-0.3848	-0.3669	
5	0.1496	0.5372	
6	-0.0893	-0.0891	
7	-0.2781	0.4309	
8	0.2062	0.5514	
9	0.6041	0.5399	
10	1.8303	0.9151	

En la Figura 37, se muestra la RNA donde se aplicará el algoritmo de retropropagación, considerando en las capas ocultas la combinación de las funciones no lineales de sigmoidal y tangente hiperbólica con una función de salida no lineal de tangente hiperbólica y $\eta = 0.05$, y en la Tabla 17 se presentan los valores iniciales de la red.

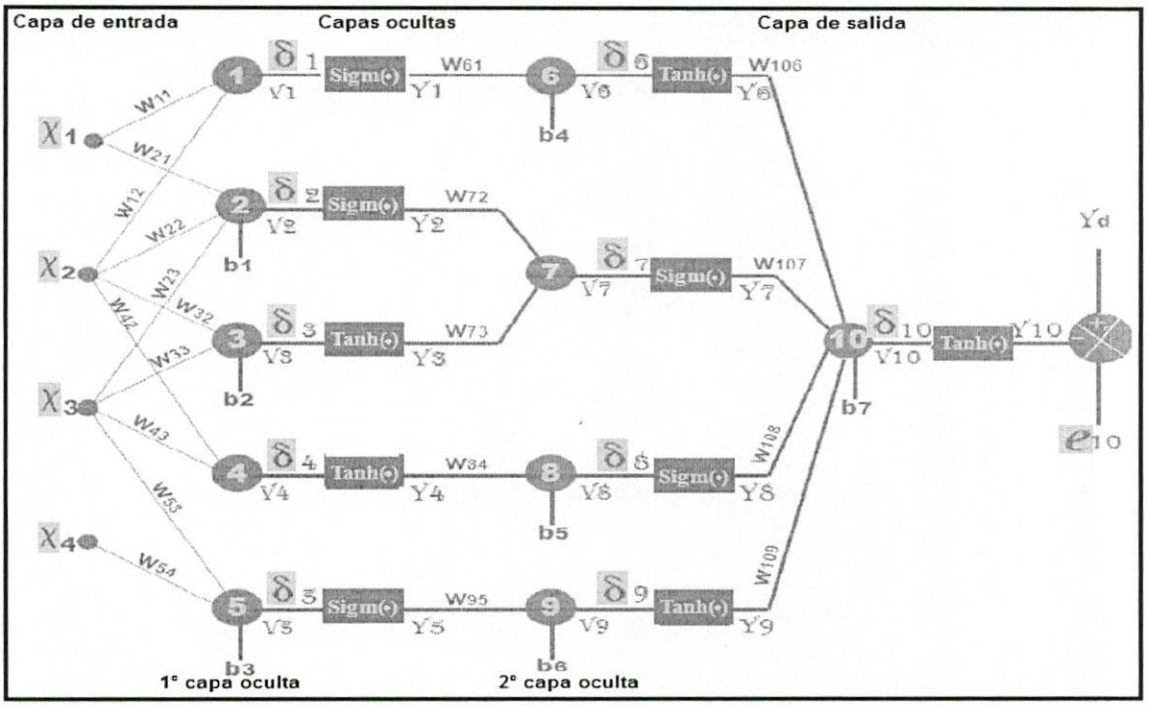

Figura 37. RNA Combinación de funciones de activación logística con tangente hiperbólica, salida no lineal de tangente hiperbólica y $\eta = 0.05$.

Tabla 17. Entrada de datos en la RNA tipo función logística con tangente hiperbólica, salida no lineal de tangente hiperbólica y $\eta = 0.05$.

Entrada	Pesos sinápticos					Polarización	Valor deseado	Factor de aprendizaje
$x_1 = 1$	$w_{11} = -0.1$	$w_{21} = 0.2$	$w_{32} = 0.1$	$w_{42} = 0.2$	$w_{53} = -0.1$	$b_1 = -0.16$	$y_d = 1$	$\eta = 0.05$
$x_2 = -1$	$w_{12} = -0.2$	$w_{22} = -0.15$	$w_{33} = 0.2$	$w_{43} = -0.1$	$w_{54} = -0.2$	$b_2 = 0.1$		
$x_3 = 2$	$w_{61} = 0.3$	$w_{23} = 0.16$	$w_{73} = -0.15$	$w_{84} = 0.1$	$w_{95} = 0.16$	$b_3 = -0.2$		
$x_4 = -2$	$w_{106} = -0.1$	$w_{72} = -0.2$		$w_{108} = -0.2$	$w_{109} = 0.1$	$b_4 = -0.1$		
		$w_{107} = -0.7$				$b_5 = 0.2$		
						$b_6 = 0.15$		
						$b_7 = 0.1$		

En la Tabla 18 se muestra que el valor obtenido del error es mayor al 10%.

Tabla 18. Cálculo de la salida (y) y el error (e) en la RNA tipo logística con tangente hiperbólica, salida no lineal de tangente hiperbólica y $\eta = 0.05$ en la 1ra. iteración.

$k = 1$	v	y	$e = y_d - y_{10}$
1	0.1	0.5250	1.3
2	0.51	0.6248	
3	0.4	0.3799	
4	−0.4	−0.3799	
5	0	0.5	
6	0.0575	0.0574	
7	−0.1819	0.4546	
8	0.1620	0.5404	
9	0.23	0.2260	
10	−0.3095	−0.3	

Lo anterior implica la necesidad de actualizar los pesos sinápticos y la polarización para continuar realizando los entrenamientos. El objetivo de aprendizaje de la RNA se alcanzó en el entrenamiento $k = 26$, utilizando un factor de aprendizaje de $\eta = 0.05$, como se muestra en la Tabla 19.

Tabla 19. Cálculo de la salida (y) y el error (e) en la RNA tipo logística con tangente hiperbólica, salida no lineal de tangente hiperbólica y $\eta = 0.05$ en la 26va. iteración.

$k = 26$	v	y	$e = y_d - y_{10}$
1	0.0868	0.5217	0.0961
2	0.5582	0.6361	
3	0.5069	0.4675	
4	-0.3921	-0.3732	
5	0.1077	0.5269	
6	-0.0654	-0.0653	
7	-0.2721	0.4324	
8	0.1843	0.546	
9	0.5181	0.4763	
10	1.4932	0.9039	

En la Figura 38, se muestra la RNA donde se aplicará el algoritmo de retropropagación, considerando en las capas ocultas las funciones no lineales de función logística con tangente hiperbólica, salida lineal y $\eta = 1$, y en la Tabla 20 se presentan los valores iniciales de la red.

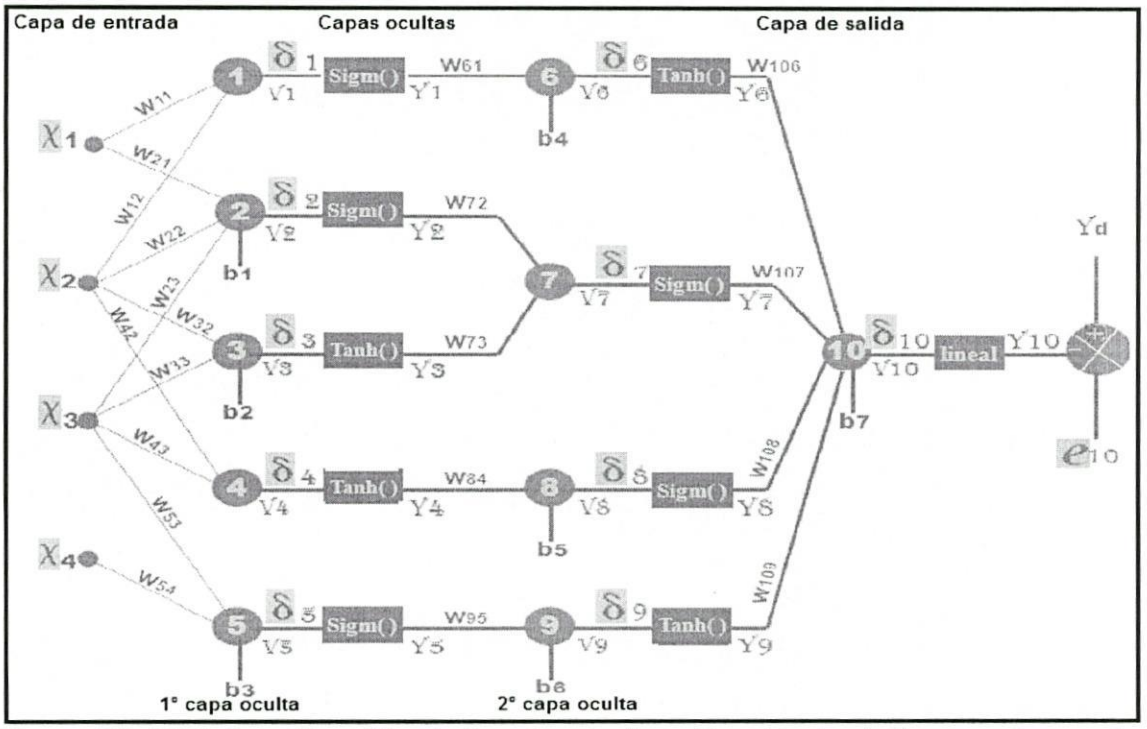

Figura 38. RNA Combinación de funciones de activación logística con tangente hiperbólica, salida lineal y $\eta = 1$.

Tabla 20. Entrada de datos en la RNA tipo función logística con tangente hiperbólica, salida lineal y $\eta = 1$.

Entrada	Pesos sinápticos					Polarización	Valor deseado	Factor de aprendizaje
$x_1 = 1$	$w_{11} = -0.1$	$w_{21} = 0.2$	$w_{32} = 0.1$	$w_{42} = 0.2$	$w_{53} = -0.1$	$b_1 = -0.16$	$y_d = 1$	$\eta = 1$
$x_2 = -1$	$w_{12} = -0.2$	$w_{22} = -0.15$	$w_{33} = 0.2$	$w_{43} = -0.1$	$w_{54} = -0.2$	$b_2 = 0.1$		
$x_3 = 2$	$w_{61} = 0.3$	$w_{23} = 0.16$	$w_{73} = -0.15$	$w_{84} = 0.1$	$w_{95} = 0.16$	$b_3 = -0.2$		
$x_4 = -2$	$w_{106} = -0.1$	$w_{72} = -0.2$		$w_{108} = -0.2$	$w_{109} = 0.1$	$b_4 = -0.1$		
		$w_{107} = -0.7$				$b_5 = 0.2$		
						$b_6 = 0.15$		
						$b_7 = 0.1$		

En la Tabla 21 se muestra que el valor obtenido del error es mayor al 10%.

Tabla 21. Cálculo de la salida (y) y el error (e) en la RNA tipo logística con tangente hiperbólica, salida lineal y $\eta = 1$ en la 1ra. iteración.

$k = 1$	v	y	$e = y_d - y_{10}$
1	0.1	0.5250	1.3095
2	0.51	0.6248	
3	0.4	0.3799	
4	-0.4	-0.3799	
5	0	0.5	
6	0.0575	0.0574	
7	-0.1819	0.4546	
8	0.1620	0.5404	
9	0.23	0.2260	
10	-0.3095	-0.3095	

En la Tabla 22 se muestra que el valor obtenido del error es mayor al 10%.

Tabla 22. Cálculo de la salida (y) y el error (e) en la RNA tipo logística con tangente hiperbólica, salida lineal y $\eta = 1$ en la 2da. iteración.

$k = 2$	v	y	$e = y_d - y_{10}$
1	0.0609	0.5252	-2.8421
2	0.6592	0.6592	
3	0.75	0.6352	
4	-0.4557	-0.4265	
5	0.0895	0.5224	
6	-0.2771	-0.2702	
7	-0.5239	0.3719	
8	0.0062	0.5015	
9	0.547	0.4983	
10	3.8421	3.8421	

En la Tabla 23 se muestra que el valor obtenido del error es mayor al 10%.

Lo anterior implica la necesidad de actualizar los pesos sinápticos y la polarización para continuar realizando los entrenamientos. El objetivo de aprendizaje de la RNA no se alcanzó, utilizando un factor de aprendizaje de $\eta = 1$, como se muestra en las Tablas 23 y 24, por lo tanto, la red no tiene convergencia.

Tabla 23. Cálculo de la salida (y) y el error (e) en la RNA tipo logística con tangente hiperbólica, salida lineal y $\eta = 1$ en la 3ra. iteración.

$k = 3$	v	y	$e = y_d - y_{10}$
1	0.0393	0.5098	3.1694
2	1.1552	0.7605	
3	1.5026	0.9056	
4	-1.5112	-0.9072	
5	-1.7978	0.1421	
6	-0.6133	-0.5465	
7	-1.3616	-0.2039	
8	-2.4609	0.0786	
9	-2.7373	-0.9917	
10	-2.1694	-2.1694	

Tabla 24. Ajuste de pesos en la 3ra. iteración en la RNA tipo logística con tangente hiperbólica, salida lineal y $\eta = 1$.

Entrada	Pesos sinápticos					Polarización	Valor deseado	Factor de aprendizaje
$x_1 = 1$	$w_{11} = -0.130$	$w_{21} = 0.2921$	$w_{32} = -0.0837$	$w_{42} = 0.4222$	$w_{53} = -0.499$	$b_1 = -0.0678$	$y_d = 1$	$\eta = 1$
$x_2 = -1$	$w_{12} = -0.169$	$w_{22} = -0.2421$	$w_{33} = 0.5675$	$w_{43} = -0.5444$	$w_{54} = 0.1995$	$b_2 = 0.2837$		
$x_3 = 2$	$w_{61} = 0.0261$	$w_{23} = 0.3443$	$w_{73} = -0.7365$	$w_{84} = 0.8860$	$w_{95} = -1.260$	$b_3 = -0.3997$		
$x_4 = -2$	$w_{106} = 1.5862$	$w_{72} = -0.9133$		$w_{108} = -1.6355$	$w_{109} = -2.140$	$b_4 = -0.6266$		
		$w_{107} = -1.6234$				$b_5 = -1.6571$		
						$b_6 = -2.5582$		
						$b_7 = -2.9653$		

En la Figura 39, se muestra la RNA donde se aplicará el algoritmo de retropropagación, considerando en las capas ocultas las funciones no lineales de la función logística con tangente hiperbólica, salida lineal multiplicada por ½ y $\eta = 1$, y en la Tabla 25 se presentan los valores iniciales de la red.

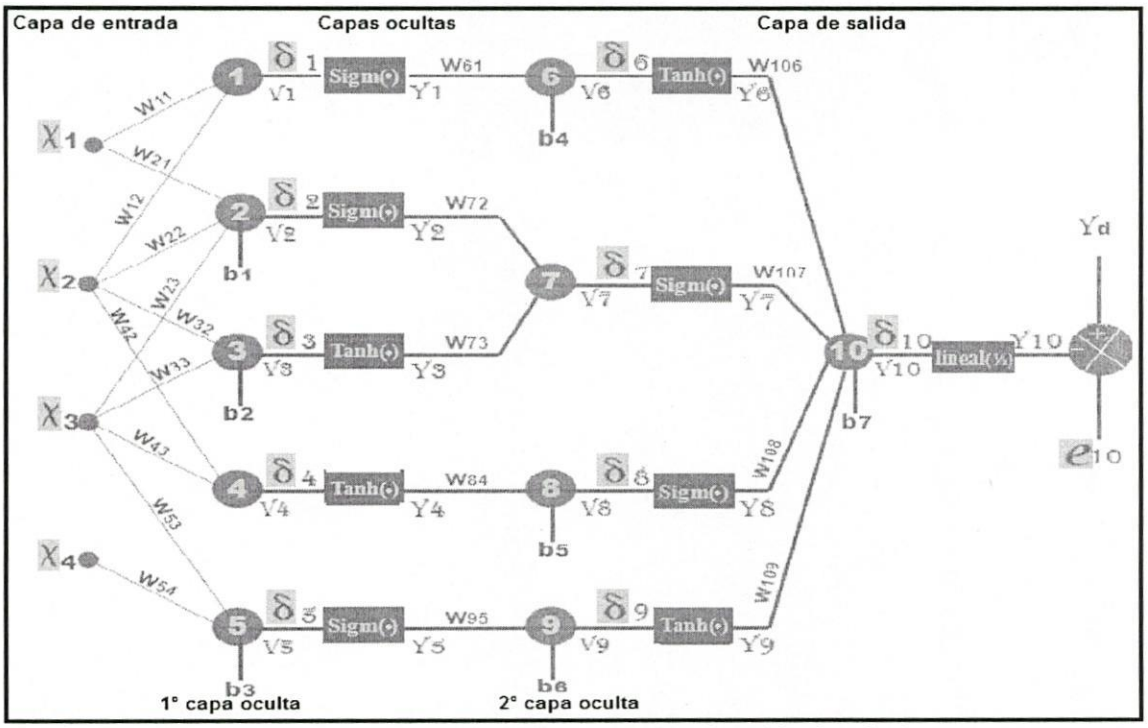

Figura 39. RNA Combinación de funciones de activación logística con tangente hiperbólica, salida lineal multiplicada por ½ y $\eta = 1$.

Tabla 25. Entrada de datos en la RNA tipo función logística con tangente hiperbólica, salida lineal multiplicada por ½ y $\eta = 1$.

Entrada	Pesos sinápticos					Polarización	Valor deseado	Factor de aprendizaje
$x_1 = 1$	$w_{11} = -0.1$	$w_{21} = 0.2$	$w_{32} = 0.1$	$w_{42} = 0.2$	$w_{53} = -0.1$	$b_1 = -0.16$	$y_d = 1$	$\eta = 1$
$x_2 = -1$	$w_{12} = -0.2$	$w_{22} = -0.15$	$w_{33} = 0.2$	$w_{43} = -0.1$	$w_{54} = -0.2$	$b_2 = 0.1$		
$x_3 = 2$	$w_{61} = 0.3$	$w_{23} = 0.16$	$w_{73} = -0.15$	$w_{84} = 0.1$	$w_{95} = 0.16$	$b_3 = -0.2$		
$x_4 = -2$	$w_{106} = -0.1$	$w_{72} = -0.2$		$w_{108} = -0.2$	$w_{109} = 0.1$	$b_4 = -0.1$		
		$w_{107} = -0.7$				$b_5 = 0.2$		
						$b_6 = 0.15$		
						$b_7 = 0.1$		

En la Tabla 26 se muestra que el valor obtenido del error es mayor al 10%.

Tabla 26. Cálculo de la salida (y) y el error (e) en la RNA tipo logística con tangente hiperbólica, salida lineal multiplicada por ½ y $\eta = 1$ en la 1ra. iteración.

$k = 1$	v	y	$e = y_d - y_{10}$
1	0.1	0.5250	1.1547
2	0.51	0.6248	
3	0.4	0.3799	
4	-0.4	-0.3799	
5	0	0.5	
6	0.0575	0.0574	
7	-0.1819	0.4546	
8	0.1620	0.5404	
9	0.23	0.2260	
10	-0.3095	-0.1547	

En la Tabla 27 se muestra que el valor obtenido del error es mayor al 10%.

Tabla 27. Cálculo de la salida (y) y el error (e) en la RNA tipo logística con tangente hiperbólica, salida lineal multiplicada por ½ y $\eta = 1$ en la 2da. iteración.

$k = 2$	v	y	$e = y_d - y_{10}$
1	0.0839	0.5209	0.2947
2	0.5714	0.6391	
3	0.5439	0.496	
4	-0.4229	-0.3994	
5	0.0368	0.5092	
6	-0.0805	-0.0803	
7	-0.3121	0.4226	
8	0.0984	0.5246	
9	0.3597	0.345	
10	1.4106	0.7053	

En la Tabla 28 se muestra que el valor obtenido del error es mayor al 10%.

Tabla 28. Cálculo de la salida (y) y el error (e) en la RNA tipo logística con tangente hiperbólica, salida lineal multiplicada por ½ y $\eta = 1$ en la 3ra. iteración.

$k = 3$	v	y	$e = y_d - y_{10}$
1	0.0809	0.5202	-0.2815
2	0.5884	0.6429	
3	0.5772	0.5206	
4	-0.3917	-0.3728	
5	0.1298	0.5324	
6	-0.1118	-0.1113	
7	-0.341	0.4156	
8	0.1725	0.543	
9	0.6137	0.5467	
10	2.5629	1.2815	

Lo anterior implica la necesidad de actualizar los pesos sinápticos y la polarización para continuar realizando los entrenamientos. El objetivo de aprendizaje de la RNA no se alcanzó, utilizando un factor de aprendizaje de $\eta = 1$, como se muestra en la Tabla 29. La red no tiene convergencia.

Tabla 29. Cálculo de la salida (y) y el error (e) en la RNA tipo logística con tangente hiperbólica, salida lineal multiplicada por ½ y $\eta = 1$ en la 4ta. iteración.

$k = 4$	v	y	$e = y_d - y_{10}$
1	0.0773	0.5193	-0.6584
2	0.5854	0.6423	
3	0.5715	0.5164	
4	-0.3652	-0.3498	
5	0.2293	0.5571	
6	-0.1526	-0.1515	
7	-0.336	0.4168	
8	0.2475	0.5616	
9	0.8065	0.6676	
10	3.3168	1.6584	

En la Figura 40, se muestra la RNA donde se aplicará el algoritmo de retropropagación, considerando en las capas ocultas la combinación de las funciones no lineales de sigmoidal y tangente hiperbólica con una función de salida no lineal de tangente hiperbólica y $\eta = 1$, y en la Tabla 30 se presentan los valores iniciales de la red.

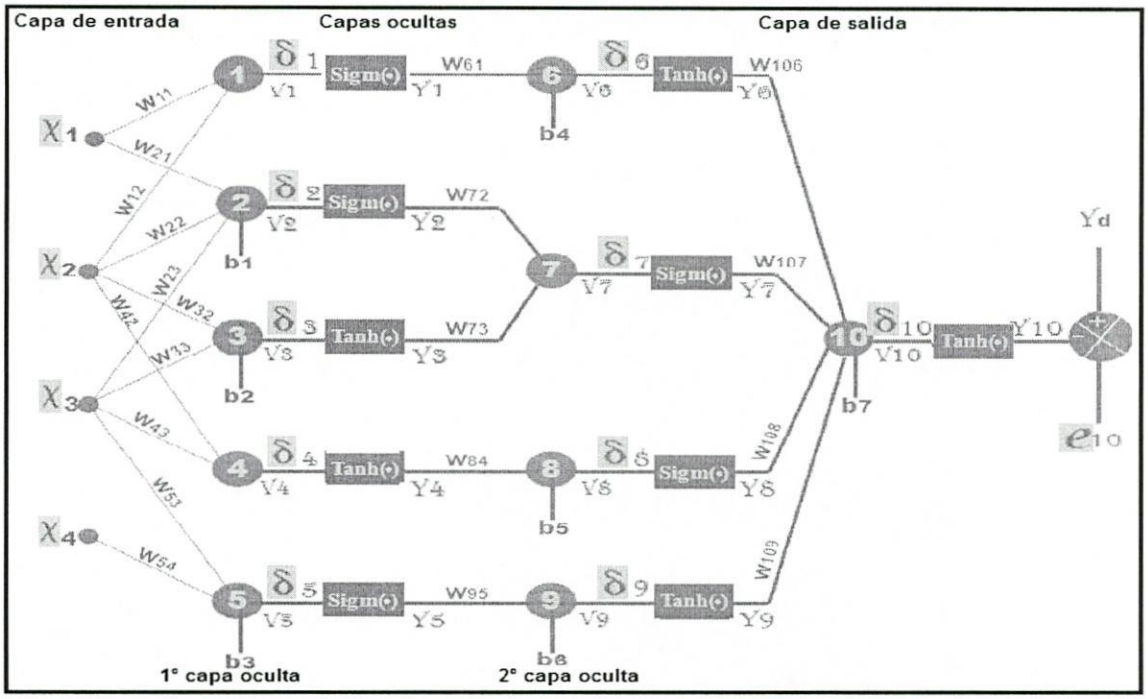

Figura 40. RNA Combinación de funciones de activación logística con tangente hiperbólica, salida no lineal de tangente hiperbólica y $\eta = 1$.

Tabla 30. Entrada de datos en la RNA tipo función logística con tangente hiperbólica, salida no lineal de tangente hiperbólica y $\eta = 1$.

Entrada	Pesos sinápticos					Polarización	Valor deseado	Factor de aprendizaje
$x_1 = 1$	$w_{11} = -0.1$	$w_{21} = 0.2$	$w_{32} = 0.1$	$w_{42} = 0.2$	$w_{53} = -0.1$	$b_1 = -0.16$	$y_d = 1$	$\eta = 1$
$x_2 = -1$	$w_{12} = -0.2$	$w_{22} = -0.15$	$w_{33} = 0.2$	$w_{43} = -0.1$	$w_{54} = -0.2$	$b_2 = 0.1$		
$x_3 = 2$	$w_{61} = 0.3$	$w_{23} = 0.16$	$w_{73} = -0.15$	$w_{84} = 0.1$	$w_{95} = 0.16$	$b_3 = -0.2$		
$x_4 = -2$	$w_{106} = -0.1$	$w_{72} = -0.2$		$w_{108} = -0.2$	$w_{109} = 0.1$	$b_4 = -0.1$		
		$w_{107} = -0.7$				$b_5 = 0.2$		
						$b_6 = 0.15$		
						$b_7 = 0.1$		

En la Tabla 31 se muestra que el valor obtenido del error es mayor al 10%.

Tabla 31. Cálculo de la salida (y) y el error (e) en la RNA tipo logística con tangente hiperbólica, salida no lineal de tangente hiperbólica y $\eta = 1$ en la 1ra. iteración.

$k = 1$	v	y	$e = y_d - y_{10}$
1	0.1	0.5250	1.3
2	0.51	0.6248	
3	0.4	0.3799	
4	-0.4	-0.3799	
5	0	0.5	
6	0.0575	0.0574	
7	-0.1819	0.4546	
8	0.1620	0.5404	
9	0.23	0.2260	
10	-0.3095	-0.3	

Lo anterior implica la necesidad de actualizar los pesos sinápticos y la polarización para continuar realizando los entrenamientos. El objetivo de aprendizaje de la RNA se alcanzó en el entrenamiento $k = 2$, utilizando un factor de aprendizaje de $\eta = 1$, como se muestra en la Tabla 32.

Tabla 32. Cálculo de la salida (y) y el error (e) en la RNA tipo logística con tangente hiperbólica, salida no lineal de tangente hiperbólica y $\eta = 1$ en la 2da. iteración.

$k = 2$	v	y	$e = y_d - y_{10}$
1	0.0654	0.5164	0.0024
2	-0.6419	0.6552	
3	0.7097	0.6105	
4	-0.4492	-0.4213	
5	0.0791	0.5198	
6	-0.2386	-0.2342	
7	-0.4805	0.3821	
8	0.0244	0.5061	
9	0.5102	0.4701	
10	3.3698	0.9976	

IV DISCUSIÓN DE RESULTADOS

En el análisis realizado se tiene que los pesos sinápticos y la polarización se van ajustando en cada iteración para encontrar los valores que minimicen el error, además se evaluaron las derivadas de la función del error con respecto de los pesos. En los problemas de combinación de funciones de activación logística con tangente hiperbolica desarrollados en el capítulo anterior, se obtiene que:

a) Para el valor de $\eta = 0.2$, se logró alcanzar la **cota del error** en la 4ta. iteración con salida lineal, en la 9na. con salida lineal multiplicada por ½ y en la 7ma. iteración con salida no lineal de tangente hiperbólica, como se muestra en las Figuras 41, 42 y 43.
b) Para el valor de $\eta = 0.05$, se logró alcanzar la cota del error en la 16va. iteración con salida lineal, en la 32va con salida lineal multiplicada por ½ y

en la 26va. iteración con salida no lineal de función tangente hiperbólica, como se muestra en las Figuras 44, 45 y 46.

c) Para el valor de $\eta = 1$, se logró alcanzar la cota del error en la 2da. iteración con salida no lineal de función tangente hiperbólica, como se muestra en la Figura 47.

d) Para el valor de $\eta = 1$ con salida lineal y lineal multiplicada por ½ el error es divergente, como se muestra en las Figuras 48 y 49.

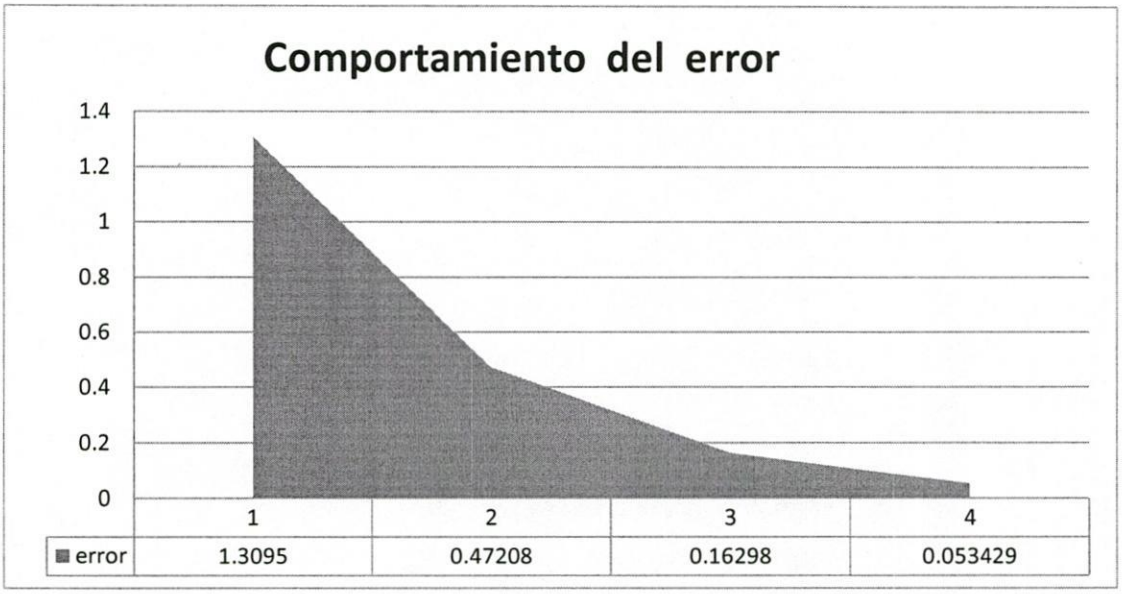

Figura 41. Convergencia de la RNA combinación de funciones de activación logística con tangente hiperbólica, salida lineal $\eta = 0.2$.

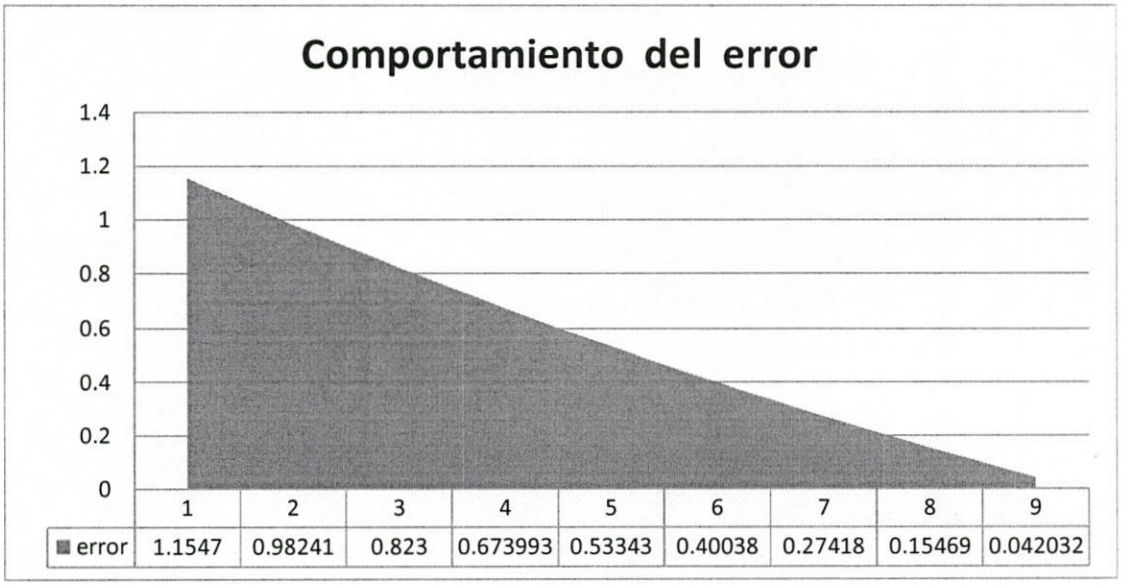

Figura 42. Convergencia de la RNA combinación de funciones de activación logística con tangente hiperbólica, salida lineal multiplicada por ½, y $\eta = 0.2$.

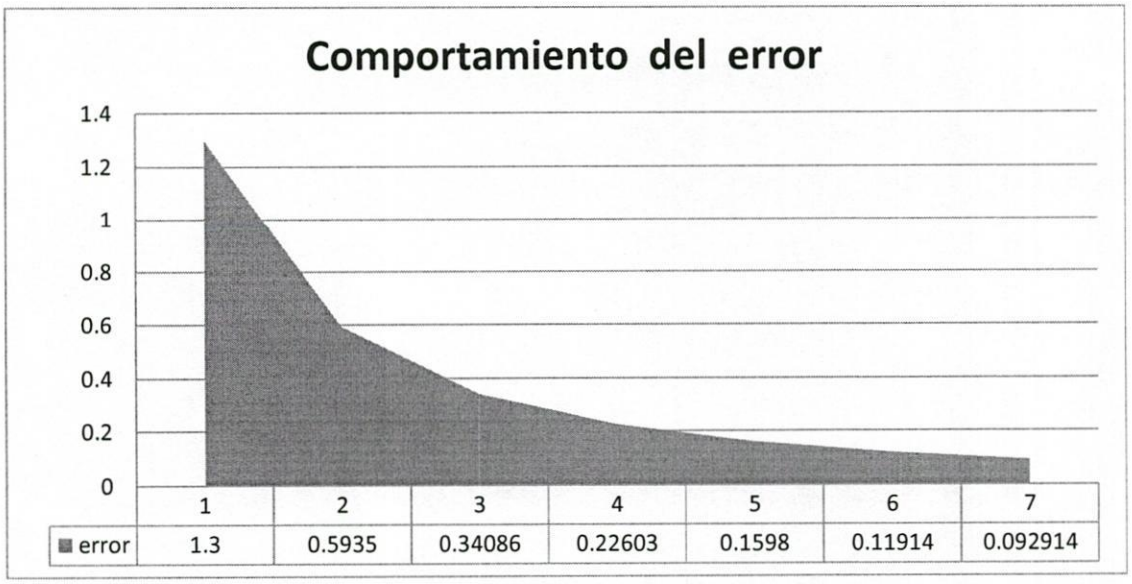

Figura 43. Convergencia de la RNA combinación de funciones de activación logística con tangente hiperbólica, salida no lineal de tangente hiperbólica y $\eta = 0.2$.

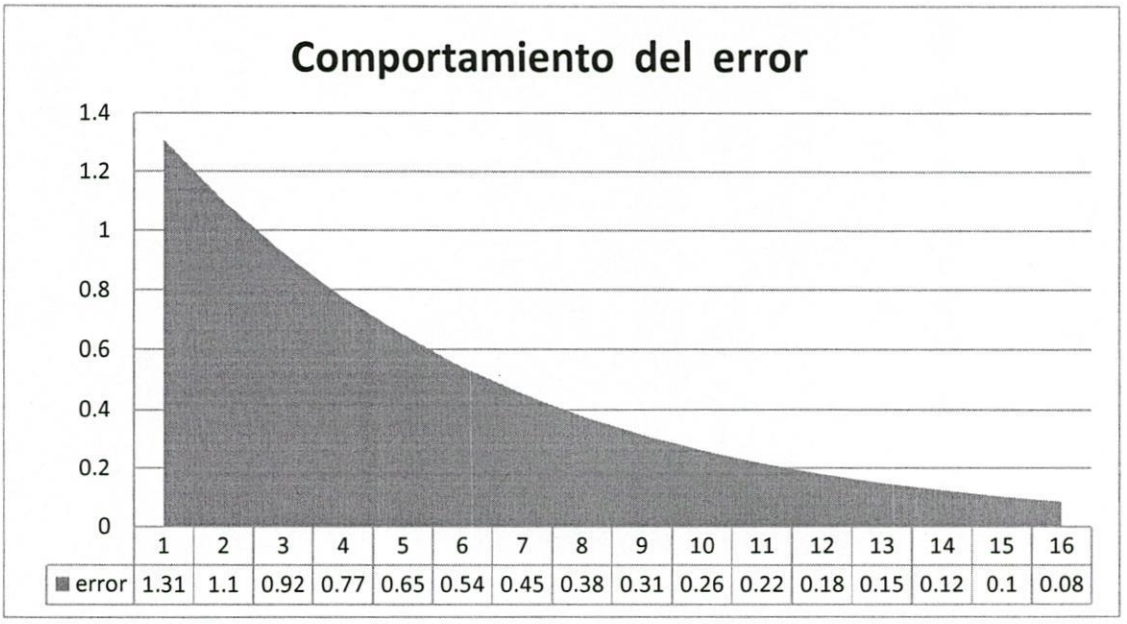

Figura 44. Convergencia de la RNA combinación de funciones de activación logística con tangente hiperbólica, salida lineal y $\eta = 0.05$.

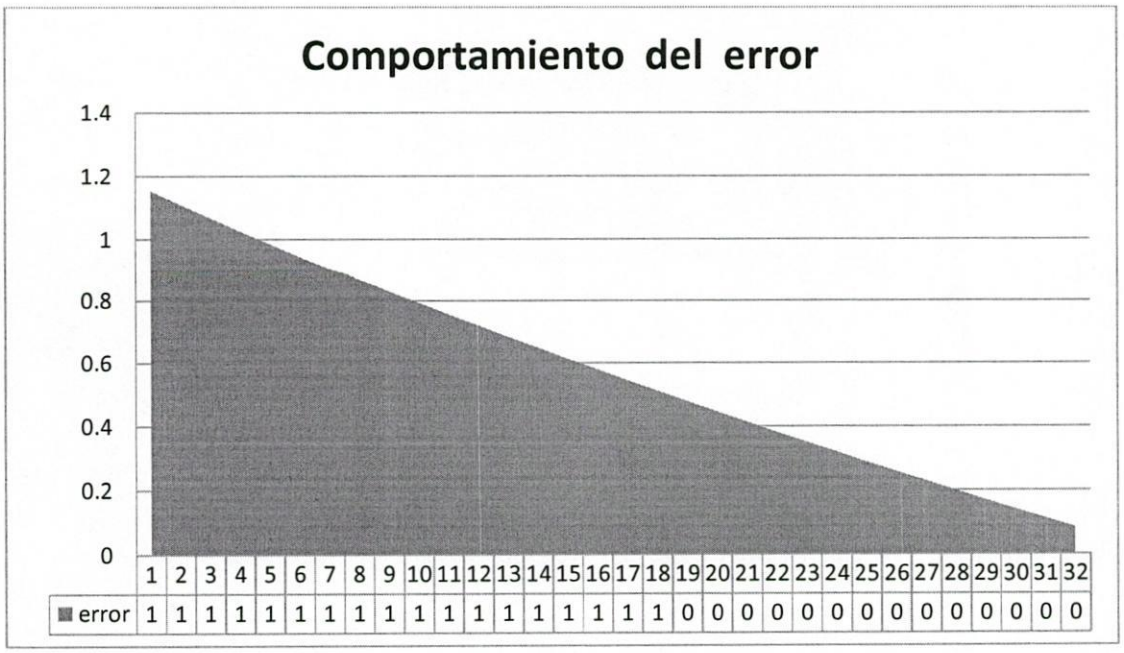

Figura 45. Convergencia de la RNA combinación de funciones de activación logística con tangente hiperbólica, salida lineal multiplicada por ½, y $\eta = 0.05$.

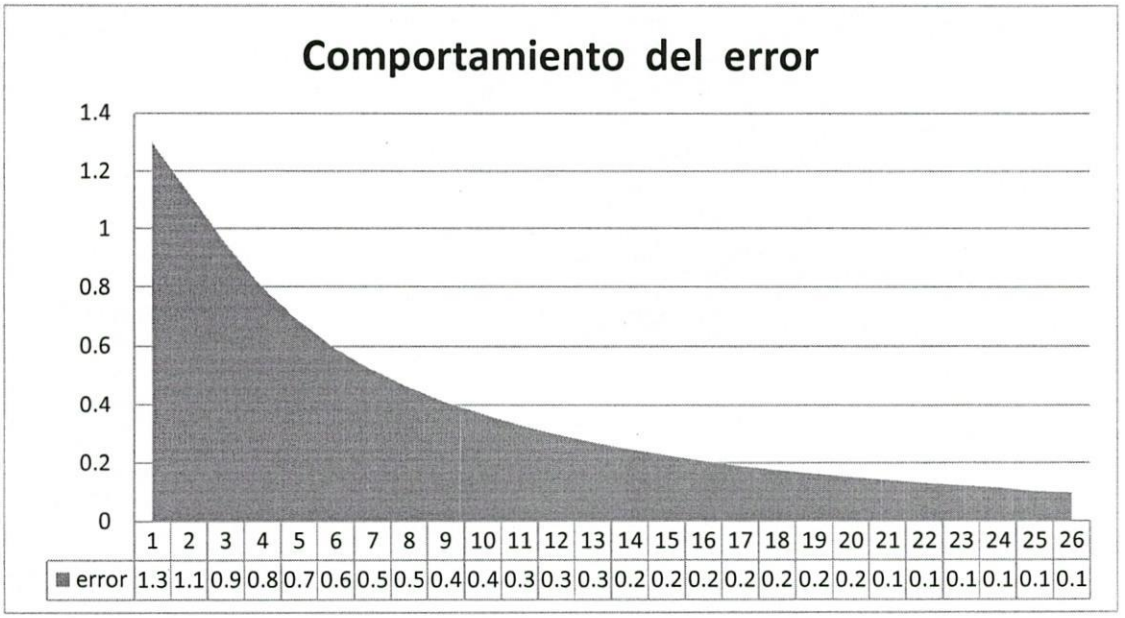

Figura 46. Convergencia de la RNA combinación de funciones de activación logística con tangente hiperbólica, salida no lineal de tangente hiperbólica y $\eta = 0.05$.

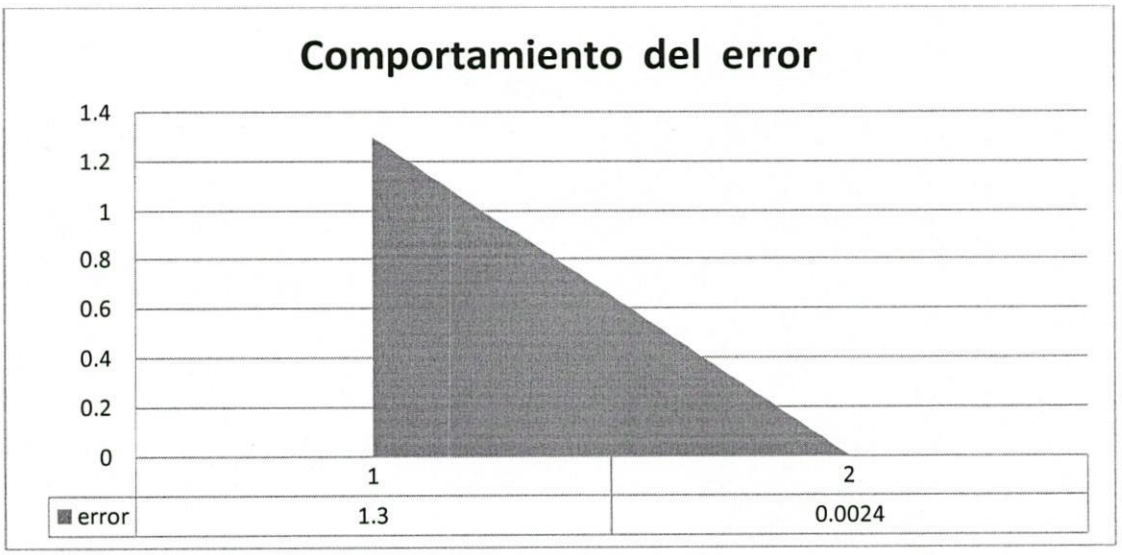

Figura 47. Convergencia de la RNA combinación de funciones de activación logística con tangente hiperbólica, salida no lineal de tangente hiperbólica y $\eta = 1$.

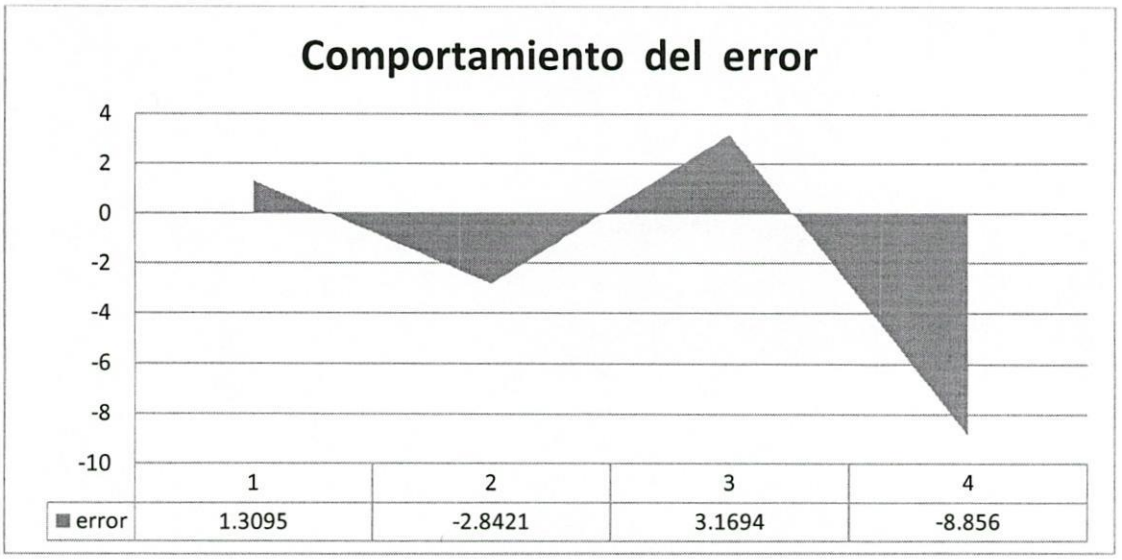

Figura 48. Divergencia de la RNA combinación de funciones de activación logística con tangente hiperbólica, salida lineal y $\eta = 1$.

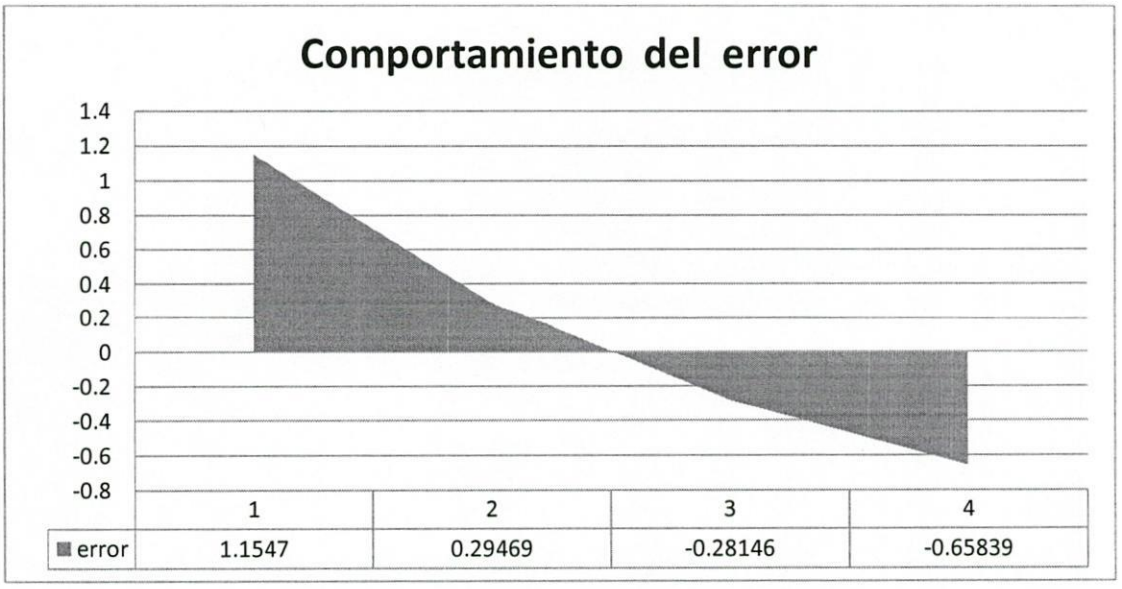

Figura 49. Divergencia de la RNA combinación de funciones de activación logística con tangente hiperbólica, salida lineal multiplicada por ½, y $\eta = 1$.

V CONCLUSIONES Y RECOMENDACIONES

Con base en los resultados obtenidos en este trabajo, se concluye que el objetivo planteado se satisface, ya que se logra conocer las funciones de activación y el factor de aprendizaje recomendable para entrenar una RNA mediante el algoritmo de retropropagación utilizando el software Matlab.

Las funciones de activación que se utilizaron y lograron alcanzar la **cota del error** planteado son:

a) La combinación de logística con tangente hiperbólica en las capas ocultas y con salida lineal, para los valores de $\eta = 0.2$ y $\eta = 0.05$, en la 4ta. y 16va. iteración, respectivamente.

b) La combinación de logística con tangente hiperbólica en las capas ocultas y con salida lineal multiplicada por ½, para los valores de $\eta = 0.2$ y $\eta = 0.05$, en la 9na. y 32va. iteración, respectivamente.

c) La combinación de logística con tangente hiperbólica en las capas ocultas y con salida no lineal de tangente hiperbólica, para los valores de $\eta = 0.2$, $\eta = 0.05$, y $\eta = 1$ en la 7ma., 26va. y 2da. iteración, respectivamente.

Lo anterior, establece usar las funciones de activación de tipo sigmoidal y de tipo tangente hiperbólica en las capas ocultas, ya que ambas son continuas y diferenciables, con valores de aprendizaje que sean $\eta \leq 0.2$.

Una vez configurada la red neuronal artificial según las necesidades del problema a resolver, se recomienda usar Matlab, ya que éste utiliza una estructura única que da acceso a todas las propiedades de la red neuronal, independientemente del tipo que ésta sea, de manera que, utilizando esta propiedad, se modifican los valores de los pesos sinápticos y polarización de las capas de entrada, capas ocultas y capas de salida.

VI BIBLIOGRAFÍA

Ackley David. H.; Hinton. G. E., y Sejnowski, T. J.: "A learning algorithm for Bolzman machines". Cognitive Science. Vol. 9. pp. 147-169. 1985.

Aldabas E. Reconocimiento de patrones mediante redes neuronales. UPC-Campus Terrassa-DEE-EUETITColom. 1 08222 Terrassa Barcelona. 2002.http://www.jcee.upc.es/JCEE2002/Aldabas.pdf

Aplicación de técnicas de redes neuronales artificiales al diagnóstico de Procesos industriales.1996. http://www.iit.upcomillas.es/docs/TesisAntonioMuNoz.pdf

Enfoque de redes neuronales hacia el control de calidad para procesos multivariados. 2001. http://ciruelo.uninorte.edu.co/pdf/ingenieria_desarrollo/9.pdf

Grossberg, S.: "Neural expectation: cerebellar and retinal analogs of cells fired by learnable or unlearned pattern classes". Kybernetik, vol. 10. pp. 49-57. 1972.

Haykin, S.: Neural Networks: A comprehensive Foundation. Second Edition. Prentice Hall. New Jersey, USA. 1999.

Hebb Donald Olding. "The organization of Behavior: A Neuropsychological Theory". Wiley. New York. 1949.

LeCun, Y.: Efficient Learning and Second-order Methods, A Tutorial NIPS 93, Demver, 1993.

LeCun, Y.: "Generalization and network design strategies", Technical Report, department of Computer Science, University of Toronto, Canada, 1989.

Leithold Louis: El Cálculo, Pepperdine University, Oxford 7 ed. pp. 975-983, 1998.

Mason Simon J.: "Feedback theory. Further properties of signal-flow graphs", proceedings of the institute of radio engineers, vol. 44, pp. 920-926, 1956.

Mason Simón J.: "Feedback theory: Some properties of signal-flow graphs", proceedings of the institute of radio engineers, vol. 41, pp. 1144-1156, 1953.

McCulloc, W. S., y Pitts, W.: "A logical calculus of the ideas immanent in neurons activity", Bulletin of Mathematical Biophysics, vol. 5, pp. 115-133, 1943.

Mendenhall William y Reinmuth James E.: Estadística para Administración y Economía, 3ra edición, Wadswordth International Iberoamérica, pp. 323-327, 1981.

Minsky M. L., y Paper, S. A.: Perceptrons, Cambridge, MA, MIT Press, 1969.

Nazari J, Ersoy O.: Implementación de redes neuronales artificiales. retropropagación con Matlab. La Universidad de Purdue Escuela de Ingeniería Eléctrica. 1992. http://docs.lib.purdue.edu/cgi/viewcontent.cgi

Roldán M.: implementación mediante Matlab de un sistema de reconocimiento de patrones por redes neuronales para la inspección visual de gajos de mandarinas. Universidad Alfonso X El Sabio. 2006. http://fundacionorange.es.pdf

Rosenblatt, F.: "The perceptrón: A probabilistic model for information storage and organization in the brain". Psychological Review. vol. 65. pp. 396-408. 1958.

Rumelhart, D. E., y Zipser, D.: "Feature discovery by competitive learning". Cognitive Science. Vol. 9. pp. 75-112. 1985.

Sánchez Camperos Edgar Nelson y Alanís García Alma Yolanda.: Redes Neuronales: conceptos fundamentales y aplicaciones a control automático. Pearson Educación, S.A. Madrid. 2006.

Simpson, P. K.: Artificial Neuronal Systems: Foundation, Paradigms, Applications and implementations. Pergamon Press. New York, USA, 1990.

Stevens, C. F.: "The neuron". Scientific American. 241 (3). pp. 54-65. 1979.

Ventajas de las Redes Neuronales:
http://www.monografias.com/trabajos12/redeneur/redneur.shtml
(http://egkafati.bligoo.com/content/view/184582/Redes-neuronales-ventajas-y-desventajas.html)e4

Villada F.; Cadavid D.: Diagnóstico de Fallas en Motores de Inducción. Facultad de Ingeniería. Departamento Ing. Eléctrica, Universidad de Antioquia. Vol. 18(2). pp. 105-112. 2007.: http://www.scielo.cl/pdf/infotec/v18n2/art16.pdf

Werbos, P. J.: Beyond regression: new tools for prediction and analysis in the behavioral sciences. Ph. D. Thesis, Harvard University. Cambridge, M.A. 1974.

Widrow, B., y Hoff Jr., M. E.: "Adaptive switching circuits". IRE WESCON Convention Record. pp. 96-104. 1960.

VII ANEXOS

VII.1. Código de Matlab para funcionamiento de una RNA con 10 neuronas y 10 funciones de activación.

```
%*************************************************************************
*
%                [Redes Neuronales Artificiales]
*
%*************************************************************************

function varargout = Red_neuronal(varargin)

gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @Red_neuronal_OpeningFcn, ...
                   'gui_OutputFcn',  @Red_neuronal_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
```

VII.1. Código de Matlab para funcionamiento de una RNA con 10 neuronas y 10 funciones de activación. *(Continuación)*

```
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end

function Red_neuronal_OpeningFcn(hObject, eventdata, handles, varargin)

axes(handles.axes1)
hm = imread('Red_neuronal.bmp');
image (hm)
axis off

function varargout = Red_neuronal_OutputFcn(hObject, eventdata, handles)
%
%**************************************************************************
**
%*                    Datos para calculo de error
%**************************************************************************

function x1_Callback(hObject, eventdata, handles)

function x1_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function x2_Callback(hObject, eventdata, handles)

function x2_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function x3_Callback(hObject, eventdata, handles)

function x3_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
```

VII.1. Código de Matlab para funcionamiento de una RNA con 10 neuronas y 10 funciones de activación. *(Continuación)*

```
function x4_Callback(hObject, eventdata, handles)

function x4_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function yd_Callback(hObject, eventdata, handles)

function yd_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function n_Callback(hObject, eventdata, handles)

function n_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function a_Callback(hObject, eventdata, handles)

function a_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function w11_Callback(hObject, eventdata, handles)

function w11_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function w12_Callback(hObject, eventdata, handles)

function w12_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'),
```

VII.1. Código de Matlab para funcionamiento de una RNA con 10 neuronas y 10 funciones de activación. *(Continuación)*

```
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function w21_Callback(hObject, eventdata, handles)

function w21_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function w22_Callback(hObject, eventdata, handles)

function w22_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function w23_Callback(hObject, eventdata, handles)

function w23_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function w32_Callback(hObject, eventdata, handles)

function w32_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function w33_Callback(hObject, eventdata, handles)

function w33_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function w42_Callback(hObject, eventdata, handles)
```

VII.1. Código de Matlab para funcionamiento de una RNA con 10 neuronas y 10 funciones de activación. *(Continuación)*

```
function w42_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function w43_Callback(hObject, eventdata, handles)

function w43_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function w53_Callback(hObject, eventdata, handles)

function w53_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function w54_Callback(hObject, eventdata, handles)

function w54_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function b1_Callback(hObject, eventdata, handles)

function b1_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function b2_Callback(hObject, eventdata, handles)

function b2_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
```

VII.1. Código de Matlab para funcionamiento de una RNA con 10 neuronas y 10 funciones de activación. *(Continuación)*

```
end

function b3_Callback(hObject, eventdata, handles)

function b3_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function b4_Callback(hObject, eventdata, handles)

function b4_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function b5_Callback(hObject, eventdata, handles)

function b5_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function b6_Callback(hObject, eventdata, handles)

function b6_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function b7_Callback(hObject, eventdata, handles)

function b7_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function v_1_Callback(hObject, eventdata, handles)

function v_1_CreateFcn(hObject, eventdata, handles)
```

VII.1. Código de Matlab para funcionamiento de una RNA con 10 neuronas y 10 funciones de activación. *(Continuación)*

```
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function V_2_Callback(hObject, eventdata, handles)

function v_2_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function v_3_Callback(hObject, eventdata, handles)

function v_3_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function v_4_Callback(hObject, eventdata, handles)

function v_4_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function v_5_Callback(hObject, eventdata, handles)

function v_5_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function v_6_Callback(hObject, eventdata, handles)

function v_6_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
```

VII.1. Código de Matlab para funcionamiento de una RNA con 10 neuronas y 10 funciones de activación. *(Continuación)*

```
function v_7_Callback(hObject, eventdata, handles)

function v_7_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function v_8_Callback(hObject, eventdata, handles)

function v_8_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function v_9_Callback(hObject, eventdata, handles)

function v_9_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function v_10_Callback(hObject, eventdata, handles)

function v_10_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function y1_Callback(hObject, eventdata, handles)

function y1_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function y2_Callback(hObject, eventdata, handles)

function y2_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'),
```

VII.1. Código de Matlab para funcionamiento de una RNA con 10 neuronas y 10 funciones de activación. *(Continuación)*

```
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function y3_Callback(hObject, eventdata, handles)

function y3_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function y4_Callback(hObject, eventdata, handles)

function y4_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function y5_Callback(hObject, eventdata, handles)

function y5_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function y6_Callback(hObject, eventdata, handles)

function y6_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function y7_Callback(hObject, eventdata, handles)

function y7_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function y8_Callback(hObject, eventdata, handles)
```

VII.1. Código de Matlab para funcionamiento de una RNA con 10 neuronas y 10 funciones de activación. *(Continuación)*

```
function y8_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function y9_Callback(hObject, eventdata, handles)

function y9_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function y10_Callback(hObject, eventdata, handles)

function y10_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function e10_Callback(hObject, eventdata, handles)

function e10_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

%**************************************************************************
*
%*                    Datos para calculo de Pesos
%**************************************************************************

function gradiente10_Callback(hObject, eventdata, handles)

function gradiente10_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function gradiente9_Callback(hObject, eventdata, handles)
```

VII.1. Código de Matlab para funcionamiento de una RNA con 10 neuronas y 10 funciones de activación. *(Continuación)*

```
function gradiente9_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function gradiente8_Callback(hObject, eventdata, handles)

function gradiente8_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function gradiente7_Callback(hObject, eventdata, handles)

function gradiente7_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function gradiente6_Callback(hObject, eventdata, handles)

function gradiente6_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function gradiente5_Callback(hObject, eventdata, handles)

function gradiente5_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function gradiente4_Callback(hObject, eventdata, handles)

function gradiente4_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
```

VII.1. Código de Matlab para funcionamiento de una RNA con 10 neuronas y 10 funciones de activación. *(Continuación)*

```
    set(hObject,'BackgroundColor','white');
end

function gradiente3_Callback(hObject, eventdata, handles)

function gradiente3_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function gradiente2_Callback(hObject, eventdata, handles)

function gradiente2_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function gradiente1_Callback(hObject, eventdata, handles)

function gradiente1_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function b1n_Callback(hObject, eventdata, handles)

function b1n_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function b2n_Callback(hObject, eventdata, handles)

function b2n_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function b3n_Callback(hObject, eventdata, handles)
```

VII.1. Código de Matlab para funcionamiento de una RNA con 10 neuronas y 10 funciones de activación. *(Continuación)*

```
function b3n_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function b4n_Callback(hObject, eventdata, handles)

function b4n_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function b5n_Callback(hObject, eventdata, handles)

function b5n_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function b6n_Callback(hObject, eventdata, handles)

function b6n_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function b7n_Callback(hObject, eventdata, handles)

function b7n_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function w11n_Callback(hObject, eventdata, handles)

function w11n_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
```

VII.1. Código de Matlab para funcionamiento de una RNA con 10 neuronas y 10 funciones de activación. *(Continuación)*

```
end

function w12n_Callback(hObject, eventdata, handles)

function w12n_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function w21n_Callback(hObject, eventdata, handles)

function w21n_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function w22n_Callback(hObject, eventdata, handles)

function w22n_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function w23n_Callback(hObject, eventdata, handles)

function w23n_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function w32n_Callback(hObject, eventdata, handles)

function w32n_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function w33n_Callback(hObject, eventdata, handles)

function w33n_CreateFcn(hObject, eventdata, handles)
```

VII.1. Código de Matlab para funcionamiento de una RNA con 10 neuronas y 10 funciones de activación. *(Continuación)*

```
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function w42n_Callback(hObject, eventdata, handles)

function w42n_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function w43n_Callback(hObject, eventdata, handles)

function w43n_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function w53n_Callback(hObject, eventdata, handles)

function w53n_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function w54n_Callback(hObject, eventdata, handles)

function w54n_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function fi1_Callback(hObject, eventdata, handles)

function fi1_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
```

VII.1. Código de Matlab para funcionamiento de una RNA con 10 neuronas y 10 funciones de activación. *(Continuación)*

```
function fi2_Callback(hObject, eventdata, handles)

function fi2_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function fi3_Callback(hObject, eventdata, handles)

function fi3_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function fi4_Callback(hObject, eventdata, handles)

function fi4_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function fi5_Callback(hObject, eventdata, handles)

function fi5_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function fi6_Callback(hObject, eventdata, handles)

function fi6_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function fi7_Callback(hObject, eventdata, handles)

function fi7_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'),
```

VII.1. Código de Matlab para funcionamiento de una RNA con 10 neuronas y 10 funciones de activación. *(Continuación)*

```
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function fi8_Callback(hObject, eventdata, handles)

function fi8_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function fi9_Callback(hObject, eventdata, handles)

function fi9_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function fi10_Callback(hObject, eventdata, handles)

function fi10_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function w61_Callback(hObject, eventdata, handles)

function w61_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function w72_Callback(hObject, eventdata, handles)

function w72_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function w73_Callback(hObject, eventdata, handles)
```

VII.1. Código de Matlab para funcionamiento de una RNA con 10 neuronas y 10 funciones de activación. *(Continuación)*

```
function w73_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function w84_Callback(hObject, eventdata, handles)

function w84_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function w95_Callback(hObject, eventdata, handles)

function w95_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function w106_Callback(hObject, eventdata, handles)

function w106_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function w107_Callback(hObject, eventdata, handles)

function w107_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function w108_Callback(hObject, eventdata, handles)

function w108_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
```

VII.1. Código de Matlab para funcionamiento de una RNA con 10 neuronas y 10 funciones de activación. *(Continuación)*

```
end

function w109_Callback(hObject, eventdata, handles)

function w109_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

%**************************************************************************
*
%*                          Calculo de error
%**************************************************************************

function calculo_error_Callback(hObject, eventdata, handles)
format short
syms v10
x_1=str2num(get(handles.x1,'String'));
x_2=str2num(get(handles.x2,'String'));
x_3=str2num(get(handles.x3,'String'));
x_4=str2num(get(handles.x4,'String'));
y_d=str2num(get(handles.yd,'String'));
w_11=str2num(get(handles.w11,'String'));
w_12=str2num(get(handles.w12,'String'));
w_21=str2num(get(handles.w21,'String'));
w_22=str2num(get(handles.w22,'String'));
w_23=str2num(get(handles.w23,'String'));
w_32=str2num(get(handles.w32,'String'));
w_33=str2num(get(handles.w33,'String'));
w_42=str2num(get(handles.w42,'String'));
w_43=str2num(get(handles.w43,'String'));
w_53=str2num(get(handles.w53,'String'));
w_54=str2num(get(handles.w54,'String'));
w_61=str2num(get(handles.w61,'String'));
w_72=str2num(get(handles.w72,'String'));
w_73=str2num(get(handles.w73,'String'));
w_84=str2num(get(handles.w84,'String'));
w_95=str2num(get(handles.w95,'String'));
w_106=str2num(get(handles.w106,'String'));
w_107=str2num(get(handles.w107,'String'));
w_108=str2num(get(handles.w108,'String'));
w_109=str2num(get(handles.w109,'String'));
b_1=str2num(get(handles.b1,'String'));
b_2=str2num(get(handles.b2,'String'));
b_3=str2num(get(handles.b3,'String'));
b_4=str2num(get(handles.b4,'String'));
b_5=str2num(get(handles.b5,'String'));
b_6=str2num(get(handles.b6,'String'));
b_7=str2num(get(handles.b7,'String'));
fi_1=(get(handles.fi1,'String'));
```

VII.1. Código de Matlab para funcionamiento de una RNA con 10 neuronas y 10 funciones de activación. *(Continuación)*

```
fi_2=(get(handles.fi2,'String'));
fi_3=(get(handles.fi3,'String'));
fi_4=(get(handles.fi4,'String'));
fi_5=(get(handles.fi5,'String'));
fi_6=(get(handles.fi6,'String'));
fi_7=(get(handles.fi7,'String'));
fi_8=(get(handles.fi8,'String'));
fi_9=(get(handles.fi9,'String'));
fi_10=(get(handles.fi10,'String'));

der10=diff((fi_10),v10);
v1=(x_1*w_11)+(x_2*w_12);
y_1=eval(fi_1);
v2=(x_1*w_21)+(x_2*w_22)+(x_3*w_23)+b_1;
y_2=eval(fi_2);
v3=(x_2*w_32)+(x_3*w_33)+b_2;
y_3=eval(fi_3);
v4=(x_2*w_42)+(x_3*w_43);
y_4=eval(fi_4);
v5=(x_3*w_53)+(x_4*w_54)+b_3;
y_5=eval(fi_5);
v6=(y_1*w_61)+b_4;
y_6=eval(fi_6);
v7=(y_2*w_72)+(y_3*w_73);
y_7=eval(fi_7);
v8=(y_4*w_84)+b_5;
y_8=eval(fi_8);
v9=(y_5*w_95)+b_6;
y_9=eval(fi_9);
v10=(y_6*w_106)+(y_7*w_107)+(y_8*w_108)+(y_9*w_109)+b_7;
y_10=eval(fi_10);
fi_10der=eval(der10);

e_10=(y_d)-y_10;

gradiente_10=2*e_10*fi_10der;

set(handles.v_1, 'String',num2str(v1))
set(handles.v_2, 'String',num2str(v2))
set(handles.v_3, 'String',num2str(v3))
set(handles.v_4, 'String',num2str(v4))
set(handles.v_5, 'String',num2str(v5))
set(handles.v_6, 'String',num2str(v6))
set(handles.v_7, 'String',num2str(v7))
set(handles.v_8, 'String',num2str(v8))
set(handles.v_9, 'String',num2str(v9))
set(handles.v_10, 'String',num2str(v10))
set(handles.y1, 'String',num2str(y_1))
set(handles.y2, 'String',num2str(y_2))
set(handles.y3, 'String',num2str(y_3))
set(handles.y4, 'String',num2str(y_4))
set(handles.y5, 'String',num2str(y_5))
set(handles.y6, 'String',num2str(y_6))
```

VII.1. Código de Matlab para funcionamiento de una RNA con 10 neuronas y 10 funciones de activación. *(Continuación)*

```
set(handles.y7, 'String',num2str(y_7))
set(handles.y8, 'String',num2str(y_8))
set(handles.y9, 'String',num2str(y_9))
set(handles.y10, 'String',num2str(y_10))
set(handles.e10, 'String',num2str(e_10))
%set(handles.gradiente10, 'String',num2str(gradiente_10))

%**************************************************************************
*
%*                          Calculo de pesos
%**************************************************************************

function calculo_pesos_Callback(hObject, eventdata, handles)
format short
syms v1 v2 v3 v4 v5 v6 v7 v8 v9 v10

n_=str2num(get(handles.n,'String'));
a_=str2num(get(handles.a,'String'));
x_1=str2num(get(handles.x1,'String'));
x_2=str2num(get(handles.x2,'String'));
x_3=str2num(get(handles.x3,'String'));
x_4=str2num(get(handles.x4,'String'));
y_d=str2num(get(handles.yd,'String'));
w_11=str2num(get(handles.w11,'String'));
w_12=str2num(get(handles.w12,'String'));
w_21=str2num(get(handles.w21,'String'));
w_22=str2num(get(handles.w22,'String'));
w_23=str2num(get(handles.w23,'String'));
w_32=str2num(get(handles.w32,'String'));
w_33=str2num(get(handles.w33,'String'));
w_42=str2num(get(handles.w42,'String'));
w_43=str2num(get(handles.w43,'String'));
w_53=str2num(get(handles.w53,'String'));
w_54=str2num(get(handles.w54,'String'));
w_61=str2num(get(handles.w61,'String'));
w_72=str2num(get(handles.w72,'String'));
w_73=str2num(get(handles.w73,'String'));
w_84=str2num(get(handles.w84,'String'));
w_95=str2num(get(handles.w95,'String'));
w_106=str2num(get(handles.w106,'String'));
w_107=str2num(get(handles.w107,'String'));
w_108=str2num(get(handles.w108,'String'));
w_109=str2num(get(handles.w109,'String'));
b_1=str2num(get(handles.b1,'String'));
b_2=str2num(get(handles.b2,'String'));
b_3=str2num(get(handles.b3,'String'));
b_4=str2num(get(handles.b4,'String'));
b_5=str2num(get(handles.b5,'String'));
b_6=str2num(get(handles.b6,'String'));
b_7=str2num(get(handles.b7,'String'));
fi_1=(get(handles.fi1,'String'));
fi_2=(get(handles.fi2,'String'));
fi_3=(get(handles.fi3,'String'));
```

VII.1. Código de Matlab para funcionamiento de una RNA con 10 neuronas y 10 funciones de activación. *(Continuación)*

```
fi_4=(get(handles.fi4,'String'));
fi_5=(get(handles.fi5,'String'));
fi_6=(get(handles.fi6,'String'));
fi_7=(get(handles.fi7,'String'));
fi_8=(get(handles.fi8,'String'));
fi_9=(get(handles.fi9,'String'));
fi_10=(get(handles.fi10,'String'));

der10=diff((fi_10),v10);
der9=diff((fi_9),v9);
der8=diff((fi_8),v8);
der7=diff((fi_7),v7);
der6=diff((fi_6),v6);
der5=diff((fi_5),v5);
der4=diff((fi_4),v4);
der3=diff((fi_3),v3);
der2=diff((fi_2),v2);
der1=diff((fi_1),v1);

der10=diff((fi_10),v10);
v1=(x_1*w_11)+(x_2*w_12);
y_1=eval(fi_1);
v2=(x_1*w_21)+(x_2*w_22)+(x_3*w_23)+b_1;
y_2=eval(fi_2);
v3=(x_2*w_32)+(x_3*w_33)+b_2;
y_3=eval(fi_3);
v4=(x_2*w_42)+(x_3*w_43);
y_4=eval(fi_4);
v5=(x_3*w_53)+(x_4*w_54)+b_3;
y_5=eval(fi_5);
v6=(y_1*w_61)+b_4;
y_6=eval(fi_6);
v7=(y_2*w_72)+(y_3*w_73);
y_7=eval(fi_7);
v8=(y_4*w_84)+b_5;
y_8=eval(fi_8);
v9=(y_5*w_95)+b_6;
y_9=eval(fi_9);
v10=(y_6*w_106)+(y_7*w_107)+(y_8*w_108)+(y_9*w_109)+b_7;
y_10=eval(fi_10);

e_10=(y_d)-y_10*der10;

fi_10der=eval(der10)
fi_9der=eval(der9)
fi_8der=eval(der8)
fi_7der=eval(der7)
fi_6der=eval(der6)
fi_5der=eval(der5)
fi_4der=eval(der4)
fi_3der=eval(der3)
fi_2der=eval(der2)
fi_1der=eval(der1)
```

VII.1. Código de Matlab para funcionamiento de una RNA con 10 neuronas y 10 funciones de activación. *(Continuación)*

```
gradiente_10=2*e_10*fi_10der
b_7n=b_7+n_*gradiente_10
w_106n=w_106+n_*gradiente_10*y_6
w_107n=w_107+n_*gradiente_10*y_7
w_108n=w_108+n_*gradiente_10*y_8
w_109n=w_109+n_*gradiente_10*y_9

gradiente_9=fi_9der*gradiente_10*w_109
b_6n=b_6+n_*gradiente_9
w_95n=w_95+n_*gradiente_9*y_5

gradiente_8=fi_8der*gradiente_10*w_108
b_5n=b_5+n_*gradiente_8
w_84n=w_84+n_*gradiente_8*y_4

gradiente_7=fi_7der*gradiente_10*w_107
w_72n=w_72+n_*gradiente_7*y_2
w_73n=w_73+n_*gradiente_7*y_3

gradiente_6=fi_6der*gradiente_10*w_106
b_4n=b_4+n_*gradiente_6
w_61n=w_61+n_*gradiente_6*y_1

gradiente_5=fi_5der*gradiente_9*w_95
b_3n=b_3+n_*gradiente_5
w_53n=w_53+n_*gradiente_5*x_3
w_54n=w_54+n_*gradiente_5*x_4

gradiente_4=fi_4der*gradiente_8*w_84
w_42n=w_42+n_*gradiente_4*x_2
w_43n=w_43+n_*gradiente_4*x_3

gradiente_3=fi_3der*gradiente_7*w_73;
b_2n=b_2+n_*gradiente_3;
w_32n=w_32+n_*gradiente_3*x_2;
w_33n=w_33+n_*gradiente_3*x_3;

gradiente_2=fi_2der*gradiente_7*w_72;
b_1n=b_1+n_*gradiente_2;
w_21n=w_21+n_*gradiente_2*x_1;
w_22n=w_22+n_*gradiente_2*x_2;
w_23n=w_23+n_*gradiente_2*x_3;

gradiente_1=fi_1der*gradiente_6*w_61;
w_11n=w_11+n_*gradiente_1*x_1;
w_12n=w_12+n_*gradiente_1*x_2;

set(handles.gradiente10, 'String',eval(gradiente_10))
set(handles.gradiente9, 'String',eval(gradiente_9))
set(handles.gradiente8, 'String',eval(gradiente_8))
set(handles.gradiente7, 'String',eval(gradiente_7))
```

VII.1. Código de Matlab para funcionamiento de una RNA con 10 neuronas y 10 funciones de activación. *(Continuación)*

```
set(handles.gradiente6, 'String',eval(gradiente_6))
set(handles.gradiente5, 'String',eval(gradiente_5))
set(handles.gradiente4, 'String',eval(gradiente_4))
set(handles.gradiente3, 'String',eval(gradiente_3))
set(handles.gradiente2, 'String',eval(gradiente_2))
set(handles.gradiente1, 'String',eval(gradiente_1))
set(handles.b1n, 'String',eval(b_1n))
set(handles.b2n, 'String',eval(b_2n))
set(handles.b3n, 'String',eval(b_3n))
set(handles.b4n, 'String',eval(b_4n))
set(handles.b5n, 'String',eval(b_5n))
set(handles.b6n, 'String',eval(b_6n))
set(handles.b7n, 'String',eval(b_7n))
set(handles.w11n, 'String',eval(w_11n))
set(handles.w12n, 'String',eval(w_12n))
set(handles.w21n, 'String',eval(w_21n))
set(handles.w22n, 'String',eval(w_22n))
set(handles.w23n, 'String',eval(w_23n))
set(handles.w32n, 'String',eval(w_32n))
set(handles.w33n, 'String',eval(w_33n))
set(handles.w42n, 'String',eval(w_42n))
set(handles.w43n, 'String',eval(w_43n))
set(handles.w53n, 'String',eval(w_53n))
set(handles.w54n, 'String',eval(w_54n))
set(handles.w61n, 'String',eval(w_61n))
set(handles.w72n, 'String',eval(w_72n))
set(handles.w73n, 'String',eval(w_73n))
set(handles.w84n, 'String',eval(w_84n))
set(handles.w95n, 'String',eval(w_95n))
set(handles.w106n, 'String',eval(w_106n))
set(handles.w107n, 'String',eval(w_107n))
set(handles.w108n, 'String',eval(w_108n))
set(handles.w109n, 'String',eval(w_109n))

%**************************************************************************
*
%*                                 Salir
%**************************************************************************

function Salir_Callback(hObject, eventdata, handles)

opc=questdlg('¿Desea salír del programa?','SALIR','Si','No','No');
if strcmp(opc,'No')
return;
end
clear,clc,close all
%**************************************************************************
*
%*                             Nuevos Pesos
%**************************************************************************

function nuevos_pesos_Callback(hObject, eventdata, handles)
format short
```

VII.1. Código de Matlab para funcionamiento de una RNA con 10 neuronas y 10 funciones de activación. *(Continuación)*

```
syms v1 v2 v3 v4 v5 v6 v7 v8 v9 v10

n_=str2num(get(handles.n,'String'));
a_=str2num(get(handles.a,'String'));
x_1=str2num(get(handles.x1,'String'));
x_2=str2num(get(handles.x2,'String'));
x_3=str2num(get(handles.x3,'String'));
x_4=str2num(get(handles.x4,'String'));
y_d=str2num(get(handles.yd,'String'));
w_11=str2num(get(handles.w11,'String'));
w_12=str2num(get(handles.w12,'String'));
w_21=str2num(get(handles.w21,'String'));
w_22=str2num(get(handles.w22,'String'));
w_23=str2num(get(handles.w23,'String'));
w_32=str2num(get(handles.w32,'String'));
w_33=str2num(get(handles.w33,'String'));
w_42=str2num(get(handles.w42,'String'));
w_43=str2num(get(handles.w43,'String'));
w_53=str2num(get(handles.w53,'String'));
w_54=str2num(get(handles.w54,'String'));
w_61=str2num(get(handles.w61,'String'));
w_72=str2num(get(handles.w72,'String'));
w_73=str2num(get(handles.w73,'String'));
w_84=str2num(get(handles.w84,'String'));
w_95=str2num(get(handles.w95,'String'));
w_106=str2num(get(handles.w106,'String'));
w_107=str2num(get(handles.w107,'String'));
w_108=str2num(get(handles.w108,'String'));
w_109=str2num(get(handles.w109,'String'));
b_1=str2num(get(handles.b1,'String'));
b_2=str2num(get(handles.b2,'String'));
b_3=str2num(get(handles.b3,'String'));
b_4=str2num(get(handles.b4,'String'));
b_5=str2num(get(handles.b5,'String'));
b_6=str2num(get(handles.b6,'String'));
b_7=str2num(get(handles.b7,'String'));
fi_1=(get(handles.fi1,'String'));
fi_2=(get(handles.fi2,'String'));
fi_3=(get(handles.fi3,'String'));
fi_4=(get(handles.fi4,'String'));
fi_5=(get(handles.fi5,'String'));
fi_6=(get(handles.fi6,'String'));
fi_7=(get(handles.fi7,'String'));
fi_8=(get(handles.fi8,'String'));
fi_9=(get(handles.fi9,'String'));
fi_10=(get(handles.fi10,'String'));

der10=diff((fi_10),v10);
der9=diff((fi_9),v9);
der8=diff((fi_8),v8);
der7=diff((fi_7),v7);
der6=diff((fi_6),v6);
der5=diff((fi_5),v5);
```

VII.1. Código de Matlab para funcionamiento de una RNA con 10 neuronas y 10 funciones de activación. *(Continuación)*

```
der4=diff((fi_4),v4);
der3=diff((fi_3),v3);
der2=diff((fi_2),v2);
der1=diff((fi_1),v1);

der10=diff((fi_10),v10);
v1=(x_1*w_11)+(x_2*w_12);
y_1=eval(fi_1);
v2=(x_1*w_21)+(x_2*w_22)+(x_3*w_23)+b_1;
y_2=eval(fi_2);
v3=(x_2*w_32)+(x_3*w_33)+b_2;
y_3=eval(fi_3);
v4=(x_2*w_42)+(x_3*w_43);
y_4=eval(fi_4);
v5=(x_3*w_53)+(x_4*w_54)+b_3;
y_5=eval(fi_5);
v6=(y_1*w_61)+b_4;
y_6=eval(fi_6);
v7=(y_2*w_72)+(y_3*w_73);
y_7=eval(fi_7);
v8=(y_4*w_84)+b_5;
y_8=eval(fi_8);
v9=(y_5*w_95)+b_6;
y_9=eval(fi_9);
v10=(y_6*w_106)+(y_7*w_107)+(y_8*w_108)+(y_9*w_109)+b_7;
y_10=eval(fi_10);

e_10=(y_d)-y_10*der10;

fi_10der=eval(der10)
fi_9der=eval(der9)
fi_8der=eval(der8)
fi_7der=eval(der7)
fi_6der=eval(der6)
fi_5der=eval(der5)
fi_4der=eval(der4)
fi_3der=eval(der3)
fi_2der=eval(der2)
fi_1der=eval(der1)

gradiente_10=2*e_10*fi_10der
b_7n=b_7+n_*gradiente_10
w_106n=w_106+n_*gradiente_10*y_6
w_107n=w_107+n_*gradiente_10*y_7
w_108n=w_108+n_*gradiente_10*y_8
w_109n=w_109+n_*gradiente_10*y_9

gradiente_9=fi_9der*gradiente_10*w_109
b_6n=b_6+n_*gradiente_9
w_95n=w_95+n_*gradiente_9*y_5

gradiente_8=fi_8der*gradiente_10*w_108
```

VII.1. Código de Matlab para funcionamiento de una RNA con 10 neuronas y 10 funciones de activación. *(Continuación)*

```
b_5n=b_5+n_*gradiente_8

w_84n=w_84+n_*gradiente_8*y_4

gradiente_7=fi_7der*gradiente_10*w_107
w_72n=w_72+n_*gradiente_7*y_2
w_73n=w_73+n_*gradiente_7*y_3

gradiente_6=fi_6der*gradiente_10*w_106
b_4n=b_4+n_*gradiente_6
w_61n=w_61+n_*gradiente_6*y_1

gradiente_5=fi_5der*gradiente_9*w_95
b_3n=b_3+n_*gradiente_5
w_53n=w_53+n_*gradiente_5*x_3
w_54n=w_54+n_*gradiente_5*x_4

gradiente_4=fi_4der*gradiente_8*w_84
w_42n=w_42+n_*gradiente_4*x_2
w_43n=w_43+n_*gradiente_4*x_3

gradiente_3=fi_3der*gradiente_7*w_73;
b_2n=b_2+n_*gradiente_3;
w_32n=w_32+n_*gradiente_3*x_2;
w_33n=w_33+n_*gradiente_3*x_3;

gradiente_2=fi_2der*gradiente_7*w_72;
b_1n=b_1+n_*gradiente_2;
w_21n=w_21+n_*gradiente_2*x_1;
w_22n=w_22+n_*gradiente_2*x_2;
w_23n=w_23+n_*gradiente_2*x_3;

gradiente_1=fi_1der*gradiente_6*w_61;
w_11n=w_11+n_*gradiente_1*x_1;
w_12n=w_12+n_*gradiente_1*x_2;

set(handles.b1, 'String',eval(b_1n))
set(handles.b2, 'String',eval(b_2n))
set(handles.b3, 'String',eval(b_3n))
set(handles.b4, 'String',eval(b_4n))
set(handles.b5, 'String',eval(b_5n))
set(handles.b6, 'String',eval(b_6n))
set(handles.b7, 'String',eval(b_7n))
set(handles.w11, 'String',eval(w_11n))
set(handles.w12, 'String',eval(w_12n))
set(handles.w21, 'String',eval(w_21n))
set(handles.w22, 'String',eval(w_22n))
set(handles.w23, 'String',eval(w_23n))
set(handles.w32, 'String',eval(w_32n))
set(handles.w33, 'String',eval(w_33n))
set(handles.w42, 'String',eval(w_42n))
set(handles.w43, 'String',eval(w_43n))
```

VII.1. Código de Matlab para funcionamiento de una RNA con 10 neuronas y 10 funciones de activación. *(Continuación)*

```
set(handles.w53, 'String',eval(w_53n))
set(handles.w54, 'String',eval(w_54n))
set(handles.w61, 'String',eval(w_61n))
set(handles.w72, 'String',eval(w_72n))
set(handles.w73, 'String',eval(w_73n))
set(handles.w84, 'String',eval(w_84n))
set(handles.w95, 'String',eval(w_95n))
set(handles.w106, 'String',eval(w_106n))
set(handles.w107, 'String',eval(w_107n))
set(handles.w108, 'String',eval(w_108n))
set(handles.w109, 'String',eval(w_109n))
```