UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO

FACULTAD DE ESTUDIOS SUPERIORES ARAGON

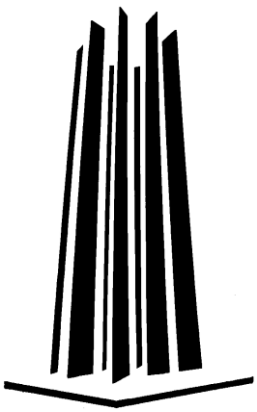
CONTROL DE ROBOTS MEDIANTE LA APLICACIÓN DE REDES NEURONALES ARTIFICIALES PROGRAMADAS EN VHDL

TESIS

QUE PARA OBTENER EL TITULO DE
INGENIERO EN COMPUTACIÓN
PRESENTA:

JORGE ABRAHAM MORALES DORANTES

ASESOR:
M. EN I. DAVID GONZALEZ MAXINEZ



SAN JUAN DE ARAGON, MEXICO 2010

ÍNDICE

INTRODUCCIÓN

A lo largo de los últimos años, la tecnología ha ido avanzando a pasos agigantados, despertando con esto el interés de las personas, de manera tal que ahora ya es casi imposible que en nuestra vida no estén involucradas las computadoras o los celulares ya que se han convertido en herramientas indispensables para tener una vida mas fácil y cómoda.

En esta cuestión un tema que esta causando mucho interés en las personas y sobre todo en las nuevas generaciones es la Robótica y dentro de este ramo los investigadores están muy interesados en la inteligencia artificial, solo que aquí hay un problema por que la mayoría de los investigadores han hecho desarrollos de este tema orientado al software, y cuanto la aplicación a la robótica, resulta muy caro ya que se utilizan microprocesadores muy sofisticados y por ende muy costosos, además que requieren un sinfín de herramientas para la adecuada utilización de estos dispositivos.

Las Redes Neuronales son un campo muy importante dentro de la Inteligencia Artificial. Inspirándose en el comportamiento conocido del cerebro humano (principalmente el referido a las neuronas y sus conexiones), trata de crear modelos artificiales que solucionen problemas difíciles de resolver mediante técnicas algorítmicas convencionales.

Esta tecnología puede ser desarrollada tanto en software como en hardware y con ella se pueden construir sistemas capaces de aprender, de adaptarse a condiciones variantes o inclusive si se dispone de una colección suficiente grande de datos, predecir el estado futuro de algunos modelos. Estas técnicas son adecuadas para enfrentar problemas que hasta ahora eran resueltos sólo por el cerebro humano y resultaban difíciles o imposibles para las máquinas lógicas secuenciales. Un procesamiento paralelo realizado por un gran número de elementos altamente interconectados, es la clave de su funcionamiento.

Las Redes Neuronales (Neural Networks) son utilizadas para la predicción, la minería de datos (data mining), el reconocimiento de patrones y los sistemas de control adaptativo. Constituyen una parte muy importante en el estudio y desarrollo de la inteligencia artificial (AI) y el de la vida artificial (a-life). Las RN pueden ser combinadas con otras herramientas como la lógica difusa (lógica fuzzy), los algoritmos genéticos, los sistemas expertos, las estadísticas, las transformadas de Fourier, etc.

Las redes neuronales como su nombre lo indica pretenden imitar a pequeñísima escala la forma de funcionamiento de las neuronas que forman el cerebro humano. Todo el desarrollo de las redes neuronales tiene mucho que ver con la neurofisiología, no en vano se trata de imitar a una neurona humana con la mayor exactitud posible. Entre los pioneros en el modelado de neuronas se encuentra Warren McCulloch y Walter Pitts. Estos dos investigadores propusieron un modelo matemático de neurona. En este modelo cada

neurona estaba dotada de un conjunto de entradas y salidas. Cada entrada está afectada por un peso. La activación de la neurona se calcula mediante la suma de los productos de cada entrada y la salida es una función de esta activación. La principal clave de este sistema se encuentra en los pesos de las diferentes entradas. Como se ha visto, las entradas son modificadas por el peso y las salidas son función de estas modificaciones. Esto nos lleva a concluir que los pesos influyen de forma decisiva en la salida y por lo tanto pueden ser utilizados para controlar la salida que se desea.

En realidad cuando se tienen interconectadas muchas de estas neuronas artificiales lo que se hace inicialmente es entrenar el sistema. El entrenamiento consiste en aplicar unas entradas determinadas a la red y observar la salida que produce. Si la salida que produce no se adecua a la que se esperaba, se ajustan los pesos de cada neurona para interactivamente ir obteniendo las respuestas adecuadas del sistema. A la red se le somete a varios ejemplos representativos, de forma que mediante la modificación de los pesos de cada neurona, la red va "aprendiendo".

Este trabajo esta orientado al desarrollo del control inteligente para un robot por medio de redes neuronales en un dispositivo lógico programable utilizando el lenguaje de programación de VHDL. De la misma forma durante este trabajo conoceremos el comportamiento de una red neuronal y sus unidades (la neurona) así como sus características.

CAPÍTULO 1.

INTRODUCCIÓN A LAS REDES NEURONALES ARTIFICIALES

Para poder realizar cualquier trabajo que involucre el tomar las características de algún elemento de de la naturaleza para replicarlo de forma artificial de modo que usemos esas características para el beneficio de nuestro trabajo es sumamente necesario el conocer de manera detallada el elemento que ocuparemos para el desarrollo de nuestro trabajo, de forma tal que este primer capitulo se centra en darnos la información necesaria para conocer el comportamiento de las neuronas y las redes neuronales.

1.1 La Neurona Biológica

A finales del siglo XIX se logró una mayor claridad sobre el trabajo del cerebro debido a los trabajos de Ramón y Cajal en España y Sherrington en Inglaterra. El primero trabajó en la anatomía de las neuronas y el segundo en los puntos de conexión de las mismas o *sinapsis*. Se estima que en cada milímetro del cerebro hay cerca de 50.000 neuronas, conteniendo en total más de cien mil millones de neuronas y sinapsis en el sistema nervioso humano.

El tamaño y la forma de las neuronas es variable, pero con las mismas subdivisiones que muestra la **figura 1.1** de la estructura de la neurona. Subdividiéndose así en tres partes:

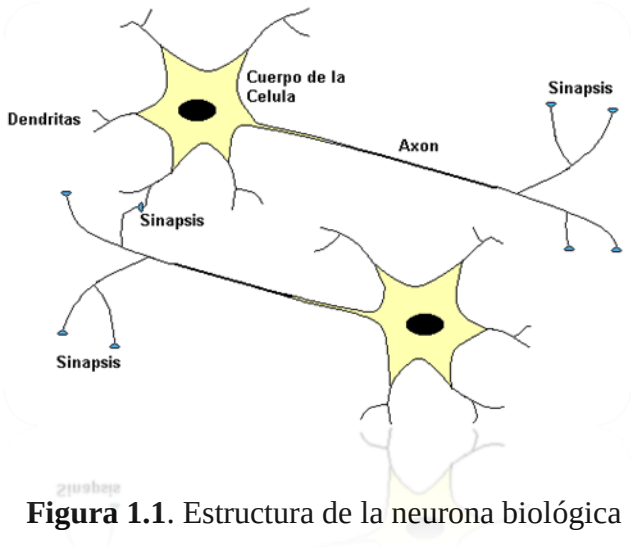


Figura 1.1. Estructura de la neurona biológica

1. el cuerpo de la neurona,
2. ramas de extensión llamadas dendritas para recibir las entradas, y
3. un axón que lleva la salida de la neurona a las dendritas de otras neuronas.

El *cuerpo* de la neurona o *Soma* contiene el *núcleo*. Se encarga de todas las actividades metabólicas de la neurona y recibe la información de otras neuronas vecinas a través de las conexiones sinápticas (algunas neuronas se comunican solo con las cercanas, mientras que otras se conectan con miles).

Las dendritas, parten del soma y tienen ramificaciones. Se encargan de la recepción de señales de las otras células a través de conexiones llamadas *sinápticas*. Si pensamos, desde ahora, en términos electrónicos podemos decir que las dendritas son las conexiones de entrada de la neurona. Por su parte el *axón* es la "salida" de la neurona y se utiliza para enviar impulsos o señales a otras células nerviosas.
Cuando el axón está cerca de sus células destino se divide en muchas ramificaciones que forman sinapsis con el soma o axones de otras células. Esta unión puede ser "inhibidora" o "excitadora" según el transmisor que las libere. Cada neurona recibe de 10.000 a 100.000 sinapsis y el axón realiza una cantidad de conexiones similar.

La transmisión de una señal de una célula a otra por medio de la sinapsis es un proceso químico. En el se liberan substancias transmisoras en el lado del emisor de la unión.
El efecto es elevar o disminuir el potencial eléctrico dentro del cuerpo de la célula receptora.
Si su potencial alcanza el umbral se envía un pulso o potencial de acción por el axón. Se dice, entonces, que la célula se disparó. Este pulso alcanza otras neuronas a través de las distribuciones de los axones.

Una neurona se puede comparar con una caja negra compuesta por varias entradas y una salida. La relación de activación entre la salida y la entrada, la función de transferencia la podemos ver en la figura 1.2.

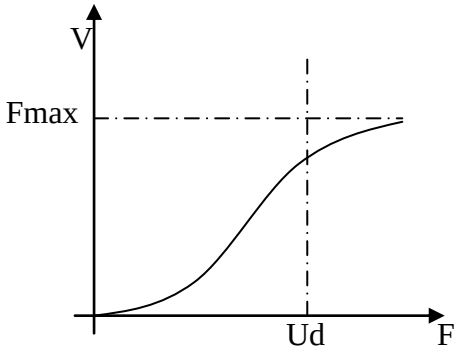


Figura 1.2. Función de transferencia de una neurona

La variable **f** es la frecuencia de activación o emisión de potenciales y **u** es la intensidad del estímulo del soma.

1.2 La Neurona Artificial

Un circuito eléctrico que realice la suma ponderada de las diferentes señales que recibe de otras unidades iguales y produzca en la salida un **uno** o un **cero** según el resultado de la suma con relación al umbral o nivel de disparo, conforma una buena representación de lo que es una *neurona artificial*.

Más simple se puede decir que una neurona artificial es un elemento con entradas, salida y memoria que puede ser realizada mediante software o hardware. En la figura 1.3 se ve que posee entradas (I) que son ponderadas (w), sumadas y comparadas con un umbral (t).
La señal computada de esa manera, es tomada como argumento para una función no lineal (f), la cual determinará el tipo de neurona y en parte el tipo de red neuronal.

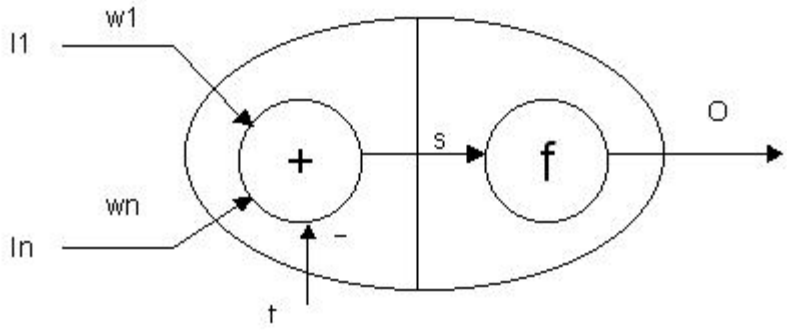


Figura 1.3. Esquema Simplificado de una Neurona artificial.

La función de transferencia para la activación o disparo de la neurona puede ser de *umbral semilineal o lógico,* de *limitación dura o signo y* de *función sigmoidal o sigmoidea (tipo s*), tal y como se muestra en la figura 1.4.

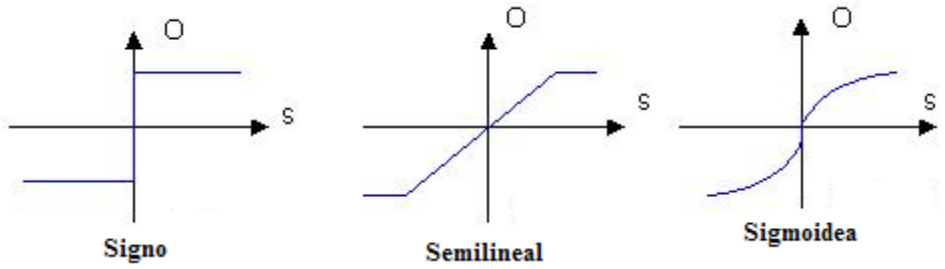


Figura 1.4. Graficas de la función de transferencia para la activación de la neurona.

En particular, la función *sigmoidal* se define así en la ecuación (1).

$$f(u_i) = \frac{1}{1 + e^{\frac{-U_i}{\sigma}}} \qquad \ldots\ldots\ldots\ldots\ldots\ldots \quad (1)$$

La respuesta la produce el circuito activo o función de transferencia que forma parte del cuerpo de la neurona. Las "dendritas" llevan las señales eléctricas al cuerpo de la misma. Estas señales provienen de sensores o son salidas de neuronas vecinas.

Las señales por las dendritas pueden ser voltajes positivos o negativos; los voltajes positivos contribuyen a la excitación del cuerpo y los voltajes negativos contribuyen a inhibir la respuesta de la neurona como se muestra en la figura 1.5.

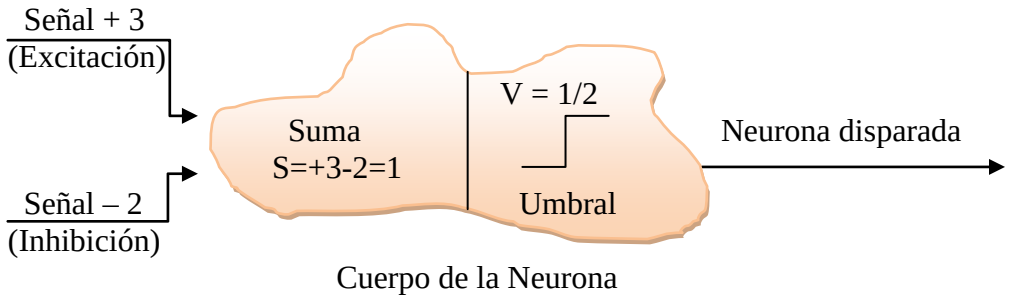


Figura 1.5. Excitación, Inhibición y disparo de la neurona

A continuación podemos apreciar en la figura 1.6 la similitud entre una neurona biológica y una artificial.

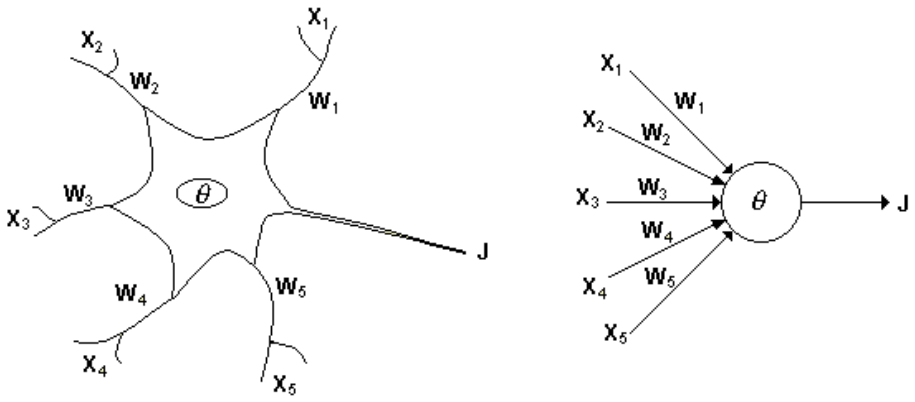


Figura 1.6. Similitud entre una neurona biológica y una artificial

1.3 Red Neuronal Biológica

El sistema de neuronas biológico está compuesto por neuronas de entrada (sensores) conectados a una compleja red de neuronas "calculadoras" (neuronas ocultas), las cuales, a su vez, están conectadas a las neuronas de salidas que controlan, por ejemplo, los músculos, este sistema lo podemos visualizar en la figura 1.7.

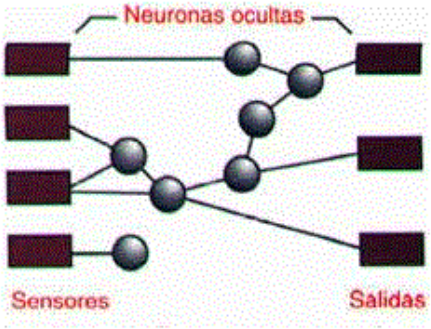


Figura 1.7. Sistema de neuronas biológico

Estructura neuronal

Los sensores pueden ser señales de los oídos, ojos, etc. Las respuestas de las neuronas de salida activan los músculos correspondientes. En el cerebro hay una gigantesca red de neuronas "calculadoras" u ocultas que realizan la computación necesaria. De esta manera similar, una red neuronal artificial debe ser compuesta por sensores del tipo mecánico o electrónico.

La distribución de neuronas dentro de la red se realiza formando niveles o capas de un número determinado de neuronas cada una, y que existen capas de entrada, de salida, y ocultas, ahora veamos las formas de conexión entre neuronas. Cuando ninguna salida de las neuronas es entrada de neuronas del mismo nivel o de niveles precedentes, la red se describe como de propagación hacia adelante. Cuando las salidas pueden ser conectadas como entradas de neuronas de niveles previos o del mismo nivel, incluyéndose ellas mismas, la red es de propagación hacia atrás. [1]

CAPÍTULO 2.

REDES NEURONALES ARTIFICIALES

Desde la década de los 40, en la que nació y comenzó a desarrollarse la informática, el modelo neuronal la ha acompañado. De hecho, la aparición de los computadores digitales y el desarrollo de las teorías modernas acerca del aprendizaje y del procesamiento neuronal se produjeron aproximadamente al mismo tiempo, a finales de los años cuarenta.

Desde entonces hasta nuestros días, la investigación neurofisiológica y el estudio de sistemas neuronales artificiales (ANS, Artificial Neural Systems) han ido de la mano. Sin embargo, los modelos de ANS no se centran en la investigación neurológica, si no que toma conceptos e ideas del campo de las ciencias naturales para aplicarlos a la resolución de problemas pertenecientes a otras ramas de las ciencias y la ingeniería.

Podemos decir que la tecnología ANS incluye modelos *inspirados* por nuestra comprensión del cerebro, pero que no tienen por qué ajustarse exactamente a los modelos derivados de dicho entendimiento. Los primeros ejemplos de estos sistemas aparecen al final de la década de los cincuenta. La referencia histórica más corriente es la que alude al trabajo realizado por Frank Rosenblatt en un dispositivo denominado *perceptrón*.

Durante todos estos años, la tecnología ANS no siempre ha tenido la misma consideración en las ramas de la ingeniería y las ciencias de la computación, más ansiosas de resultados que las ciencias neuronales. A partir de 1969, el pesimismo debido a las limitadas capacidades del perceptrón hizo languidecer este tipo de investigación.

A principios de los 80, por un lado Hopfield y sus conferencias acerca de la memoria autoasociativa y por otro lado la aparición del libro *Parallel Distributed Processing* (PDP), escrito por Rumelhart y McClelland reactivaron la investigación en el campo de las redes neuronales. Hubo grandes avances que propiciaron el uso comercial en campos tan variados como el diagnóstico de enfermedades, la aproximación de funciones o el reconocimiento de imágenes.

2.1 ¿Qué es?

Es un sistema compuesto por un gran número de elementos básicos (Artificial Neurons), agrupados en capas (Layers) y que se encuentran altamente interconectados de manera tal, que las salidas de una capa están completamente conectadas a las entradas de la capa siguiente; en este caso decimos que tenemos una red completamente conectada.

Es posible tener redes en las cuales sólo algunas de las unidades están conectadas, también podemos tener conexiones de realimentación, conectando algunas salidas hacia entradas en capas anteriores tal como se muestra en la figura 2.1

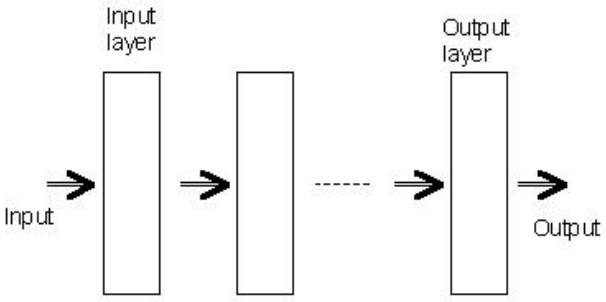


Figura 2.1. Conexión de capas Referencias

Para obtener un resultado aceptable, el número de capas debe ser por lo menos tres. No existen evidencias, de que una red con cinco capas resuelva un problema que una red de cuatro capas no pueda. Usualmente se emplean tres o cuatro capas.

Estos sistemas emulan, de una cierta manera, al cerebro humano. Requieren aprender a comportarse (Learning) y alguien debe encargarse de enseñarles o entrenarles (Training), en base a un conocimiento previo del entorno del problema.

Las redes de neuronas artificiales (denominadas habitualmente como RNA o en inglés como: "ANN" (Artificial Neural Networks)) son un paradigma de aprendizaje y procesamiento automático inspirado en la forma en que funciona el sistema nervioso de los animales. Se trata de un sistema de interconexión de neuronas en una red que colabora para producir un estímulo de salida. El aprendizaje se puede dar:

Supervisado.- mediante este tipo se introduce a la red una serie de patrones de entrada y salida. La red es capaz de ajustar los pesos con el fin de memorizar la salida deseada.

No supervisado.- aquí la red responde clasificando los patrones de entrada en función de las características mas adecuadas de cada uno.

Auto supervisado.- en este tipo la propia red corrige los errores en la interpretación a través de una realimentación.

El entrenamiento de la red es muy importante ya que servirá para que posteriormente la respuesta del sistema sea la adecuada. Si nos fijamos un poco eso tiene mucho que ver con el aprendizaje humano. Cuando a un niño se le ordena tomar un vaso, empieza moviendo el brazo de forma cuasi-aleatoria hasta que choca con el vaso y lo presiona con sus dedos. La próxima vez que se le ordene al niño, éste alcanzará el vaso con mayor soltura y precisión.
Este mismo modelo se ha ensayado en redes neuronales de características similares a las del niño. Una vez que el brazo mecánico choca con la pieza y memoriza la secuencia, en posteriores ocasiones al brazo le cuesta menos realizar la misma operación se dice entonces que el sistema adquirió experiencia.

2.2 ¿Para que sirven?

Una de las misiones en una red neuronal consiste en simular las propiedades observadas en los sistemas neuronales biológicos a través de modelos matemáticos recreados mediante mecanismos artificiales (como un circuito integrado, un ordenador o un conjunto de válvulas). El objetivo es conseguir que las máquinas den respuestas similares a las que es capaz el cerebro que se caracterizan por su generalización y su robustez.

Esta tecnología es muy útil en unos pocos y muy especiales problemas. A grandes rasgos, estas aplicaciones son aquellas en las cuales se dispone de un registro de datos y nadie sabe exactamente la estructura y los parámetros que pudieran modelar el problema. Grandes cantidades de datos y mucha incertidumbre en cuanto a la manera de como estos son producidos.

Como ejemplos de las aplicaciones de las redes neuronales se pueden citar: las variaciones en la bolsa de valores, los riesgos en préstamos, el clima local, el reconocimiento de patrones en rostros y la minería de datos.

En conclusión las redes neuronales se orientan a desarrollar máquinas o sistemas inteligentes capaces de simular, desarrollar y optimizar muchas de las funciones de un ser humano así como también la investigación científica ya que pueden hacer cosas que el hombre por sus limitaciones físicas no puede realizar.

2.3 ¿Cómo funcionan?

Una red neuronal se compone de unidades llamadas neuronas. Cada neurona recibe una serie de entradas a través de interconexiones y emite una salida. Esta salida viene dada por tres funciones:

1. Una función de propagación (también conocida como función de excitación), que por lo general consiste en el sumatorio de cada entrada multiplicada por el peso de su interconexión (valor neto). Si el peso es positivo, la conexión se denomina *excitatoria*; si es negativo, se denomina *inhibitoria*.

2. Una función de activación, que modifica a la anterior. Puede no existir, siendo en este caso la salida la misma función de propagación.

3. Una función de transferencia, que se aplica al valor devuelto por la función de activación. Se utiliza para acotar la salida de la neurona y generalmente viene dada por la interpretación que queramos darle a dichas salidas. Algunas de las más utilizadas son la sigmoidea (para obtener valores en el intervalo [0,1]) y la hiperbólica-tangente (para obtener valores en el intervalo [-1,1]).

En la figura 2.2 podemos observar el modelo del perceptrón con 2 entradas.

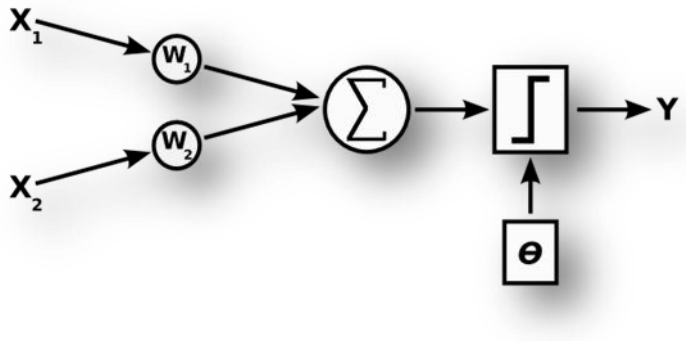


Figura 2.2. Perceptrón con 2 entradas

2.4 ¿Cómo es su arquitectura?

La arquitectura es la estructura o patrón de conexiones de la red. Es conveniente recordar que las conexiones sinápticas son direccionales, es decir, la información sólo se transmite en un sentido.
En general, las neuronas suelen agruparse en unidades estructurales llamadas *capas*. Dentro de una capa, las neuronas suelen ser del mismo tipo. Se pueden distinguir tres tipos de capas:

- De *entrada*: reciben datos o señales procedentes del entorno.

- De *salida*: proporcionan la respuesta de la red a los estímulos de la entrada.

- *Ocultas*: no reciben ni suministran información al entorno (procesamiento interno de la red).

Generalmente las conexiones se realizan entre neuronas de distintas capas, pero puede haber conexiones intracapa o *laterales* y conexiones de *realimentación* que siguen un sentido contrario al de entrada-salida, en la figura 2.3 podemos ver un ejemplo de esto.

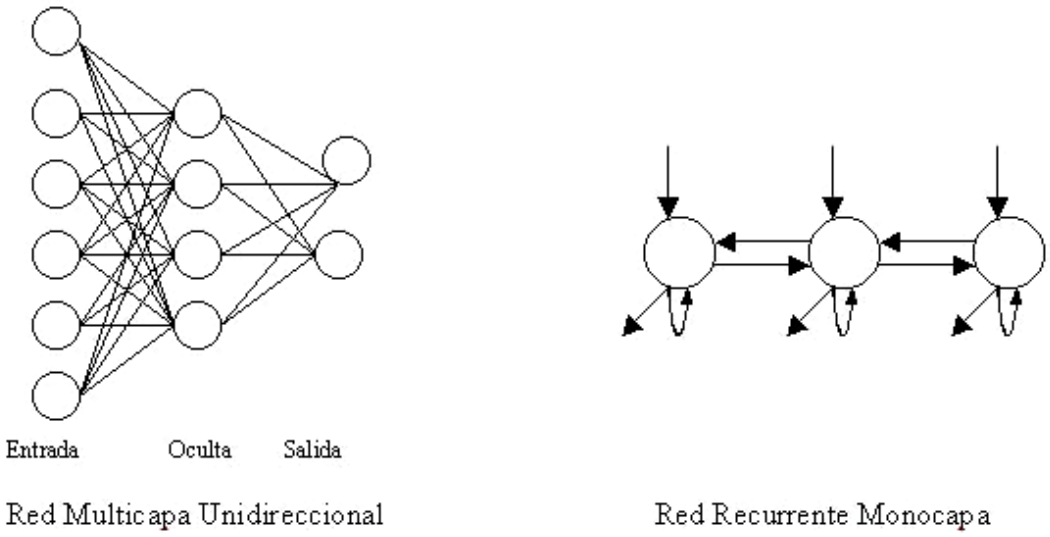


Figura 2.3. Capas de las neuronas

2.5 ¿Cómo se clasifican?

Los distintos modelos de red neuronal pueden clasificarse de acuerdo con cuatro criterios básicos: (1) la *naturaleza* de las señales de entrada y salida, (2) la *topología* de la red, (3) el *mecanismo de aprendizaje* que utilizan y (4) el *tipo de asociación* de las señales de entrada y salida y la forma de representar estas señales.
Las distintas posibilidades de presentarse estos aspectos junto con las distintas funciones de activación y transferencia nos permiten la clasificación de los distintos modelos.

De Acuerdo Con Su Naturaleza

De acuerdo con la *naturaleza* de las señales de entrada y de salida podemos clasificar las redes neuronales en *analógicas, discretas* (generalmente, binarias) e *híbridas*:

Las *redes analógicas* procesan datos de entrada de naturaleza analógica, valores reales continuos, habitualmente acotados y usualmente en el compacto [-1,1] o en el [0,1], para dar respuestas también continuas. Las redes analógicas suelen presentar funciones de activación continuas, habitualmente lineales o sigmoides.

Entre estas redes neuronales destacan las redes de Backpropagation , la red continua de Hopfield, la de Contrapropagación, la Memoria Lineal Asociativa, la Brain-State-in-Box, y los modelos de Kohonen (mapas auto-organizados (S.O.M.) y Learning Vector Quantizer, (L.V.Q.) .

Las *redes discretas* (binarias) procesan datos de naturaleza discreta, habitualmente {0,1}, para acabar emitiendo una respuesta discreta. Entre las redes binarias destacan la Maquina de Boltzman, la Máquina de Cauchy, la red discreta de Hopfield, el Cognitrón y el Neogognitrón.

Las *redes híbridas*, procesan entradas analógicas para dar respuestas binarias, entre ellas destacan el Perceptrón, la red Adaline y la Madaline.

En la figura 2.3 podemos observar una clasificación de modelos de redes neuronales.

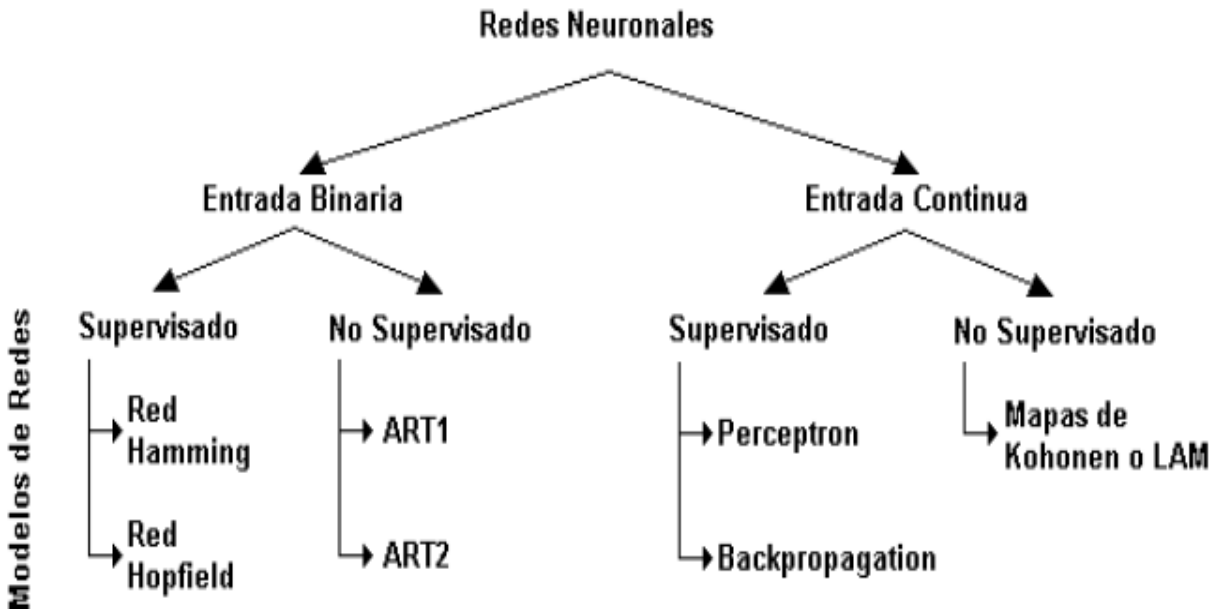


Figura 2.3. Clasificación de las Redes Neuronales

De Acuerdo Con Su Topología

Por lo que hace a la *topología* de la red, las redes pueden clasificarse de acuerdo con el número de capas o niveles de neuronas, el número de neuronas por capa y el grado y tipo de conectividad entre las mismas. La primera distinción a establecer es entre las redes *Monocapa* y las *Multicapa*.

Las *redes Monocapa* sólo cuentan con una capa de neuronas, que intercambian señales con el exterior y que constituyen a un tiempo la entrada y salida del sistema. En las redes Monocapa (red de Hopfield o red Brain-State-in-Box, máquina de Boltzman, máquina de Cauchy), se establecen conexiones laterales entre las neuronas, pudiendo existir, también conexiones autor recurrentes (la salida de una neurona se conecta con su propia entrada), como en el caso del modelo Brain-State-in Box.

Las *redes Multicapa* disponen de conjuntos de neuronas jerarquizadas en distintos niveles o capas, con al menos una capa de entrada y otra de salida, y, eventualmente una o varias capas intermedias (ocultas).
Normalmente todas las neuronas de una capa reciben señales de otra capa anterior y envían señales a la capa posterior (en el sentido Entrada - Salida). A estas conexiones se las conoce como conexiones hacia delante feedforward. Si una red sólo dispone de conexiones de este tipo se la conoce como red feedforward.

Sin embargo, puede haber redes en las que algunas de sus neuronas presenten conexiones con neuronas de capas anteriores, conexiones hacia atrás o *feedback*. En tal caso hablaremos de una red feedback o *interactiva*. Entre las primeras destacan los distintos modelos de Kohonen, aunque presentan conexiones laterales y autorrecurrrentes, el Perceptrón (multicapa) o M.L.P., las redes Adaline y Madaline, la Memoria Lineal Adaptativa y las Backpropagation.

Entre las segundas debemos mencionar el Cognitrón y el Neocognitrón, junto con los modelos de Resonancia y las máquinas multicapa de Boltzman y Cauchy.

2.6 ¿Cómo se construyen?

Se pueden realizar de varias maneras. Por ejemplo en hardware utilizando transistores a efecto de campo (FET) o amplificadores operacionales, pero la mayoría de las RN se construyen en software, esto es en programas de computación.

Existen muy buenas y flexibles herramientas disponibles en internet que pueden simular muchos tipos de neuronas, conexiones sinápticas (Synapses) y estructuras.

Aspectos a considerar en la red neuronal:

Elemento básico. Neurona artificial.
Pueden ser con salidas binarias, análogas o con codificación de pulsos (PCM). Es la unidad básica de procesamiento que se conecta a otras unidades a través de conexiones sinápticas (Synaptic Connection).

La estructura de la red. La interconexión de los elementos básicos.
Es la manera como las unidades básicas se interconectan.
Un ejemplo de como se combinan los aspectos mencionados es el perceptrón multicapa (Multi-Layer Perceptrón), entrenado con el algoritmo del "backpropagation". Se trata de una red compuesta por varios estratos de neuronas con respuestas basadas en funciones exponenciales, y cuyas conexiones sinápticas se determinan de manera de minimizar un error cuadrático medio.

Otro ejemplo es el mapa auto-organizado de Kohonen (Self-organizing Map), en el cual sólo se requiere conocer la salida del sistema.

2.7 ¿Cómo aprenden?

Este proceso consiste en una adaptación progresiva de los valores de las conexiones sinápticas (Synaptic Connections), para permitir a la Red Neuronal (Neural Network) el aprendizaje de un comportamiento deseado. Para lograr esto, alimentamos a la red con una entrada de los datos de entrenamiento, comparamos la salida de la red con la salida de los datos de entrenamiento; la diferencia se usa para computar el error (cuadrático medio) de la respuesta de la red.

Con un algoritmo apropiado (como el "Backpropagation") es posible retocar los valores de los pesos sinápticos con el fin de reducir el error. Estas correcciones deben realizarse varias veces o ciclos, para todo el conjunto de entradas-salidas de los datos de entrenamiento.
El proceso por el que una RNA actualiza los pesos se muestra en la figura 2.4

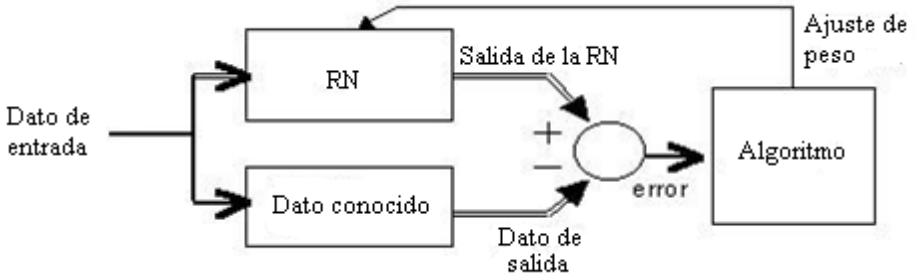


Figura 2.4. Proceso de aprendizaje de una Red Neuronal

Una RNA actualiza los pesos (y, en algunos casos, la arquitectura) con el propósito de que la red pueda llevar a cabo de forma efectiva una tarea determinada.

Hay tres conceptos fundamentales en el aprendizaje:
Paradigma de aprendizaje: información de la que dispone la red.
Regla de aprendizaje: principios que gobiernan el aprendizaje.
Algoritmo de aprendizaje: procedimiento numérico de ajuste de los pesos.

Existen dos paradigmas fundamentales de aprendizaje:

Supervisado: la red trata de minimizar un error entre la salida que calcula y la salida deseada (conocida), de modo que la salida calculada termine siendo la deseada.

No supervisado o auto organizado: la red conoce un conjunto de patrones sin conocer la respuesta deseada. Debe extraer rasgos o agrupar patrones similares.

En cuanto a los algoritmos de aprendizaje, tenemos cuatro tipos:

Minimización del error: reducción del gradiente, retropropagación, etc. La modificación de pesos está orientada a que el error cometido sea mínimo.

Boltzmann: para redes estocásticas, donde se contemplan parámetros aleatorios.

Hebb: cuando el disparo de una célula activa otra, el peso de la conexión entre ambas tiende a reforzarse (Ley de Hebb).

Competitivo: sólo aprenden las neuronas que se acercan más a la salida deseada.

Los algoritmos, y en general el proceso de aprendizaje, son complejos y suelen llevar bastante tiempo computacionalmente hablando. Su ventaja es que una vez ha aprendido, la red puede congelar sus pesos y funcionar en modo *recuerdo* o *ejecución.*

Ejemplo de aprendizaje de una red Adaline

El "Elemento Lineal Adaptable", también llamado Adaline (primeramente conocido como Neurona Lineal Adaptable), fue sugerido por Widrow y Hoff en su obra "Adaptive switching circuits". En una simple implementación física, la cual es Ejemplo de aplicación a una compuerta logica OR

1. inicialmente la neurona toma los siguientes pesos

$$W_n = \begin{bmatrix} 1.5 \\ 0.5 \\ 1.5 \end{bmatrix}$$

Y tomando una función de activación en escalón de un solo polo {0 1}

a. iniciamos calculando el valor ponderado de la suma de las entradas por los pesos en la iteración 1 (k=1)

$$S^1 = \sum_{i=0}^{\infty} X_i \bullet W_i \Rightarrow S^1 = 0*1.5 + 0*0.5 + 1*1.5 = 1.5$$

b. Luego se compara el valor con la función de activación

$$\sigma\begin{cases} S > 0 \Rightarrow Y = 1 \\ S < 0 \Rightarrow Y = 0 \end{cases} \Leftrightarrow Y = 1$$

c. Calculando el error

$e^K = D - Y$ Tenemos que $e^K = 0 - 1 = -1$

d. Los nuevos pesos quedan

$$W_n^{1+1} = \begin{bmatrix} W_0^1 \\ W_1^1 \\ W_2^1 \end{bmatrix} + \alpha \bullet \hat{e^K} \bullet \begin{bmatrix} X_0^1 \\ X_1^1 \\ X_2^1 \end{bmatrix}$$

$$W_n^2 = \begin{bmatrix} 1.5 \\ 0.5 \\ 1.5 \end{bmatrix} + 1 \bullet -1 \bullet \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 0.5 \\ 0.5 \\ 1.5 \end{bmatrix}$$

1. a. Se inicia una nueva iteración (k=2), hallamos de nuevo S

$$S^2 = 1 * 0.5 + 0 * 0.5 + 1 * 1.5 = 2.5$$

b. La comparación con la función de activación

$$\sigma\begin{cases} S > 0 \Rightarrow Y = 1 \\ S < 0 \Rightarrow Y = 0 \end{cases} \Leftrightarrow Y = 1$$

c. El error es

$e^K = D - Y$ Tenemos que $e^K = 1 - 1 = 0$

c. El nuevo calculo de los pesos no se ve alterado por e = 0

$$W_n^{2+1} = \begin{bmatrix} W_0^2 \\ W_1^2 \\ W_2^2 \end{bmatrix} + \alpha \bullet \hat{e}^K \bullet \begin{bmatrix} X_0^2 \\ X_1^2 \\ X_2^2 \end{bmatrix}$$

$$W_n^3 = \begin{bmatrix} 0.5 \\ 0.5 \\ 1.5 \end{bmatrix} + 0 = \begin{bmatrix} 0.5 \\ 0.5 \\ 1.5 \end{bmatrix}$$

2. a. Se inicia una nueva iteración (k = 3)

$$S^3 = 1*0.5 + 1*0.5 + 0*1.5 = 1$$

b. La comparación con la función de activación

$$\sigma \begin{cases} S > 0 \Rightarrow Y = 1 \\ S < 0 \Rightarrow Y = 0 \end{cases} \Leftrightarrow Y = 1$$

c. El error es

$e^K = D - Y$ Tenemos que $e^K = 1 - 1 = 0$

d. El nuevo calculo de los pesos no se ve alterado por e = 0

$$W_n^{3+1} = \begin{bmatrix} W_0^3 \\ W_1^3 \\ W_2^3 \end{bmatrix} + \alpha \bullet \hat{e}^K \bullet \begin{bmatrix} X_0^3 \\ X_1^3 \\ X_2^3 \end{bmatrix}$$

$$W_n^{\,4} = \begin{bmatrix} 0.5 \\ 0.5 \\ 1.5 \end{bmatrix} + 0 = \begin{bmatrix} 0.5 \\ 0.5 \\ 1.5 \end{bmatrix}$$

4. a. Iteración con k = 4

$$S^4 = 1*0.5 + 1*0.5 + 1*1.5 = 2.5$$

b. La comparación con la función de activación

$$\sigma \begin{cases} S > 0 \Rightarrow Y = 1 \\ S < 0 \Rightarrow Y = 0 \Leftrightarrow Y = 1 \end{cases}$$

c. El error es

$e^K = D - Y$ Tenemos que $e^K = 1 - 1 = 0$

d. El nuevo calculo de los pesos no se ve alterado por e = 0

$$W_n^{\,4+1} = \begin{bmatrix} W_0^4 \\ W_1^4 \\ W_2^4 \end{bmatrix} + \alpha \bullet e^{\hat{K}} \bullet \begin{bmatrix} X_0^4 \\ X_1^4 \\ X_2^4 \end{bmatrix}$$

$$W_n^{\,5} = \begin{bmatrix} 0.5 \\ 0.5 \\ 1.5 \end{bmatrix} + 0 = \begin{bmatrix} 0.5 \\ 0.5 \\ 1.5 \end{bmatrix}$$

Después de llegar hasta la novena iteración (k=0) y ver que el e=0 decimos que la neurona aprendió después de haber pasado un ciclo ó sea dwi/dt = 0

2.8 Comportamiento final de la Red Neuronal

Para este trabajo debemos disponer de una red entrenada. Es posible alimentar a este sistema con una nueva entrada (nunca antes vista), una situación nueva, y nuestra Red Neuronal producirá una respuesta razonable ó inteligente en sus salidas. Puede tratarse de la predicción de un valor en la bolsa en ciertas circunstancias, el riesgo de un nuevo préstamo, una advertencia sobre el clima local ó la identificación de una persona en una nueva imagen.

Es sencillo, pero funciona. El futuro de las Redes Neuronales estará determinado en parte por el desarrollo de chips ad hoc, avances en la computación óptica/paralela y tal vez en un nuevo tipo de unidad química de procesamiento.

CAPÍTULO 3.

EL LENGUAJE DE DESCRIPCIÓN EN HARDWARE VHDL

La lógica programable es una forma más rápida y directa de integrar aplicaciones, la cual permite independizar el proceso de fabricación del proceso del diseño fuera de la fábrica de semiconductores.

Los dispositivos lógicos programables favorecen la integración de aplicaciones y desarrollos lógicos mediante el empaquetamiento de soluciones en un circuito integrado. El resultado es la reducción de espacio físico dentro de la aplicación, se trata de dispositivos fabricados y revisados que se pueden personalizar desde el exterior con diversas técnicas de programación.

Una gran ventaja al diseñar sistemas digitales mediante lógica programable radica en el bajo costo de los recursos requeridos para el desarrollo de estas aplicaciones. El soporte básico de recursos se encuentra formado por una computadora personal, un grabador de dispositivos lógicos programables y el software de aplicación que soporta las diferentes familias de circuitos integrados.

La lógica programable es una herramienta de diseño muy poderosa, que se aplica en el mundo industrial y en proyectos universitarios en todo el mundo. En la actualidad se usan desde los Dispositivos Lógicos Programables (PLD) más sencillos, hasta los potentes Dispositivos Lógicos Programables Complejos (CPLD) y Arreglos de Compuertas Programables en Campo (FPGA) que tienen aplicaciones en distintas áreas como las telecomunicaciones, computación, redes, microondas, telefonía celular, filtros digitales programables, medicina, sistemas digitales, multiprocesamiento de datos, procesamiento digital de señales, etc.

Los CPLD son recomiendan en aplicaciones donde se requieren muchos ciclos de sumas de productos, ya que pueden introducirse en el dispositivo para ejecutarse al mismo tiempo, lo que conduce a pocos retrasos. En la actualidad, los CPLD son muy utilizados a nivel industrial, ya que resulta fácil convertir diseños compuestos por múltiples PLD sencillos en un circuito CPLD. Razón por la cual en este trabajo ocuparemos para realizar el proyecto circuitos CPLD.

3.1 Lenguajes de Descripción de Hardware (HDL)

Ante la necesidad de integrar un número mayor de dispositivos en un solo circuito, se desarrollaron nuevas herramientas de diseño que auxilian a integrar sistemas de mayor complejidad. Este permitió que en la década de los cincuenta aparecieran los lenguajes de descripción de hardware (HDL).

Estos lenguajes alcanzaron su mayor desarrollo en los años setenta, tiempo en el cual se desarrollaron varios de ellos como IDL de IBM, TI-DHL de Texas Instruments, ZEUS de General Electric, etc., orientados al ámbito industrial, pero también se crearon los del ámbito universitario AHPL, DDL, CDL, ISPS, etc.

El desarrollo continuó y en la década de los ochenta surgieron lenguajes como **VHDL**, Verilog, ABEL 5.0, AHDL, etc., considerados lenguajes de descripción de Hardware por que permitieron abordar un problema lógico a nivel funcional esto es describir un problema solo conociendo las entradas y las salidas, lo cual facilita la evaluación de soluciones alternativas antes de iniciar un diseño detallado.

Una de las principales características de estos lenguajes radica en su capacidad para describir en distintos niveles de abstracción cierto diseño. Los niveles de abstracción se utilizan para clasificar modelos HDL según el grado de detalle y precisión de sus descripciones.
Los niveles de abstracción descritos desde el punto de vista de simulación y síntesis del circuito pueden definirse como sigue:

- **Algorítmico:** Se refiere a la relación funcional entre las entradas y salidas del circuito o sistema, sin hacer referencia a la realización final.

- **Transferencia de registros (RT):** Consiste en la partición del sistema en bloques funcionales sin considerar a detalle la realización final de cada bloque.

- **Lógico o de compuertas:** El circuito se expresa en términos de ecuaciones lógicas o de compuertas.

3.2 VHDL

Actualmente el lenguaje de descripción en hardware más utilizado a nivel industrial es VHDL, que aparecía en la década de los ochenta como lenguaje estándar, capaz de soportar el proceso de diseño de sistemas electrónicos complejos, con propiedades para reducir el tiempo de diseño y los recursos tecnológicos requeridos.

El departamento de defensa de los Estados Unidos creó el lenguaje VHDL como parte del programa “Very High Speed Integrated Circuits” (VHSIC) a partir del cual se detecto la necesidad de contar con un medio estándar de comunicación y la documentación para analizar la gran cantidad de datos asociados para el diseño de dispositivos de escala y complejidad deseados, por lo cual VHSIC debe de entenderse como la rapidez en el diseño de circuitos integrados.

Después de varias versiones revisadas por el gobierno de los Estados Unidos, industrias y universidades, el IEEE (Instituto de Ingenieros Eléctricos y Electrónicos) publico en 1987 el estándar de IEEEstd 1076-1987.

Tiempo después surgió la necesidad de describir en VHDL todos los ASIC (Circuitos Integrados Desarrollados para Aplicaciones Específicas) creados por el Departamento de Defensa por lo que en 1993 se adoptó el estándar adicional de VHDL IEEE1164.

En la actualidad VHDL se considera como un estándar para la descripción, modelado y síntesis de circuitos digitales y sistemas complejos. Este lenguaje presenta diversas características que lo hacen uno de los HDL mas utilizados. Finalmente la letra “V” dentro de VHDL hace referencia al proyecto VHSIC.

El proceso de estandarización de VHDL no se detuvo con la primera versión del lenguaje VHDL ’87, si no que a continuado con nuevas versiones y constantes actualizaciones, mejoras y metodologías de uso. Entre estas adiciones o actualizaciones se encuentra una muy importante: la extensión analógica 1076.1, que permite la utilización de un lenguaje único en todas las tareas de especificación, simulación y síntesis de sistemas electrónicos digitales, analógicos o mixtos.

En resumen como la indican las siglas, VHDL es un lenguaje orientado a la descripción o modelado de sistemas digitales, se trata de un lenguaje mediante el cual se puede describir, analizar y evaluar el comportamiento de un sistema electrónico digital.
Es un lenguaje poderoso que permite la integración de sistemas digitales sencillos, elaboraos o ambos en un mismo dispositivo programable.

3.3 Unidades de Diseño de VHDL

La estructura general de un programa en VHDL está formada por módulos o unidades de diseño, cada uno de ellos compuesto por un conjunto de declaraciones e instrucciones que definen, describen, estructuran, analizan y evalúan el comportamiento de un sistema digital.

Existen cinco tipos de unidades de diseño en VHDL:

- **Declaración de entidad**
- **Arquitectura**
- **Configuración**
- **Declaración del paquete**
- **Cuerpo del paquete**

El desarrollo de programas en VHDL pueden utilizarse o no tres de los cinco módulos, pero dos de ellos (entidad y arquitectura) son indispensables en la estructuración de un programa con VHDL.
Las declaraciones de entidad, paquete y configuración se consideran unidades de diseño primarias, mientras que la arquitectura y el cuerpo de paquete son unidades de diseño secundarias por que depende de una entidad primaria que se debe de analizar antes que ellas.

3.4 Entidad

Una entidad es el bloque elemental de diseño en VHDL, las entidades son todos los elementos electrónicos que forman de manera individual o en conjunto un sistema digital. La entidad puede representarse de diversas maneras como se muestran en las figuras 3.1, 3.2 y 3.3.

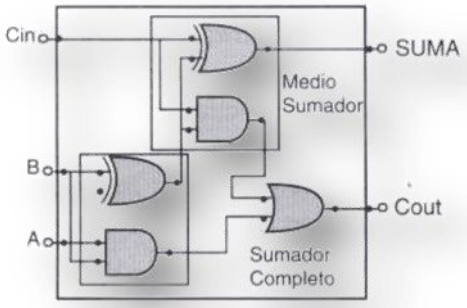


Figura 3.1. Arquitectura de un sumador completo a nivel de compuertas.

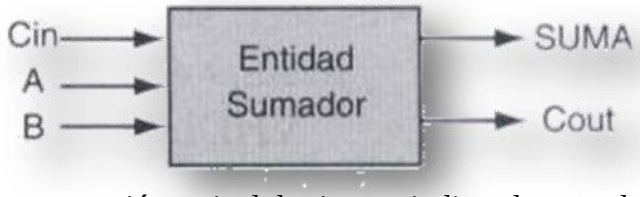


Figura 3.2. Representación a nivel de sistema indicando tan solo las entradas (Cin, A y B) y las salidas (Cout y SUMA) del circuito.

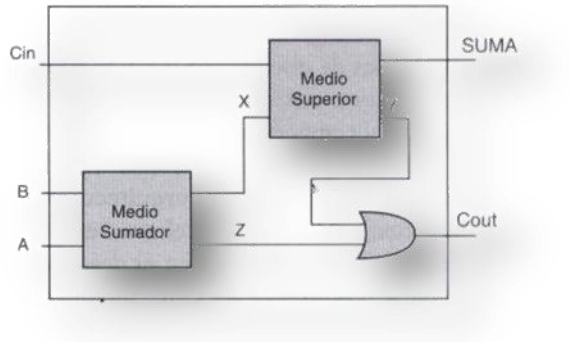


Figura 3.3. Integración de varios subsistemas (medio sumador) puede representarse mediante una entidad.

Los sistemas pueden conectarse internamente entre si; pero la entidad sigue identificando con claridad sus entradas y salidas generales.

Cada una de las señales de entrada y salida en una entidad son referidas como puerto, el cual es similar a una terminal (pin) de un símbolo esquemático. Todos los puertos que son declarados deben tener un **nombre**, un **modo** y un **tipo de dato**. El **nombre** se utiliza como una forma de llamar el puerto; el **modo** permite definir la dirección que tomara la información y el **tipo** define que clase de información se transmitirá por el puerto.

En la figura 3.4 tenemos a los puertos de la entidad que representan a un comparador de igualdad, las variables **a** y **b** denotan los puertos de entrada y la variable **c** se refiere al puerto de salida.

Figura 3.4. Comparador de igualdad

Un **modo** permite definir la dirección en la cual el dato es transferido a través de un puerto. Un **modo** puede tener uno de cuatro valores:

- Modo **in**: Se refiere a las señales de entrada a la entidad. Este sólo es unidireccional y nada más permite el flujo de datos hacia dentro de la entidad.

- Modo **out**: Indica las señales de salida de la unidad.

- Modo **inout**: Permite declarar a un puerto de forma bidireccional, además permite la retro alimentación de señales dentro o fuera de la entidad.

- Modo **buffer**: Permite hacer retroalimentaciones internas dentro de la entidad, pero a diferencia del modo **inout**, el puerto declarado se comporta como una terminal de salida.

Todos los modos descritos se muestran en la figura 3.5

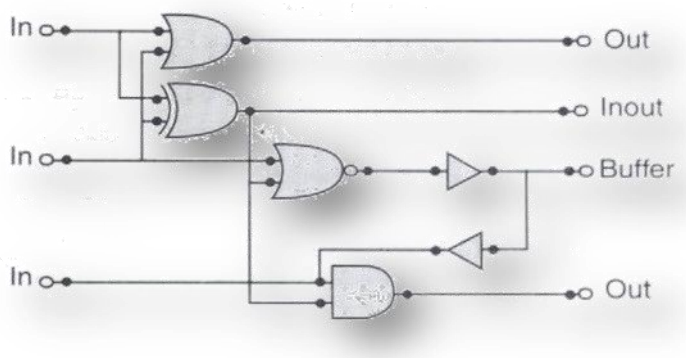


Figura 3.5. Modos y el curso de sus señales

Los **tipos de datos** son los valores que el diseñador establece para los puertos de entrada y salida dentro de una entidad; se asignan de acuerdo con las características de un diseño en particular. Algunos de los tipos más utilizados en VHDL son:

- **Bit**: Tiene valores de 0 y 1 lógico.
- **Boolean**: Esto es un valor booleano que define valores de verdadero o falso en una expresión.
- **Bit_vector**: Esto es un vector de bits que representa un conjunto de estos para cada variable de entrada o salida.
- **Integer**: Esto representa un número entero.
- **std_logic, std_ulogic, std_logic_vector, std_ulogic_vector**: Esto es que puede tener nueve valores para indicar la intensidad de la señal.
- **Character:** Esto es que puede tener el valor de cualquier carácter.
- **Time:** Utilizado par indicar tiempo

3.5 Declaración de entidades

La declaración de una entidad consiste en la descripción de las entradas y salidas de un circuito de diseño identificado como **entity**, la declaración señala los pines de entrada y salida, con que cuenta la entidad de diseño. En la figura 3.6 tenemos un ejemplo de la declaración de una entidad.

Figura 3.6. Ejemplo de entidad suma

```
1  --Declaración de la entidad de un circuito sumador
2  entity sumador is
3  port (A, B, Cin: in bit;
4           SUMA, Cout: out bit);
5  end sumador;
```

Los números del lado izquierdo del 1 al 5 no son parte del código, si no que se usan como referencia para explicar alguna sección en particular. Las palabras en negritas están reservadas para el lenguaje de programación en VHDL y el programador asigna los demás términos.

Analizando el código podemos observar que en la línea 1 inicia con un par de guiones intermedios "- -", esto indica que todo lo que esta seguido de estos es un comentario el cual no afectara a nuestro código ya que no será tomado en cuenta por el compilador.
En la línea 2 tenemos la declaración de la entidad por medio de la palabra reservada "**entity**" seguida del nombre con el cual identificaremos a la entidad que en este caso es "sumador" seguida de la palabra reservada "**is**".
En la línea 3 tenemos la palabra reservada "**port**" seguido de un paréntesis de apertura en el cual declararemos las entradas y salidas, después del paréntesis tenemos la declaración de nuestros puertos de entrada que son A, B y Cin, separados por comas, al declarar la ultima entrada siguen dos puntos, seguido del modo que tendrán los datos que en este caso son de entrada y se definen así con la palabra reservada "**in**", seguido del tipo de datos que en este caso es "bit" por que solo usaremos 0 ó 1, terminando la línea con punto y coma (;).
En la línea 4 declaramos los puertos restantes que en este caso son los de salida identificados con los nombres de SUMA y Cout, seguidos de dos puntos para poder definir el modo, que es de salida por lo cual usamos la palabra reservada "**out**" y el tipo de datos es "bit", al terminar de declarar todos nuestros puertos cerramos nuestro paréntesis y le agregamos punto y coma (;).
Por ultimo en la línea 5 terminamos la declaración de de la entidad con la palabra reservada "**end**" seguido del nombre de la entidad y finalizado con punto y coma (;).

El uso del punto y coma (;) es al finalizar una declaración y de dos puntos (:) al asignar nombres a las entradas y salidas.

3.6 Arquitectura

La Arquitectura se define como la estructura que describe el funcionamiento de una entidad, de tal forma que permita el desarrollo de los procedimientos que se llevaran a cabo con el fin de que la entidad cumpla las condiciones de funcionamiento deseadas.

La ventaja que tiene VHDL para definir la arquitectura radica en la manera en que pueden describirse los diseños, mediante el algoritmo de programación empleado se puede describir desde el nivel de compuertas hasta sistemas complejos.

De forma general, los estilos de programación utilizados en el diseño de arquitecturas se clasifican como:

- Estilo funcional
- Estilo por flujo de datos
- Estilo estructural

Descripción del Estilo Funcional

En la Figura 3.7 se trata de una descripción funcional porque expone la forma en que trabaja el sistema; es decir, las descripciones consideran la relación que hay entre las entradas y las salidas del circuito, sin importar cómo esté organizado en su interior. Para este caso:

$$\text{Si } a = b \text{ entonces } c = 1$$

$$\text{Si } a \neq b \text{ entonces } c = 0$$

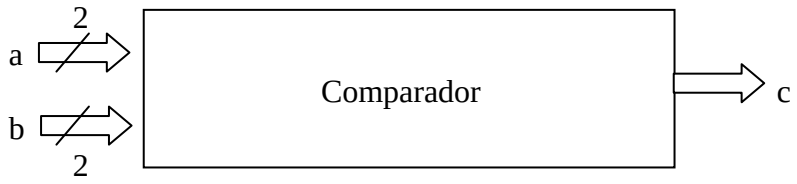


Figura 3.7. Descripción funcional de un comparador de igualdad de dos bits

El código que describe al circuito de la figura 3.7 es el siguiente:

```
1   --Ejemplo de Estilo funcional
2   library ieee;
3   use ieee.std_logic_1164.all;
4   entity comp is
5   port (a,b: in bit_vector (1 downto 0);
6           c: out bit);
7   end comp;
8   architecture funcional of comp is
9   begin
10  compara: process (a,b)
11  begin
12        if  a  =  b  then
13              c  <= '1';
14        else
15              c  <= '0';
16        end if;
17  end process compara;
18  end funcional;
```

La explicación del código es la siguiente:

En la línea 1 tenemos un mensaje que no será tomado en cuenta para el programa. En la línea 2 hacemos la declaración de la librería "ieee" y en la línea 3 tenemos la declaración del paquete que usaremos.
De la línea 4 a la 7 tenemos la declaración de la entidad "comp".
En la línea 8 empieza la declaración de la arquitectura con la palabra reservada "architecture" seguida del nombre que se le asignara en este caso "funcional" la palabra reservada "of" y el nombre de la entidad declarada anteriormente "comp" y la palabra "is".
En la línea 9 solo ponemos la palabra reservada "begin" indicando el inicio de nuestra arquitectura.
En la línea 10 declaramos un proceso con la palabra reservada "process" antecedida por una etiqueta para identificar el proceso que en este caso es compara. Después del proceso tenemos una lista de variables sensitivas dentro de unos paréntesis, estas variables determinaran el funcionamiento del proceso.
En la línea 11 solo tenemos la palabra reservada "begin" que nos indicara el inicio del proceso.
En la línea 12 a la 17 tenemos el comportamiento del proceso con la sentencia secuencial **if-then-else,** con la siguiente interpretación, en la línea 12: Si el valor de **a** es igual al de **b**

entonces, en la línea 13: A **c** asigna el valor de 1 lógico, en la línea 14: sino, en la línea 15: Asigna a **c** el valor de 0 lógico.
En la línea 16 tenemos el final de la sentencia "if" que se da por medio de la palabra reservada "end" seguida de la sentencia usada (if) y punto y coma (;).
En la línea 17 tenemos el final del proceso con las palabras reservadas "end process" seguidas de la etiqueta del proceso "compara" y punto y coma al final de la sentencia.
Por ultimo en la línea 18 tenemos el termino de la arquitectura que se da con la palabra reservada "end" seguida del nombre de la arquitectura "funcional" y se termina con punto y coma (;).

Descripción del Estilo por Flujo de Datos

La descripción por flujo de datos indica la forma en que los datos se pueden transferir de una señal a otra sin necesidad de declaraciones secuenciales. Este tipo de descripciones permite definir el flujo de datos entre módulos encargados de realizar operaciones. En este tipo de descripción se puede utilizar dos formatos: mediante instrucciones **when – else** o por medio de ecuaciones booleanas.

A continuación se muestra el programa del comparador de la figura 3.7 mediante las instrucciones **when – else**:

```
1   library ieee;
2   use ieee.std_logic_1164.all;
3   entity comp is
4   port (a,b: in bit_vector (1 downto 0);
5           c: out bit);
6   end comp;
7
8   architecture f_datos of comp is
9   begin
10     c  <= '1' when (a = b)  else '0';
11  end f_datos;
```

Descripción del Estilo por Flujo de Datos mediante Ecuaciones Booleanas

El interior del circuito comparador de la izquierda de la figura 3.8 se puede representar por medio de compuertas básicas como se muestra en el lado derecho de la figura 3.8 y de esta forma puede describirse en un programa mediante la obtención de sus ecuaciones booleanas.

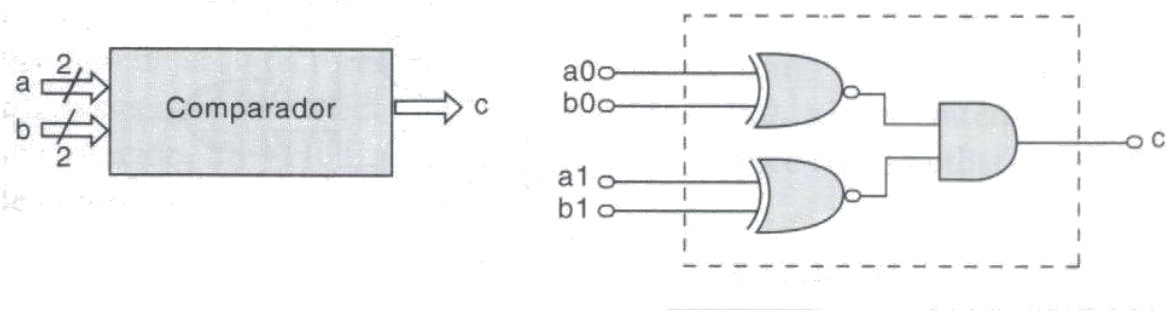


Figura 3.8 Diagrama de circuito comparador

```
1   library ieee;
2   use ieee.std_logic_1164.all;
3   entity comp is
4   port (a,b: in bit_vector (1 downto 0);
5           c: out bit);
6   end comp;
7
8   architecture booleana of comp is
9   begin
10     c  <= ((a(1) xnor b(1)) and (a(0) xnor b(0)));
11  end booleana;
```

La forma de flujo de datos en cualquiera de sus representaciones describe el camino que los datos siguen al ser transferidos de las operaciones efectuadas entre las entradas a y b a la señal de salida c.

Descripción del Estilo Estructural

Una descripción estructural basa su comportamiento en modelos lógicos establecidos (compuertas, sumadores, contadores, etc.).
Con esto el usuario puede diseñar estas estructuras y guardarlas para su uso posterior o tomarlas de los paquetes contenidos en las librerías de diseño del software que se este utilizando. En la figura 3.9 podemos observar el diagrama del comparador mas a detalle.

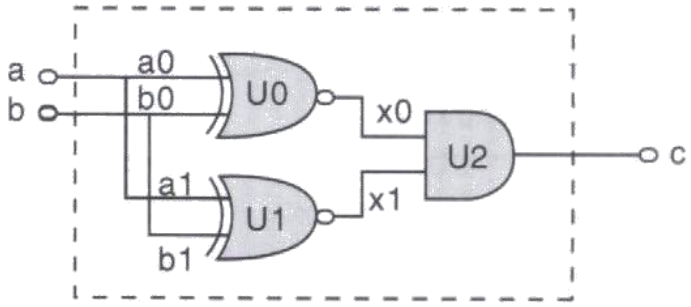


Figura 3.9. Representación esquemática de un comparador de 2 bits

En este caso, cada compuerta se encuentra del paquete **gatespkg**, del cual se toman para estructurar el diseño. Este tipo de arquitecturas estándares se conoce como **componentes**, que al interconectarse por medio de señales internas (**x0, x1**) permiten proponer una solución. En VHDL esta conectividad se conoce como listado de componentes o netlist.

Para la programación de una entidad estructural, es necesaria la descomposición lógica del diseño en pequeños submódulos (jerarquizar), los cuales permiten analizar de manera practica al circuito, ya que la función de entrada/salida es conocida.
En la figura 3.9 se conoce la función de salida de las dos compuertas **xnor**, por lo que al unirlas a la compuerta **and**, la salida **c** es resultado de la operación **and** efectuada en el interior a través de las señales **x0** y **x1.**

Una jerarquía en VHDL se refiere al procedimiento de dividir en bloques y no a que un bloque tenga mayor peso que otro. Esta forma de dividir el problema hace de la descripción estructural una forma sencilla de programar. En el contexto del diseño lógico esto se observa cuando se analiza por separado alguna sección del problema integral.

```
1   library ieee;
2   use ieee.std_logic_1164.all;
3   entity comp is port (
4         a,b: in bit_vector (0 to 1);
```

```
5           c: out bit);
6   end comp;
7   use work.compuerta.all;
8   architecture estructural of comp is
9   signal x: bit_vector (0 to 1);
10  begin
11     U0: xnor2  port map (a(0), b(0), x(0));
12     U1: xnor2  port map (a(1), b(1), x(1));
13     U2: xnor2  port map (x(0), x(1), c);
14  end estructural;
```

En el código se puede ver que la entidad nada más se describe las entradas y salidas del circuito (a, b y c), según se ha venido haciendo de las líneas 3 a las 6.

Los componentes **xnor** y **and** no se declaran debido a que se encuentran en el paquete de compuertas (**gatespkg),** el cual a su vez esta dentro de la librería de trabajo (**work**), que se define en la línea 7.

En la línea 8 se inicia la declaración de la arquitectura estructural. El algoritmo propuesto de las líneas 11 a la 13 describe la estructura de la siguiente forma:
Cada compuerta se maneja como un bloque lógico independiente del diseño original, al cual se le asigna una variable temporal (U0, U1 y U2); la salida de cada una de estos bloques se maneja como una señal línea 9, **signal** x (x0 y x1), las cuales se declaran dentro de la arquitectura y no en la entidad, debido a que no representan a una terminal (pin) y solo se utilizan para conectar bloques de manera interna a la entidad.

Por ultimo, podemos observar que la compuerta **and** recibe las dos señales provenientes de x (x0 y x1), ejecuta la operación y asigna el resultado a la salida de c del circuito.

Este estilo estructural de programación es el que usaremos mas adelante para poder programar nuestras redes neuronales artificiales.

CAPÍTULO 4.

INTERRELACIÓN ENTRE UNA RED NEURONAL Y LA PROGRAMACIÓN EN VHDL

Las redes neuronales artificiales son sistemas basados en una representación simplificada de estructura y funcionamiento del sistema nervioso, ya sea simulado en software o construido en hardware. Se deben entrenar mediante ejemplos conocidos hasta que son capaces de asociar patrones de entrada con respuestas definidas sin necesidad de una programación explícita para un patrón en particular.

Esta característica permite a las redes resolver problemas nuevos donde incluso se crean nuevas categorías de patrones para los cuales no es posible diseñar algoritmos de solución, con lo cual pueden enfrentar con éxito casos en que no es posible definir una relación entre las causas y los efectos.

Las redes neuronales son modelos de cómputo paralelo que se conocen que se conocen en ingeniería con el nombre de redes neuronales artificiales. Ofrecen una alternativa de procedimientos a problemas cuya solución por técnicas tradicionales resulta de difícil aplicación en la mayoría de los casos.

4.1 Similitudes entre una Red Neuronal Artificial y una Neurona Biológica

El funcionamiento de las redes neuronales se basa en el sistema nervioso. En general, las creaturas vivientes tienen receptores que utilizan para ver, oler, oír y sentir.

Con base en el desarrollo de las redes neuronales, se han construido circuitos electrónicos que pueden aprender de manera similar como lo hacen los seres vivos. El fundamento del aprendizaje esta en el funcionamiento de la celda nerviosa básica de la neurona biológica.
Estos pequeños bloques o estructuras de todos los sistemas vivientes, tanto del cerebro como del sistema nervioso de casi cualquier ser vivo, es posible modelarlos mediante un filtro transversal cuyos pesos varían o se adaptan cuando han alcanzado el aprendizaje deseado.
A esta estructura se le conoce también como filtro transversal adaptable.

La neurona del sistema nervioso esta compuesta por un cuerpo celular y sus prolongaciones. Una neurona tiene muchas entradas y una salida. Las dendritas en las neuronas equivalen a los receptores de información provenientes de otra neurona.
Dicha información llega al cuerpo celular y de ahí se envía por medio del axón a otras neuronas. Este proceso de comunicación entre neuronas se llama **sinapsis**.

La entrada de una red neuronal biológica puede estar compuesta por un conjunto de sensores que desempeñan la función de receptores de datos del patrón, los cuales proporcionan los estímulos provenientes del exterior al interior de la red; dichos estímulos están representados por impulsos eléctricos que se encargan de transmitir la información hacia el sistema nervioso central, provocando efectos de respuesta humana que se puede expresar en una gran variedad de acciones.

Al igual que en la neurona, en los circuitos electrónicos existe un estado bajo alto (0 ó 1), las entradas a los circuitos se acumulan y luego se procesan por medio de una función de activación que determina la respuesta. Para la función de activación suele utilizarse una función matemática que varia en forma continua con un perfil sigmoidal. La salida de la neurona o rango dinámico varían entre los limites de 0 y 1. Esto significa que la neurona transmitirá información cuando el nivel 1 es sobrepasado pero no para valores inferiores, pues es un sistema de todo o nada en su respuesta.

Tanto en la neurona biológica y en los circuitos electrónicos la comunicación se establece mediante un pulso eléctrico. En la célula nerviosa podemos medir el voltaje de la membrana, que es cercano a 0.1 volt. En los alambres electrónicos también se puede medir el voltaje cuando circula la corriente eléctrica.

La transmisión del impulso nervioso se propaga en la célula a una velocidad que oscila entre 27 y 132 metros por segundo. Así también la corriente en los alambres conductores se puede propagar a una velocidad cercana a la de la luz por efectos de la acción de los campos.

4.2 La paradoja de las RNAs

La paradoja de las RNAs es que son simulaciones matemáticas en ordenadores. Esto hace que su utilización en situaciones reales sea mucho más limitada, ya que la simulación lleva tiempo por lo que las RNAs no pueden funcionar en tiempo real.
Las RNAs pueden funcionar bien para conducir un coche o controlar un avión en vuelo ya que son extremadamente robustas frente a las contaminaciones de la señal y/o impulso y pueden seguir funcionando a pesar de que alguna de las unidades de la red deje de funcionar. Sin embargo, los sistemas expertos que se usan generalmente en los pilotos automáticos son ordenadores digitales programados con software determinístico convencional que por seguridad siempre requieren de un sistema de "backup".

Si las cosas se complican terriblemente y el sistema no puede funcionar, entonces el piloto debe hacerse cargo de la situación. Los algoritmos actuales de entrenamiento para las RNAs son muy lentos para este tipo de situaciones de emergencia. Si las neuronas de silicio pudieran aprender, lo que actualmente no pueden hacer, muchos de estos problemas se podrían evitar. A la vez que seguimos comprendiendo como funciona el cerebro, seremos capaces de construir redes neuronales más sofisticadas que nos permitirían un funcionamiento real similar al del cerebro.

4.3 Programando la Neurona en Pseudocódigo.

Normalmente para programar utilizamos determinadas técnicas para poder graficar mejor nuestro trabajo. Una de ellas es la utilización del llamado **Pseudocódigo.** Este se utiliza para determinar los pasos lógicos que se deberían utilizar para realizar un programa independientemente del lenguaje elegido.
Ahora vamos a escribir el programa maestro con la cual funciona la neurona:

PROGRAMA NEURONA

HACER MIENTRAS SE RECIBAN IMPULSOS DE LAS NEURONAS VECINAS
RECOGER IMPULSO ELÉCTRICO
DECIDIR A CUALES NEURONAS SE DEBE DESCARGAR LA ENERGÍA
HACER MIENTRAS QUEDEN NEURONAS SIN HABER RECIBIDO IMPULSO
SELECCIONAR NEURONA RECEPTORA
DESCARGAR ENERGÍA A LA NEURONA RECEPTORA
REFORZAR CONEXIÓN CON ESTA NEURONA
VOLVER A SELECCIONAR NEURONA
VOLVER A EMPEZAR

Este proceso continúa hasta que no existen más neuronas receptoras sino que las neuronas están conectadas a una terminal nerviosa, (sensores), lo cual activa la secuencia de acción correspondiente.
De esto se desprende la existencia de niveles o capas de neuronas, que tienen como función la potenciación o disminución de los impulsos iníciales.
Esto es coherente con las investigaciones de la fisiología cerebral, que coinciden en la existencia de diferentes zonas del cerebro especializadas en funciones determinadas.

Memoria

La memoria consiste desde este enfoque en la suma de los pesos de las ponderaciones o preferencias correspondientes a las conexiones en un conjunto o capa de neuronas relacionadas. La activación de este sector por un impulso nervioso obtiene como resultado que la ultima capa del sector (las neuronas relacionadas con otras no pertenecientes al mismo) ofrezca una salida hacia los centros neurales que provocarán la sensación de un "recuerdo".
Tenemos entonces que el mismo elemento (la neurona) actúa como conductor de señales, programa de acción y memoria

Axiomas para la simulación computacional del proceso biológico

Para poder simular estos procesos mediante dispositivos computacionales se deben elaborar los siguientes axiomas, sin los cuales no tiene sentido la misma.

Axioma principal

· No es importante la substancia física de la cual están hechas las neuronas, sino su función.

Las neuronas, al igual que el resto de la estructura de los seres vivos están compuestas por cadenas de átomos de carbono entrelazadas. Sin embargo, la composición química de las mismas no es un factor determinante a la hora de tratar de duplicarlas, sino su función. Esto lleva a la idea de que es posible crear una neurona artificial simulando sus funciones en un dispositivo conveniente como ser una computadora o circuitos eléctricos. De no existir este supuesto, se tendría que suspender el estudio de las redes neuronales artificiales ya que no se podría fabricarlas. Este axioma nos lleva a pensar en los tejidos cerebrales como una especie de "redes de carbono", con lo que algunas de sus funciones pueden ser simuladas por dispositivos creados en otro material. Una rueda puede estar hecha en madera, metal o caucho vulcanizado, pero siempre es una rueda.

Axioma secundario

· El comportamiento "inteligente" (limitado en este caso al reconocimiento de patrones) esta dado por la interrelación de los mensajes e interacciones de los nodos de la red.

Esto significa que de tenerse un sustituto artificial de la neurona y de aplicarse un mecanismo de interacción entre las mismas similar al utilizado por el cerebro humano, se podrían duplicar o simular algunas de las funciones abstractas del mismo. De no contarse con este axioma, tampoco sería posible proseguir con la investigación ya que no habría forma de reproducir el comportamiento aunque existieran neuronas artificiales.

Considerando entonces la aplicabilidad de los teoremas enunciados, resta entonces definir equivalentes computacionales de los procesos descriptos anteriormente.

Equivalentes artificiales de los dispositivos biológicos

El cuadro 1 muestra la equivalencia entre los elementos artificiales y los biológicos a los fines de reproducir la trama neuronal

CUADRO 1

ELEMENTO BIOLÓGICO	***ELEMENTO ARTIFICIAL***
Neurona	• NODO DE LA RED = POSICIÓN DE MEMORIA RAM + ESPACIO DISPONIBLE DE DISCO RÍGIDO + PROGRAMA DE SOFTWARE NEURONAL
Entradas Sensoriales	• DIGITALIZACION DE IMÁGENES • ARCHIVOS DE COMPUTADORA DE DIVERSOS FORMATOS
Salidas Sensoriales **Comportamiento**	• CREACIÓN DE ARCHIVO DE COMPUTADORA CON DATOS PROCESADOS • ACTIVACIÓN DE PROGRAMA DE COMPUTADORA PREDEFINIDO • ACTIVACIÓN DE INTERFACES ROBOTICAS ADECUADAS
Interconexión Sinaptica	• PREFERENCIAS Y PONDERACIONES DE LA TRANSFERENCIA DE VALORES ENTRE LAS DISTINTAS POSICIONES DE MEMORIA

4.4 Pasos para la ejecución de una red neuronal

El proceso artificial consiste en crear un elemento que sea capaz de realizar los siguientes pasos:

1. Obtener datos.
2. Generar múltiples espacios de memoria. Esto se puede hacer mediante la creación de múltiples vectores.
3. Dar a los mismos un orden de preferencia o ponderación simbolizando de este modo las distintas capas de neuronas.
4. Asignar a la primera capa el status de capa de entrada, la cual se nutrirá entonces de los datos de entrada.
5. Asignar a la última capa el status de capa de salida, permitiendo que sus valores sean transformados por la interacción de los demás elementos de la red.
6. Asignar a las capas intermedias el status de capas ocultas, permitiendo que sus valores sean modificados por la interacción de los demás elementos de la red.
7. Propagar Adelante: Realizar operaciones de transferencia entre los datos de los capas simulando de este modo las interconexiones sinápticas.
8. Recolectar el valor remanente de la capa de salidas, lo cual será tomado como el valor de la red para la entrada determinada.
9. Una vez obtenida la salida de la red, asignar una acción determinada para la misma, ya sea la creación de un archivo con la información, o la activación de un dispositivo automático.

Obtención de Datos

La red neuronal debe partir como base del equivalente artificial de un estímulo para poder empezar a operar. En el organismo viviente esté esta dado por una excitación de cualquiera de los sentidos o de una combinación de la excitación de varios de ellos.

En nuestro robot, este proceso debe dividirse en dos partes:

a) El robot con su control de redes neuronales debe estar prendido, es decir se requiere una activación externa para que funcione, cosa que es automática en los seres vivos.
b) Se requiere una entrada, es decir un patrón al cual reconocer. Cualquier entrada debe ser convertida a formato digital para de esa manera poder ser manejada de manera más eficiente y además que en VHDL las señales se utilizan en su formato digital.

Digitalizar información consiste en transformar la misma en una cadena de ceros y unos, llamada cadena de caracteres binarios.

Se denomina genéricamente a estos datos: "Trama de Entrada".

Generación de espacios de memoria

Una de las formas de simular artificialmente neuronas es la de asimilarlas a elementos de vectores o arrays.
Cualquier lenguaje de programación ofrece la posibilidad de crear vectores n-dimensionales. La cantidad de vectores depende de la arquitectura de la red a elegir, el dispositivo que vamos a utilizar y la aplicación para la que queremos la red.

Orden de preferencia o peso para las capas de neuronas

La red trata de simular el movimiento de interconexión sináptica ente distintas capas neuronales, reforzando (ponderando) las conexiones mas frecuentes de tal manera que se cree una ruta mas potente dirigida a la salida adecuada.
Esto se logra anexando a cada valor de la capa una ponderación llamada "peso" en el vocabulario específico.
Como he mencionado anteriormente los pesos se conforman entonces como valores, los cuales se multiplican por los valores de las capas para lograr las "salidas" o resultados que se traspasan de una capa a otra.

Los pesos o valores de ponderación se establecen a priori de la ejecución efectiva de la red mediante un proceso llamado "entrenamiento" el cual se comentará a posteriori y también he explicado anteriormente el proceso y mas adelante se hará este procedimiento con una red programada en VHDL .
Para ejecutar una red, entonces se parte de la base que la red ya fue entrenada, lo que implica que existen guardados datos de valores iníciales de los pesos de los valores.
Estos pesos se deben almacenar en vectores.

Asignar al primer vector el status de capa de entrada

Los valores de la trama de entrada se encuentran en este punto en un vector provenientes de la obtención de los mismos. Se procede a cargarlos en el vector de entradas de la red. Al concluir este paso tendremos a la red lista para empezar a calcular.

Asignar al último vector el status de capa de salida

El vector de "Salidas" ya ha sido creado. Si no fueron dados valores iníciales, se debe proceder a realizar esto dando valor 0 a los elementos constitutivos.

Asignar a los vectores intermedios el status de "capas ocultas"

Se repite el proceso del paso 5 para las capas intermedias.

Propagar Adelante

La transferencia entre datos de los vectores recibe el nombre de propagación. En el caso de la propagación hacia adelante consiste en obtener:

1. Valores para la capa oculta.
2. Valores para la capa de salida.

Los valores concretos a transferir a la siguiente capa se obtienen mediante la aplicación de una función de activación a los datos y luego a la multiplicación del resultante por los "pesos" correspondientes que ya se explico anteriormente.

Recolectar el valor remanente de la capa de salidas

Este valor será el resultado de la búsqueda de la red. El valor puede almacenarse en una variable.

Asignación de acciones determinadas para la salida de la red

Las acciones que puede tomar un programa en base a la salida de la red pueden ser:

Informativas

El programa se limita a informar que ha encontrado una salida para la trama presentada como entrada.

Operativas

En este caso el sistema realiza algún procedimiento además de informar el suceso. Y este es el que nos interesa a nosotros ya que la salida va ordenar a nuestro robot las acciones que debe de realizar.

Predictivas

En este caso el sistema no se limita a observar un fenómeno y describirlo sino a vaticinar la evolución probable de una variable. Ejemplo: Una red alimentada con datos climáticos podría predecir el clima para los próximos días a partir de datos climáticos actuales.

En la figura 4.1 observamos la ejecución de una red neuronal

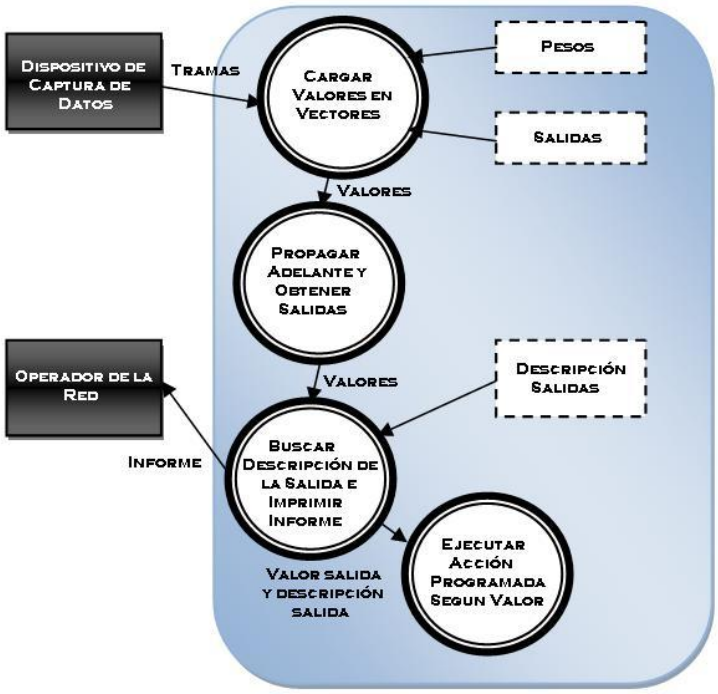


Figura 4.1. Ejecución de red neuronal

CAPÍTULO 5.

ENTRENAMIENTO DE REDES MEDIANTE VHDL

Anterior mente se dieron a conocer diversos tipos de neuronas y ahora nos adentraremos a programar el control de nuestro robot en VHDL.
Para esto vamos a trabajar 2 tipos de redes y veremos cual es la más conveniente.

5.1 Neurona Hebbiana

En 1949, Donald Hebb menciono la primera regla de aprendizaje para una RNA, la cual establece que cuando una neurona estimula a otra, la conexión entre ellas se refuerza. El modelo de la neurona Hebbiana se aprecia en la figura 5.1.

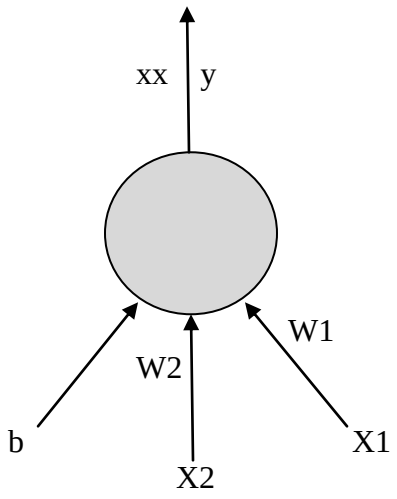


Figura 5.1. Modelo de una Neurona Hebbiana

El funcionamiento de ésta es muy sencillo. La neurona artificial calcula la entrada ponderada *I*.

$$I = b + \sum_{i=1}^{n} w_i x_i = b + \mathbf{w} \cdot \mathbf{x} \quad \ldots\ldots\ldots\ldots \quad (1)$$

Donde **w** es la matriz de pesos, **x** es el vector de entrada y **b** es una constante inicializada en 1. Si el valor de la entrada ponderada es mayor o igual a cero, la salida de la neurona es igual a +1; si es menor a cero, la salida es -1.

$$y = \begin{cases} +1; & si\ I \geq 0 \\ -1; & si\ I < 0 \end{cases}$$

El entrenamiento básicamente es el calcular el vector de ponderaciones (pesos) **w** que le permita comportarse de acuerdo con la información de entrada. Y este proceso para una neurona hebbiana es sencillo, es suficiente con cambiar el vector de ponderaciones **w**, por cada patrón de entrenamiento, de acuerdo con la siguiente regla hebbiana:

$$\mathbf{w}_{nueva} = \mathbf{w}_{anterior} + y\,\mathbf{x} \quad \ldots\ldots\ldots\ldots \quad (2)$$

Algoritmo de entrenamiento

*Inicializar el vector de ponderaciones **w** en cero.*
Para cada patrón de entrenamiento:
{
  *Igualar el vector de entrada **x** con el patrón de entrenamiento actual.*
  Igualar la salida 'y' con la salida deseada;
  *Ajustar el vector de ponderaciones **w** como*
    *Nuevo **w** = anterior **w** + **x** y;*
  Ajustar
    Nuevo b = anterior b + y;
}

En este caso vamos a entrenar una neurona hebbiana para reconocer la compuerta AND.

Entrada		**Salida**
(x1	x2)	
1	1	1
1	-1	-1
-1	1	-1
-1	-1	-1

Para el primer vector de entrenamiento (1, 1), tenemos:

$$\boldsymbol{w}_1 = \boldsymbol{w}_0 + \boldsymbol{x}\, y = (1,1)$$

$$b_1 = b_0 + y = 2$$

Para el segundo patrón de entrenamiento (1, -1):

$$\boldsymbol{x} = (1,-1), \boldsymbol{x}_1 = 1, \boldsymbol{x}_2 = -1, y = -1, \boldsymbol{w}_1 = (1,1)$$

Se calcula el siguiente vector de ponderaciones

$$\boldsymbol{w}_2 = \boldsymbol{w}_1 + \boldsymbol{x}\, y = (0,2)$$

$$b_2 = b_1 + y = 1$$

Para el tercer patrón de entrenamiento (-1, 1):

$$\boldsymbol{x} = (-1,1), \boldsymbol{x}_1 = -1, \boldsymbol{x}_2 = 1, y = -1, \boldsymbol{w}_2 = (0,2)$$

Se calcula el siguiente valor de ponderaciones

$$\boldsymbol{w}_3 = \boldsymbol{w}_2 + \boldsymbol{x}\, y = (1,1)$$

$$b_3 = b_2 + y = 0$$

Para el último patrón de entrenamiento (-1, -1):

$$\boldsymbol{x} = (-1,-1), \boldsymbol{x}_1 = -1, \boldsymbol{x}_2 = -1, y = -1, \boldsymbol{w}_3 = (1,1)$$

Se calcula el siguiente valor de ponderaciones

$$\boldsymbol{w}_4 = \boldsymbol{w}_3 + \boldsymbol{x}\, y = (2,2)$$

$$b_4 = b_3 + y = -1$$

Ahora solo queda transformarlo a un programa de VHDL:

```
use std.textio.all;

entity hebb is end;

architecture neurona of hebb is
   file arch_ent: TEXT OPEN READ_MODE IS "patron.in";
   file arch_sal: TEXT OPEN WRITE_MODE IS "pesos.in";
begin
 process
    variable buf_in, buf_out: line;
    variable x1, x2, y: integer range -1 to 1;
    variable w1, w2: integer : = 0;
    variable b:integer : = 1;
begin
  while not endfile(arch_ent) loop
      readline (arch_ent, buf_in);
      read (buf_in, x1);
      read (buf_in, x2);
      read (buf_in, y);
      w1 : = w1 + x1*y;
      w2 : = w2 + x2*y;
      b : = b + y;
  end loop;
  write(buf_out, w1);
  write(buf_out, "  ");
  write(buf_out, w2);
  write(buf_out, "  ");
  write(buf_out, b);
  writeline(arch_sal, buf_out);
  wait;
 end process;
end neurona;
```

Ahora deseamos que la neurona reconozca y realice la operación lógica AND.

El archivo de entrada *patrón.in* contiene:

```
-1   -1   -1
-1    1   -1
 1   -1   -1
 1    1    1
```

El archivo de pesos pesos.out contiene:

```
2  2  -1
```

Algoritmo

Inicializar ***w***
Inicializar b
Leer la entrada ***x***
Calcular la entrada ponderada I
Si I > = 0, entonces
    Salida es igual a +1
Si I < 0, entonces
    Salida es igual a -1;

Este programa inicializa los pesos **w** y b con los valores encontrados en el algoritmo anterior. Se calcula la entrada ponderada con el programa anterior en VHDL y se calcula el valor de la salida **y**.

Vamos a demostrarla:

Del programa anterior obtuvimos **w** = (2, 2) y b = -1.
Para el primer patrón tenemos:

$\boldsymbol{x} = (1,1), \boldsymbol{x}_1 = 1, \boldsymbol{x}_2 = 1$

Por lo tanto:

$I = x_1\boldsymbol{w}_1 + \boldsymbol{x}_2 w_2 + b = (1)(2) + (1)(2) - 1 = 3$, como I > 0 entonces y = 1.

Para el segundo patrón (1, -1) tenemos:

$\boldsymbol{x} = (1, -1), \boldsymbol{x_1} = 1, \boldsymbol{x_2} = -1$

Por lo tanto,

$I = x_1 \boldsymbol{w_1} + \boldsymbol{x_2} w_2 + b = (1)(2) + (-1)(2) - 1 = -1$, como I < 0 entonces y = -1.

Para el tercer patrón (-1, 1) tenemos:

$\boldsymbol{x} = (-1, 1), \boldsymbol{x_1} = -1, \boldsymbol{x_2} = 1$

Por lo tanto,

$I = x_1 \boldsymbol{w_1} + \boldsymbol{x_2} w_2 + b = (-1)(2) + (1)(2) - 1 = -1$, como I < 0 entonces y = -1.

Para el último patrón (-1, -1) tenemos:

$\boldsymbol{x} = (-1, -1), \boldsymbol{x_1} = -1, \boldsymbol{x_2} = -1$

Por lo tanto,
$I = x_1 \boldsymbol{w_1} + \boldsymbol{x_2} w_2 + b = (-1)(2) + (-1)(2) - 1 = -5$, como I < 0 entonces y = -1.

Con los resultados anteriores podemos apreciar que las entradas se clasifican correctamente. Ahora vamos a programarlo en VHDL para que funcione con $\boldsymbol{w_1} = \mathbf{2}$, $\boldsymbol{w_2} = 2$ y beta = -1.

```
1  entity hebb is
2   port (x1, x2: in integer range -1 to 1;
3         y : out integer range -1 to 1);
4   end;
5
6  architecture neurona of hebb is
7  begin
8    process (x1, x2)
9      constant w1, w2: integer : = 2;
10     constant b: integer : = -1;
11     variable I: integer;
12    begin
13      I : = b + x1*w1 + x2*w2;
14      if I > = 0 then
15         y < = 1;
16      else
17         y < = -1;
18      end if;
19   end process;
20 end neurona;
```

5.2 Perceptrón

Warren S. McCulloch y Walter Pitts desarrollaron la red neuronal perceptrón en 1943.
Su funcionamiento radica en que la neurona suma las señales del vector de entrada **x**, multiplicada por el vector de ponderaciones (pesos) **w**, lo que da la entrada ponderada I.

$$I = \sum_{i=1}^{n} w_i x_i = \mathbf{w} \cdot \mathbf{x} \quad \ldots\ldots\ldots \quad (3)$$

La ecuación compara la señal con un valor de umbral **θ**. Si es mayor la salida es +1. De lo contrario, es -1.

$$y = \begin{cases} +1; & si\ I \geq \boldsymbol{\theta} \\ -1; & si\ I < \boldsymbol{\theta} \end{cases}$$

Para entrenar la red se escogen valores de **x** como entrada llamados patrones de entrenamiento. Por cada patrón de entrenamiento se calcula si la salida del perceptrón 'y' es correcta o incorrecta, con lo que se obtiene β.

$$\beta = \begin{cases} +1; & si\ la\ respuesta\ es\ correcta \\ -1; & si\ la\ respuesta\ es\ incorrecta \end{cases}$$

Por cada iteración el vector de ponderaciones **w** cambiará de acuerdo con la ley de Rosenblat.

$$\mathbf{w}_{nueva} = \mathbf{w}_{anterior} + \beta\, y\, \mathbf{x} \quad \ldots\ldots\ldots\ldots \quad (4)$$

Algoritmo de entrenamiento perceptrón

Para cada patrón de entrenamiento
{ Calcular las entradas I;
Calcular la salida del perceptrón 'y';
Si es correcta, entonces
{ Si la respuesta es +1, entonces
Nuevo w = anterior w + el patrón de entrada actual;
Si la respuesta es -1, entonces
Nuevo w = anterior w – el patrón de entrada actual;
}
Si es incorrecta, entonces
{ Si la respuesta es +1, entonces
Nuevo w = anterior w - el patrón de entrada actual;
Si la respuesta es -1, entonces
Nuevo w = anterior w + el patrón de entrada actual;
}
}

Ahora vamos a entrenar el perceptrón para reconocer la compuerta AND.

Entrada		Salida
x1	x2	
1	1	1
1	-1	-1
-1	1	-1
-1	-1	-1

Para el primer valor de entrenamiento (1, 1) tenemos:

$$\boldsymbol{x} = (1,1), \boldsymbol{x}_1 = 1, \boldsymbol{x}_2 = 1, \boldsymbol{w}_0 = (0,0), \theta = 1$$

$I = \boldsymbol{w}_1 x_1 + w_2 \boldsymbol{x}_2 = (0)(1) + (0)(1) = 0$, como I < θ entonces y = -1.

La respuesta esperada es incorrecta; por lo tanto, $\beta = -1$ entonces

$\mathbf{w_1} = \mathbf{w_0} + \beta$ y $\mathbf{x} = (0, 0) + (-1)(-1)(1, 1) = (1, 1)$

Para el segundo valor de entrenamiento (1, -1) tenemos:

$\boldsymbol{x} = (1, -1), \boldsymbol{x_1} = 1, \boldsymbol{x_2} = -1, \boldsymbol{w_1} = (1,1), \theta = 1$

$I = \boldsymbol{w_1} x_1 + w_2 \boldsymbol{x_2} = (1)(1) + (1)(-1) = 0$, como I < θ entonces y = -1.

La respuesta esperada es correcta; por lo tanto, $\beta = 1$ entonces

$\mathbf{w_2} = \mathbf{w_1} + \beta$ y $\mathbf{x} = (1, 1) + (1)(-1)(1, -1) = (0, 2)$

Para el tercer valor de entrenamiento (-1, 1) tenemos:

$\boldsymbol{x} = (-1, 1), \boldsymbol{x_1} = -1, \boldsymbol{x_2} = 1, \boldsymbol{w_2} = (0,2), \theta = 1$

$I = \boldsymbol{w_1} x_1 + w_2 \boldsymbol{x_2} = (0)(-1) + (2)(1) = 2$, como I > θ entonces y = +1.

La respuesta esperada es incorrecta; por lo tanto, $\beta = -1$ entonces

$\mathbf{w_3} = \mathbf{w_2} + \beta$ y $\mathbf{x} = (0, 2) + (-1)(1)(-1, 1) = (1, 1)$

Para el último valor de entrenamiento (-1, -1) tenemos:

$\boldsymbol{x} = (-1, -1), \boldsymbol{x_1} = -1, \boldsymbol{x_2} = -1, \boldsymbol{w_3} = (1,1), \theta = 1$

$I = \boldsymbol{w_1} x_1 + w_2 \boldsymbol{x_2} = (1)(-1) + (1)(-1) = -2$, como I < θ entonces y = -1.
La respuesta esperada es correcta; por lo tanto, $\beta = 1$ entonces

$\mathbf{w_4} = \mathbf{w_3} + \beta$ y $\mathbf{x} = (1, 1) + (1)(-1)(-1, -1) = (2, 2)$

El siguiente paso es transformarlo en un programa de VHDL:

```
use std.textio.all;

entity perceptron is end;

architecture neurona of perceptron is
   file arch_ent: TEXT OPEN READ_MODE IS "patron.in";
   file arch_sal: TEXT OPEN WRITE_MODE IS "pesos.in";
begin
 process
    variable buf_in, buf_out: line;
    variable x1, x2, y: integer range -1 to 1;
    variable w1, w2: integer : = 0;
    variable I : integer ;
    variable respuesta: integer range -1 to 1;
    constant theta : integer : = 1;
begin
  while not endfile(arch_ent) loop
      readline (arch_ent, buf_in);
      read (buf_in, x1);
      read (buf_in, x2);
      read (buf_in, y);
      I: = w1*x1 + w2*x2;
      if I> = theta then
         respuesta: = 1;
      else
         respuesta: = -1;
      end if;

      if y = respuesta then
         if respuesta = 1 then
            w1: = w1 + x1;
            w2: = w2 + x2;
         else
            w1: = w1 - x1;
            w2: = w2 - x2;
         end if;
      else
         if respuesta = 1 then
            w1: = w1 - x1;
            w2: = w2 - x2;
         else
            w1: = w1 + x1;
            w2: = w2 - x2;
         end if;
       end if;
  end loop;
  write(buf_out, w1);
  write(buf_out, "  ");
  write(buf_out, w2);
  writeline(arch_sal, buf_out);
  wait;
 end process;
end neurona;
```

El anterior es un programa en VHDL para entrenar un perceptrón con dos entradas (x_1, x_2) y una salida (y). El programa lee los patrones de entrenamiento de un archivo *patron.in* y escribe las ponderaciones (pesos) finales en un archivo *pesos.out*.

Ahora al igual que en el caso de la neurona Hebbiana vamos a hacer que el perceptrón sea capaz de reconocer y realizar la operación lógica AND.

El archivo de entrada *patrón.in* contiene:

```
-1  -1  -1
-1   1  -1
 1  -1  -1
 1   1   1
```

El archivo de pesos pesos.out contiene:

```
2  2
```

Algoritmo

Inicializar w;
Inicializar theta;
Leer las entrada x;
Calcular la entrada ponderada I;
Si I> = 0, entonces
    Salida es igual a +1
Si I< 0, entonces
    Salida es igual a -1

Vamos a demostrar que esto sea cierto:
De los cálculos anteriores tenemos w = (2, 2) y θ = 1

Para el primer patrón (1, 1) tenemos:

$\boldsymbol{x} = (1, 1), \boldsymbol{x_1} = 1, \boldsymbol{x_2} = 1$

Por lo tanto:

$I = x_1 \boldsymbol{w_1} + \boldsymbol{x_2} w_2 = (1)(2) + (1)(2) = 4$, como I > θ entonces y = 1.

Para el segundo patrón (1, -1) tenemos:

$\boldsymbol{x} = (1, -1), \boldsymbol{x_1} = 1, \boldsymbol{x_2} = -1$
Por lo tanto:

$I = x_1 \boldsymbol{w_1} + \boldsymbol{x_2} w_2 = (1)(2) + (-1)(2) = 0$, como I < θ entonces y = -1.

Para el tercer patrón (-1, 1) tenemos:

$\boldsymbol{x} = (-1, 1), \boldsymbol{x_1} = -1, \boldsymbol{x_2} = 1$

Por lo tanto:

$I = x_1 \boldsymbol{w_1} + \boldsymbol{x_2} w_2 = (-1)(2) + (1)(2) = 0$, como I < θ entonces y = -1.

Para el último patrón (-1, -1) tenemos:

$\boldsymbol{x} = (-1, -1), \boldsymbol{x_1} = -1, \boldsymbol{x_2} = -1$

Por lo tanto:

$I = x_1 \boldsymbol{w_1} + \boldsymbol{x_2} w_2 = (-1)(2) + (-1)(2) = -4$, como I < θ entonces y = -1.

Con lo anterior podemos constar que con los resultados obtenidos las diferentes entradas se clasifican de manera correcta.

Ahora el programa que clasifica los patrones de una compuerta AND en VHDL mediante $w_1 = 2, w_2 = 2$ y theta = 1:

```
entity perceptron is
  port ( x1,x2: in integer range -1 to 1;
         y: out integer range -1 to 1);
end entity;

architecture neu of perceptron is
  begin
    process (x1,x2)
      constant w1: integer : = 2;
      constant w2: integer : = 2;
      constant theta: integer : = 1;
      variable I: integer;
    begin
      I: = x1*w1+x2*w2;
      if I> = theta then
        y< = 1;
      else
        y< = -1;
      end if;
    end process;
  end neu;
```

Una vez hechos los programas con ambas estructuras podemos ver que el perceptrón es más sencillo un su forma de neurona individual, pero nosotros requerimos de una red, de un conjunto de neuronas y para la utilización de esto debemos ver las redes asociativas:

5.3 Redes asociativas

En las redes asociativas los patrones se almacenan asociándolos con otros patrones. Existen varias clasificaciones para las redes asociativas como: autoasociativas o heteroasociativas. Una red autoasociativa almacena los patrones asociándolos con ellos mismos, mientras que una red heteroasociativa se utiliza para asociarlos con otros patrones.

El funcionamiento de la red asociativa es:

$$I_j = \sum_i^n x_i w_{ij}. \quad . \ . \ . \ . \ . \ . \ . \ . \ . \quad (5)$$

Se calcula las entradas ponderadas a cada una de las neuronas
La función de transferencia para este tipo de redes esta dada por la siguiente ecuación si las salidas son de tipo bipolar:

$$y_j = \begin{cases} 1; si\ I > 0 \\ 0; si\ I = 0 \\ -1; si\ I < 0 \end{cases}$$

Si las salidas son de tipo binario, se puede utilizar una función como la siguiente:

$$y = \begin{cases} 1; & si\ I > 0 \\ 0; & si\ I \leq 0 \end{cases}$$

5.4 Ley Hebb para la asociación de patrones

La ley de Hebb es la regla más simple y el método mas usado para determinar los pesos de una red asociativa. Se puede usar con patrones representados por patrones binarios (0, 1) o bipolares (-1, 1), en este caso se denotan los pares de vectores de entrenamiento como **e: s**. Donde **e** es el vector de entrada y **s** es el vector de salida esperado para el vector de entrada.

Entrenamiento de una red heteroasociativa mediante la ley de Hebb

Algoritmo

Inicializar los pesos $w_{ij} = 0$*;*
Para cada patrón de entrenamiento ***e : s***
{Hacer las entradas de activación igual al patrón de entrada de entrenamiento,
$x_i = e_i$
Hacer las salidas de activación igual al patrón de entrada de entrenamiento,
$y_j = s_j$
Ajustar los pesos:

$$w_{ij(nueva)} = w_{ij(anterior)} + x_i y_i$$

}

Vamos a entrenar la red mediante la ley de Hebb, suponiendo que tenemos un vector de entrada **e** = (e_1, e_2, e_3, e_4) y el vector de salida **s** = (s_1, s_2)

e_1	e_2	e_3	e_4
1	0	0	0
1	0	1	0
0	1	0	1
0	0	0	1

s_1	s_2
1	0
1	0
0	1
0	1

Aplicando nuestro algoritmo obtenemos:

Inicializar los pesos en cero.
Para el primer par de vectores de entrenamiento e : s, (1 0 0 0) : (1 0)

$$x_1 = 1,\ x_2 = \ x_3 = x_4 = \ 0$$

$$y_1 = 1,\ y_2 = 0$$

$$w_{11(nueva)} = \ w_{11(anterior)} + \ x_1 y_1 = 0 + (1)(1) = 1$$

Los demás pasos se mantienen en cero.

Para el segundo par de vectores de entrenamiento e : s, (1 0 1 0) : (1 0)

$$x_1 = 1,\ x_2 = 0, x_3 = 1, x_4 = \ 0$$

$$y_1 = 1,\ y_2 = 0$$

$$w_{11(nueva)} = \ w_{11(anterior)} + \ x_1 y_1 = 1 + (1)(1) = 2$$

$$w_{31(nueva)} = \ w_{31(anterior)} + \ x_3 y_1 = 0 + (1)(1) = 1$$

Los demás pasos se mantienen en cero.

Para el tercer par de vectores de entrenamiento e : s, (0 1 0 1) : (0 1)

$$x_1 = 0,\ x_2 = 1, x_3 = 0, x_4 = \ 1$$

$$y_1 = 0,\ y_2 = 1$$

$$w_{22(nueva)} = \ w_{22(anterior)} + \ x_2 y_2 = 0 + (1)(1) = 1$$

$$w_{42(nueva)} = \ w_{42(anterior)} + \ x_4 y_2 = 0 + (1)(1) = 1$$

Los demás pasos se mantienen en cero.

Para el tercer par de vectores de entrenamiento e : s, (0 0 0 1) : (0 1)

$$x_1 = 0,\ x_2 = 0, x_3 = 0, x_4 = \ 1$$

$y_1 = 0,\ y_2 = 1$

$w_{42(nueva)} = w_{42(anterior)} + x_4 y_2 = 1 + (1)(1) = 2$

Los demás pasos se mantienen en cero.

La matriz de pesos queda así:

$$\boldsymbol{W} = \begin{bmatrix} w_{11} & w_{12} \\ w_{21} & w_{22} \\ w_{31} & w_{32} \\ w_{41} & w_{42} \end{bmatrix} \qquad \boldsymbol{W} = \begin{bmatrix} 2 & 0 \\ 0 & 1 \\ 1 & 0 \\ 0 & 2 \end{bmatrix}$$

Ahora el programa en VHDL para entrenar esta red:

```
use std.textio.all;

entity hetas is end;

architecture red of hetas is
   file arch_ent: TEXT OPEN READ_MODE IS "patron.in";
   file arch_sal: TEXT OPEN WRITE_MODE IS "pesos.in";
   type ponderaciones is array (0 to 3, 0 to 1) of integer;
   type ent is array (0 to 3) of integer;
   type sal is array (0 to 3) of integer;
begin
 process
    variable buf_in, buf_out: line;
    variable w : ponderaciones : = ((0,0),(0,0),(0,0),(0,0));
    variable x : ent;
    variable y : sal;
begin
  while not endfile(arch_ent) loop
      readline (arch_ent, buf_in);
      for i in 0 to 3 loop
        read(buf_in, x(i));
      end loop;
      read (buf_in, y(0));
      read (buf_in, y(1));
      for i in 0 to 3 loop
        for j in 0 to 1 loop
          w(i,j) : = w(i,j) + x(i)*y(j);
        end loop;
      end loop;
  end loop;
  for i in 0 to 3 loop
    for j in 0 to 1 loop
      write(buf_out, w(i,j));
      write(buf_out, "  ");
    end loop;
    writeline(arch_sal, buf_out);
  end loop;
  wait;
 end process;
end red;
```

5.5 Uso de una red heteroasociativa

Algoritmo

Inicializar los pesos.
Para cada patrón de entrada hacer
{Hacer **x** igual al patrón de entrada;
Calcular la entrada ponderada a cada una de las neuronas

$$I_j = \sum_i^n x_i w_{ij}$$

Determinar la activación de las salidas
}

$$y_j = \begin{cases} 1; si\ I > 0 \\ 0; si\ I = 0 \\ -1; si\ I < 0 \end{cases}$$

Probamos que el ejemplo anterior funcione para las entradas dadas, utilizando la función de transferencia:

$$y_i = \begin{cases} 1; & si\ I > 0 \\ 0; & si\ I \leq 0 \end{cases}$$

En el ejemplo anterior obtuvimos la siguiente matriz de ponderaciones:

$$\boldsymbol{W} = \begin{bmatrix} 2 & 0 \\ 0 & 1 \\ 1 & 0 \\ 0 & 2 \end{bmatrix}$$

Para el primer patrón de entrada,

x = (1 0 0 0)

$$I_1 = x_1 w_{11} + x_2 w_{21} + x_3 w_{31} + x_4 w_{41} = (1)(2) + (0)(0) + (0)(1) + (0)(0) = 2$$

$$I_2 = x_1 w_{12} + x_2 w_{22} + x_3 w_{32} + x_4 w_{42} = (1)(0) + (0)(1) + (0)(0) + (0)(2) = 0$$

Por lo tanto,

$y_1 = 1$, dado que I > 0

$y_2 = 0$, dado que I = 0

La respuesta es correcta.

Para el segundo patrón de entrada,

x = (1 0 1 0)

$$I_1 = x_1 w_{11} + x_2 w_{21} + x_3 w_{31} + x_4 w_{41} = (1)(2) + (0)(0) + (1)(1) + (0)(0) = 3$$

$$I_2 = x_1 w_{12} + x_2 w_{22} + x_3 w_{32} + x_4 w_{42} = (1)(0) + (0)(1) + (1)(0) + (0)(2) = 0$$

Por lo tanto,

$y_1 = 1$, dado que I > 0

$y_2 = 0$, dado que I = 0

La respuesta es correcta.

Para el tercer patrón de entrada,

x = (0 1 0 1)

$$I_1 = x_1 w_{11} + x_2 w_{21} + x_3 w_{31} + x_4 w_{41} = (0)(2) + (1)(0) + (0)(1) + (1)(0) = 0$$

$$I_2 = x_1 w_{12} + x_2 w_{22} + x_3 w_{32} + x_4 w_{42} = (0)(0) + (1)(1) + (0)(0) + (1)(2) = 3$$

Por lo tanto,

$y_1 = 0$, dado que I = 0

$y_2 = 1$, dado que I > 0

La respuesta es correcta.

Para el último patrón de entrada,

x = (0 0 0 1)

$$I_1 = x_1w_{11} + x_2w_{21} + x_3w_{31} + x_4w_{41} = (0)(2) + (0)(0) + (0)(1) + (1)(0) = 0$$

$$I_2 = x_1w_{12} + x_2w_{22} + x_3w_{32} + x_4w_{42} = (0)(0) + (0)(1) + (0)(0) + (1)(2) = 2$$

Por lo tanto,

$y_1 = 0$, dado que I = 0

$y_2 = 1$, dado que I > 0

La respuesta es correcta.

Ahora vamos a ver como actúa la red para una red similar al patrón de entrenamiento, que en este caso será el vector (0 0 1 0):

x = (0 0 1 0)

$$I_1 = x_1w_{11} + x_2w_{21} + x_3w_{31} + x_4w_{41} = (0)(2) + (0)(0) + (1)(1) + (0)(0) = 1$$

$$I_2 = x_1w_{12} + x_2w_{22} + x_3w_{32} + x_4w_{42} = (0)(0) + (0)(1) + (1)(0) + (0)(2) = 0$$

Por lo tanto,

$y_1 = 1$, dado que I > 0

$y_2 = 0$, dado que I = 0

La red asocia al vector (0 0 1 0) con el vector (1 0), a pesar de que el vector de entrada difiere de los que se usaron para entrenar la red.

Ahora vamos a ver su reacción a una entada completamente diferente, como el vector (0 1 1 0).

x = (0 1 1 0)

$$I_1 = x_1w_{11} + x_2w_{21} + x_3w_{31} + x_4w_{41} = (0)(2) + (1)(0) + (1)(1) + (0)(0) = 1$$

$$I_2 = x_1w_{12} + x_2w_{22} + x_3w_{32} + x_4w_{42} = (0)(0) + (1)(1) + (1)(0) + (0)(2) = 1$$

Por lo tanto,

$y_1 = 1$, dado que I > 0

$y_2 = 1$, dado que I > 0

La red asocia al vector (0 1 1 0) con el vector (1 1), el cual no se encuentra en los patrones de entrenamiento. La red responde correctamente al no encontrar similitud con los patrones de entrenamiento, y por eso responde con un valor diferente.

El siguiente paso es programar en VHDL, para poder utilizar la red.

```
entity hetas is
 port (x1, x2, x3, x4: in integer range 0 to 1;
       y1, y2: out integer range 0 to 1);
 end;

architecture red of hetas is
  type ponde is array (0 to 3, 0 to 1) of integer;
begin
  process (x1, x2, x3, x4)
    variable w : ponde : = ((2,0),(0,1),(1,0),(0,2));
    variable I1, I2: integer;
   begin
     I1 : = x1*w(0,0) + x2*w(1,0) + x3*w(2,0) + x4*w(3,0);
     I2 : = x1*w(0,1) + x2*w(1,1) + x3*w(2,1) + x4*w(3,1);
     if I1 > = 0 then
        y1 < = 1;
     else
        y1 < = 0;
     end if;
2     if I1 > = 0 then
        y2 < = 1;
     else
        y2 < = 0;
     end if;
  end process;
end red;
```

Una vez que sabemos como es el proceso de programación de una red neuronal lo siguiente es diseñar el programa para controlar un robot.
En este caso haremos el programa para un robot seguidor de línea "Inteligente" con una red heteroasociativa.

5.6 Ejemplo de Entrenamiento de una red heteroasociativa para un Robot

Vamos a entrenar la red mediante la ley de Hebb, suponiendo que tenemos un vector de entrada **e** = (e_1, e_2, e_3, e_4) y el vector de salida **s** = (s_1, s_2)

e_1	e_2	e_3	e_4
1	0	0	0
0	1	0	0
0	0	1	0
0	0	0	1

s_1	s_2
1	0
1	0
0	1
0	1

Los valores de '**e**' van a ser determinados por 2 pares de sensores infrarrojos que son los que se encargaran de sensar la línea a seguir y '**s**' serán las salidas que indican la dirección y el movimiento de los motores de nuestro robot.

Aplicando nuestro algoritmo obtenemos:

Inicializar los pesos en cero.
Para el primer par de vectores de entrenamiento e : s, (1 0 0 0) : (1 0)

$$x_1 = 1,\ x_2 = x_3 = x_4 = 0$$

$$y_1 = 1,\ y_2 = 0$$

$$w_{11(nueva)} = w_{11(anterior)} + x_1 y_1 = 0 + (1)(1) = 1$$

Los demás pasos se mantienen en cero.

Para el segundo par de vectores de entrenamiento e : s, (0 1 0 0) : (1 0)

$x_1 = 0,\ x_2 = 1, x_3 = 0, x_4 = 0$

$y_1 = 1,\ y_2 = 0$

$w_{21(nueva)} = w_{21(anterior)} + x_2 y_1 = 0 + (1)(1) = 1$

Los demás pasos se mantienen en cero.

Para el tercer par de vectores de entrenamiento e : s, (0 0 1 0) : (0 1)

$x_1 = 0,\ x_2 = 0, x_3 = 1, x_4 = 0$

$y_1 = 0,\ y_2 = 1$

$w_{32(nueva)} = w_{32(anterior)} + x_3 y_2 = 0 + (1)(1) = 1$

Los demás pasos se mantienen en cero.

Para el último par de vectores de entrenamiento e : s, (0 0 0 1) : (0 1)

$x_1 = 0,\ x_2 = 0, x_3 = 0, x_4 = 1$

$y_1 = 0,\ y_2 = 1$

$w_{42(nueva)} = w_{42(anterior)} + x_4 y_2 = 0 + (1)(1) = 1$

Los demás pasos se mantienen en cero.

La matriz de pesos queda así:

$$\boldsymbol{W} = \begin{bmatrix} 1 & 0 \\ 1 & 0 \\ 0 & 1 \\ 0 & 1 \end{bmatrix}$$

Ahora el programa en VHDL para entrenar esta red:

```
use std.textio.all;

entity hetas is end;

architecture red of hetas is
   file arch_ent: TEXT OPEN READ_MODE IS "patron.in";
   file arch_sal: TEXT OPEN WRITE_MODE IS "pesos.in";
   type ponderaciones is array (0 to 3, 0 to 1) of integer;
   type ent is array (0 to 3) of integer;
   type sal is array (0 to 3) of integer;
begin
 process
    variable buf_in, buf_out: line;
    variable w : ponderaciones : = ((0,0),(0,0),(0,0),(0,0));
    variable x : ent;
    variable y : sal;
begin
  while not endfile(arch_ent) loop
      readline (arch_ent, buf_in);
      for i in 0 to 3 loop
        read(buf_in, x(i));
      end loop;
      read (buf_in, y(0));
      read (buf_in, y(1));
      for i in 0 to 3 loop
        for j in 0 to 1 loop
          w(i,j) : = w(i,j) + x(i)*y(j);
        end loop;
      end loop;
  end loop;
  for i in 0 to 3 loop
    for j in 0 to 1 loop
      write(buf_out, w(i,j));
      write(buf_out, "  ");
    end loop;
    writeline(arch_sal, buf_out);
  end loop;
  wait;
 end process;
end red;
```

Vamos a probar que funcionan las entradas dadas, con la siguiente función de transferencia:

$$y_i = \begin{cases} 1; & si\ I > 0 \\ 0; & si\ I \leq 0 \end{cases}$$

$$\boldsymbol{W} = \begin{bmatrix} 1 & 0 \\ 1 & 0 \\ 0 & 1 \\ 0 & 1 \end{bmatrix}$$

Para el primer patrón de entrada,

x = (1 0 0 0)

$$I_1 = x_1 w_{11} + x_2 w_{21} + x_3 w_{31} + x_4 w_{41} = (1)(1) + (0)(1) + (0)(0) + (0)(0) = 1$$

$$I_2 = x_1 w_{12} + x_2 w_{22} + x_3 w_{32} + x_4 w_{42} = (1)(0) + (0)(0) + (0)(1) + (0)(1) = 0$$

Por lo tanto,

$y_1 = 1$, dado que I > 0

$y_2 = 0$, dado que I = 0

La respuesta es correcta.

Para el segundo patrón de entrada,

x = (0 1 0 0)

$$I_1 = x_1 w_{11} + x_2 w_{21} + x_3 w_{31} + x_4 w_{41} = (0)(1) + (1)(1) + (0)(0) + (0)(0) = 1$$

$$I_2 = x_1 w_{12} + x_2 w_{22} + x_3 w_{32} + x_4 w_{42} = (0)(0) + (1)(0) + (0)(1) + (0)(1) = 0$$

Por lo tanto,

$y_1 = 1$, dado que I > 0

$y_2 = 0$, dado que I = 0

La respuesta es correcta.

Para el tercer patrón de entrada,

x = (0 0 1 0)

$$I_1 = x_1w_{11} + x_2w_{21} + x_3w_{31} + x_4w_{41} = (0)(1) + (0)(1) + (1)(0) + (0)(0) = 0$$

$$I_2 = x_1w_{12} + x_2w_{22} + x_3w_{32} + x_4w_{42} = (0)(0) + (0)(0) + (1)(1) + (0)(1) = 1$$

Por lo tanto,

$y_1 = 0$, dado que I = 0

$y_2 = 1$, dado que I > 0

La respuesta es correcta.

Para el último patrón de entrada,

x = (0 0 0 1)

$$I_1 = x_1w_{11} + x_2w_{21} + x_3w_{31} + x_4w_{41} = (0)(1) + (0)(1) + (0)(0) + (1)(0) = 0$$

$$I_2 = x_1w_{12} + x_2w_{22} + x_3w_{32} + x_4w_{42} = (0)(0) + (0)(0) + (0)(1) + (1)(1) = 1$$

Por lo tanto,

$y_1 = 0$, dado que I = 0

$y_2 = 1$, dado que I > 0

La respuesta es correcta.

Ahora vamos a ver como actúa la red para una red similar al patrón de entrenamiento, que en este caso será el vector (0 0 1 1):

x = (0 0 1 1)

$$I_1 = x_1 w_{11} + x_2 w_{21} + x_3 w_{31} + x_4 w_{41} = (0)(1) + (0)(1) + (1)(0) + (1)(0) = 0$$

$$I_2 = x_1 w_{12} + x_2 w_{22} + x_3 w_{32} + x_4 w_{42} = (0)(0) + (0)(0) + (1)(1) + (1)(1) = 2$$

Por lo tanto,

$y_1 = 0$, dado que I = 0

$y_2 = 1$, dado que I > 0

La red asocia al vector (0 0 1 1) con el vector (0 1), a pesar de que el vector de entrada difiere de los que se usaron para entrenar la red.

Ahora vamos a ver su reacción a una entada completamente diferente, como el vector (1 1 1 1).

x = (1 1 1 1)

$$I_1 = x_1 w_{11} + x_2 w_{21} + x_3 w_{31} + x_4 w_{41} = (1)(1) + (1)(1) + (1)(0) + (1)(0) = 2$$

$$I_2 = x_1 w_{12} + x_2 w_{22} + x_3 w_{32} + x_4 w_{42} = (1)(0) + (1)(0) + (1)(1) + (1)(1) = 2$$

Por lo tanto,

$y_1 = 1$, dado que I > 0

$y_2 = 1$, dado que I > 0

La red asocia al vector (1 1 1 1) con el vector (1 1), el cual no se encuentra en los patrones de entrenamiento. La red responde correctamente al no encontrar similitud con los patrones de entrenamiento, y por eso responde con un valor diferente, esto aplicado a nuestro robot quiere decir que hará una acción completamente diferente en este caso se parara ya que no reconoce los patrones de entrada pero si encuentra un patrón similar lo asociara "inteligentemente" a una respuesta de entrenamiento.

El siguiente paso es programar en VHDL, para poder utilizar la red.

```
entity hetas is
 port (x1, x2, x3, x4: in integer range 0 to 1;
       y1, y2: out integer range 0 to 1);
 end;

architecture red of hetas is
  type ponde is array (0 to 3, 0 to 1) of integer;
begin
  process (x1, x2, x3, x4)
    variable w : ponde : = ((1,0),(1,0),(0,1),(0,1));
    variable I1, I2: integer;
   begin
     I1 : = x1*w(0,0) + x2*w(1,0) + x3*w(2,0) + x4*w(3,0);
     I2 : = x1*w(0,1) + x2*w(1,1) + x3*w(2,1) + x4*w(3,1);
     if I1 > = 0 then
        y1 < = 1;
     else
        y1 < = 0;
     end if;
      if I1 > = 0 then
        y2 < = 1;
     else
        y2 < = 0;
     end if;
  end process;
end red;
```

CAPÍTULO 6.

DISEÑO Y PROGRAMACIÓN DE UN ROBOT

La robótica es la ciencia que estudia el diseño y construcción de máquinas capaces de desempeñar tareas repetitivas, peligrosas para el ser humano, en las que se necesita una alta precisión o irrealizables sin intervención de una máquina.

Un robot, es un agente artificial mecánico o virtual. Es una máquina usada para realizar un trabajo automáticamente y que es controlada por una computadora o controlador.

Si bien la palabra robot puede utilizarse para agentes físicos y agentes virtuales de software, estos últimos son llamados "bots" para diferenciarlos.

En general, un robot, para ser considerado como tal, debería presentar algunas de estas propiedades:

- No es natural, sino que ha sido creado artificialmente.
- Puede sentir su entorno.
- Puede manipular cosas de su entorno.
- Tiene cierta inteligencia o habilidad para tomar decisiones basadas en el ambiente o en una secuencia pre programada automática.
- Es programable.
- Puede moverse en uno o más ejes de rotación o traslación.
- Puede realizar movimientos coordinados.

Las acciones de este tipo de robots son generalmente llevadas a cabo por motores o actuadores que mueven extremidades o impulsan al robot. Un robot está formado por los siguientes elementos: estructura mecánica, transmisiones, actuadores, sensores, elementos terminales y controlador. Aunque los elementos empleados en los robots no son exclusivos de estos (máquinas herramientas y otras muchas máquinas emplean tecnologías semejantes), las altas prestaciones que se exigen a los robots han motivado que en ellos se empleen elementos con características específicas.

De todas maneras, no hay acuerdos ni una definición precisa de qué se considera robot. Joseph Engelberger, un pionero en la industria robótica, expresó claramente esta idea con su frase: "No puedo definir un robot, pero reconozco uno cuando lo veo".

6.1 Arquitectura de los robots

Existen diferentes tipos y clases de robots, entre ellos con forma humana, de animales, de plantas o incluso de elementos arquitectónicos pero todos se diferencian por sus capacidades y se clasifican en 5 formas:

1. Androides: robots con forma humana. Imitan el comportamiento de las personas, su utilidad en la actualidad es de solo experimentación. La principal limitante de este modelo es la implementación del equilibrio a la hora del desplazamiento, pues es bípedo.
2. Móviles: se desplazan mediante una plataforma rodante (ruedas); estos robots aseguran el transporte de piezas de un punto a otro.
3. Zoomórficos: es un sistema de locomoción imitando a los animales. La aplicación de estos robots sirve, sobre todo, para el estudio de volcanes y exploración espacial.
4. Poli articulados: mueven sus extremidades con pocos grados de libertad. Su utilidad es principalmente industrial, para desplazar elementos que requieren cuidados.
5. Híbridos: Son una combinación de dos o mas de los 4 tipos de robots.

6.2 Elección del diseño del robot

En nuestro caso nosotros tendremos el proyecto de un robot que por medio de unos sensores detectara si el producto transportado por una banda esta en buenas condiciones con lo cual este tomara decisiones de la forma de catalogarlo.

Por este motivo he decidido que el tipo de robot que diseñaremos para este proyecto será un robot poli articulado, más en particular un brazo robótico cartesiano y conocido en el ámbito como sistema robótico cartesiano (SRC).

6.3 Descripción de prototipo

El prototipo del SRC es un robot cartesiano de aplicación industrial, que está diseñado para cumplir funciones de manipulación, carga y descarga de objetos.
El prototipo está compuesto por los siguientes módulos o subsistemas: estructura mecánica, actuadores, elementos de transmisión, unidad de control y efector.

6.4 Estructura Mecánica

Mecánicamente, el robot está formado por una estructura modular tipo *gantry*, cuya cadena cinemática posee 3 grados de libertad lineales asociados a los ejes X, Y y Z respectivamente. El volumen de trabajo del SRC, definido por el rango de variación de las coordenadas articulares, es de x x [m].

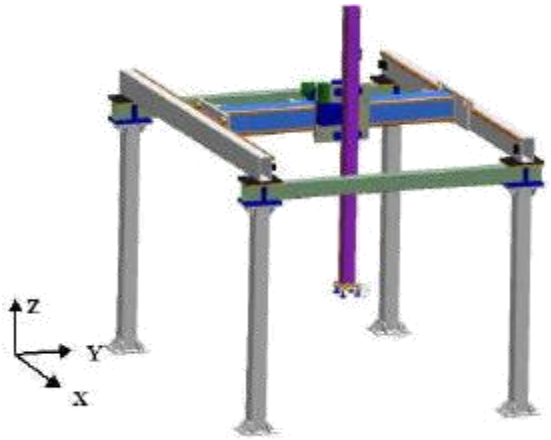


Figura 6.1 Prototipo de SRC

Actuadores y Elementos de Transmisión

Los actuadores tienen por misión generar el movimiento de los eslabones móviles del robot, según las órdenes dadas por la unidad de control. Los actuadores del SRC son servomotores, cuyas características constructivas permiten obtener una elevada respuesta dinámica, gran precisión en el posicionamiento y una mínima relación peso/potencia. Los elementos de transmisión permiten, por una parte, guiar el movimiento de los eslabones móviles, y por otra, transmitir el movimiento desde los actuadores a las articulaciones, adaptando la fuerza y la velocidad a los valores requeridos por el movimiento. Los elementos de transmisión son: reductores sinfín corona, cremalleras y piñones de dientes rectos, guías de sección prismática y elementos rodantes tipo roller cam.

Tanto los actuadores como los elementos de transmisión han sido seleccionados de modo tal que el prototipo SRC sea suficientemente robusto para desempeñarse en un ambiente industrial, con elevada confiabilidad, disponibilidad y seguridad.

6.5 Unidad de Control

El prototipo del SRC utiliza un control por medio de una red neuronal artificial del tipo Heteroasociativa, con señales de retroalimentación de velocidad y posición, para controlar el movimiento de los eslabones.

La unidad de control de movimiento del SRC está integrada por el controlador (CPLD), los accionamientos o drivers, los sensores y la interfaz hombre -máquina. El controlador realiza los cálculos y genera las órdenes de movimiento, según la aplicación. Por otro lado, los accionamientos operan como interfaz entre el controlador y los actuadores. El prototipo del SRC posee sensores de posición angular y sensores onoff, ambos de tipo interno, los que están relacionados con la fiabilidad de la operación, permitiendo a la unidad de control conocer el estado del sistema, monitorear y controlar el movimiento, corregir las desviaciones o detener el funcionamiento si detecta alguna anomalía.
Finalmente, la interfaz hombre máquina permite al usuario obtener información sobre el sistema y errores de funcionamiento.

Efector

El efector permite al robot manipular objetos, siendo el último eslabón de la cadena cinemática. El efector como tal, no pertenece al producto estándar, pues debe ser desarrollado para cada aplicación en particular. Sin embargo, en esta primera etapa del proyecto, el prototipo del SRC posee un efector del tipo griper mecánico.

6.6 Modelo Dinámico del SRC

El modelo físico de un sistema robótico debe considerar los componentes estructurales, los elementos de transmisión y los dispositivos de accionamiento y control. Sin embargo, considerando el alcance de este documento, se presenta solamente un modelo simplificado para los accionamientos y los elementos de transmisión de movimiento del prototipo del SRC.

Por otra parte, el control de movimiento de un mecanismo se puede realizar según dos modalidades: velocidad–posición, o bien, fuerza–posición. En virtud de las aplicaciones a las que está orientado el SRC, el control en modo velocidad–posición es más adecuado. Este modo de control requiere el modelo dinámico del sistema, y se utiliza cuando interesa un posicionamiento efectivo y control sobre la velocidad de movimiento del mecanismo. Respecto al problema dinámico, éste puede ser interpretado como una caja negra donde sólo intervienen las señales de entrada según sea el comportamiento de las señales de salida; indudablemente es necesario que en todo instante la potencia disponible del sistema

sea suficiente para realizar el movimiento ordenado por la unidad de control. El modelo dinámico tiene por objetivo dimensionar los equipos eléctricos y los componentes mecánicos, según los requerimientos definidos para el prototipo del SRC. La Fig. 6.2 presenta un esquema conceptual para el dimensionamiento de los equipos.

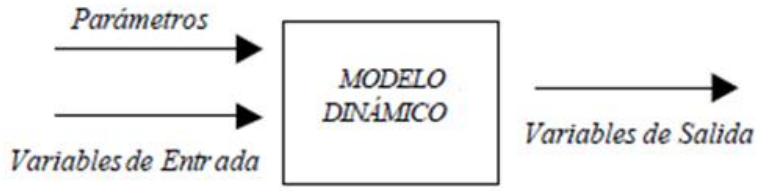


Figura 6.2 Esquema conceptual del modelo del SRC

Los requerimientos definidos para el prototipo del SRC están determinados por el ciclo de trabajo en cada eje del robot. El ciclo de trabajo (gráfica de velocidad lineal contra tiempo) para el eje X del prototipo SRC se presenta en la Fig. 6.3.

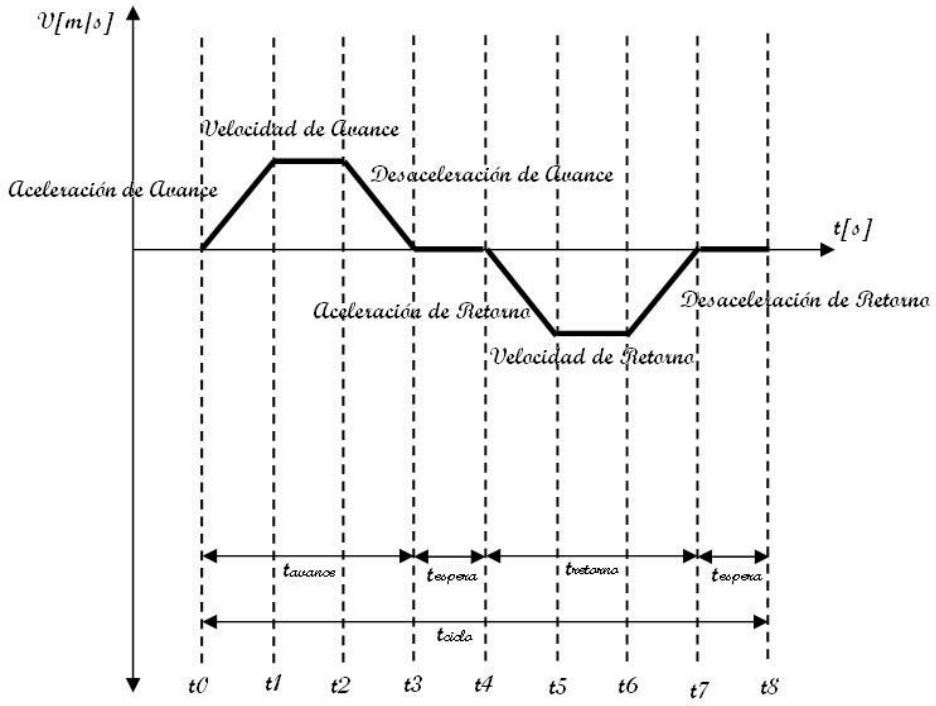


Figura 6.3 Ciclo de trabajo del eje X del SRC

Definición de Parámetros

Los parámetros son aquellas propiedades de los componentes cuyo valor permanece constante durante el análisis de una configuración determinada. Entre otros, los parámetros del modelo son:

L_K : Carrera útil, eje k [mm].
m_K: Masa total, eje k [Kg].
N_{mK} : Velocidad máxima del motor, eje k [RPM].
i_K : Factor de reducción de transmisión, eje k.
D_K : Diámetro primitivo del piñón de transmisión, eje k.
μ_K : Coeficiente de roce componentes móviles, eje k.
η_{mk} : Eficiencia mecánica del motor, eje k.
η_{rk} : Eficiencia mecánica del reductor, eje k.

Definición de Variables

Las variables de entrada son aquellas que están definidas por el ciclo de trabajo especificado. Las principales variables de entrada son:

ta_{ik}: Tiempo de aceleración, eje k [s].
td_{ik}: Tiempo de desaceleración, eje k [s].
tv_{ik}: Tiempo con velocidad máxima constante, eje k [s].
te_{ik}: Tiempo de espera, eje k [s].
tc_k: Tiempo de ciclo, eje k [s].
f_K : Ciclos por minuto, eje k [min^{-1}].

Donde:

$$tc_k = \frac{1}{f_K} \quad . \ . \ . \ . \ . \ . \ . \ . \ . \ . \ . \quad (1)$$

y los subíndices representan:

i : avance, retorno.
k : ejes x, y, z.

Por otro lado, las variables de salida son aquellas que dependen de los parámetros y de las variables de entrada del modelo. A partir de las variables de salida es posible dimensionar los equipos eléctricos y los componentes mecánicos del SRC.

Dimensionamiento de los Actuadores

Los servomotores del prototipo SRC están dimensionados a partir de las variables: torque efectivo, velocidad promedio de rotación y potencia. Se presenta el desarrollo de los cálculos para el eje X, basado en el diagrama de cuerpo libre, cuyas variables y parámetros se indican a continuación.

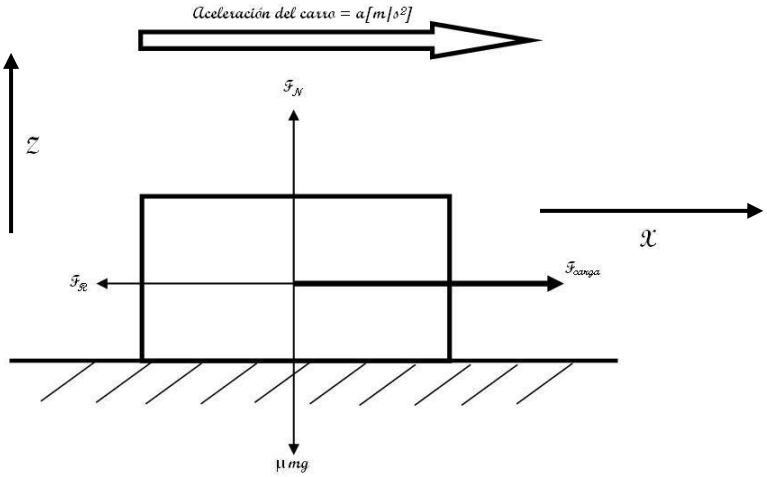


Figura 6.3 Diagrama del cuerpo libre para el eje X

m_X: Masa total del eje X [Kg].
a_X: Aceleración del eje X [m/s^2].
μ_X: Coeficiente de roce dinámico, eje X.
g: Aceleración de gravedad = 9.8 [m/s^2].
F_{NX}: Fuerza normal del eje X [N].
F_{RX}: Fuerza de roce del eje X [N].
F_{CX}: Fuerza de accionamiento del eje X.

Las ecuaciones de equilibrio de fuerzas son:

$$\sum F_X: F_{CX} - F_{RX} = m_X a_X \quad . . . . . (2)$$

$$\sum F_Z: F_{NX} - m_X g = 0 \quad . . . . . (3)$$

Por otro lado:

$$F_{RX} = \mu_X F_{NX} \quad . . . . . (4)$$

Remplazando (4) y (3) en (2), se obtiene la fuera necesaria para generar una aceleración a_X:

$$F_{CX} = m_X(a_X + \mu_x g) \quad . \ . \ . \ . \ . \ . \ . \ . \ (5)$$

La relación entre la fuerza de accionamiento lineal, F_{CX}, y el torque del piñón de transmisión es:

$$T_{PX} = \frac{1}{2} F_{CX} D_X \quad . \ . \ . \ . \ . \ . \ . \ . \ (6)$$

El torque referido al servomotor es:

$$T_{SMX} = \beta \left(\frac{T_{PX}}{i_X}\right) \quad . \ . \ . \ . \ . \ . \ . \ . \ (7)$$

Donde:

β = $\eta_{mx}\eta_{rx}$, si el servomotor está acelerando.
β =$(\eta_{mx}\eta_{rx})^{-1}$, si el servomotor está frenando

Por otra parte, el torque de inercia del sistema está dado por la relación:

$$T_{iX} = J_X \propto_X \quad . \ . \ . \ . \ . \ . \ . \ . \ (8)$$

Donde

J_X: Momento de inercia del eje X [Kg m^2].
$\propto_X$: Aceleración angular del servomotor [rad/s^2].

El torque instantáneo, en cada intervalo de tiempo Δt_j del ciclo de trabajo, está dado por:

$$T_j = T_{SMX} + T_{iX} \quad . \ . \ . \ . \ . \ . \ . \ . \ (9)$$

El torque peak en el ciclo de trabajo es:

$$T_{PK} = \max_j\{T_j\} \quad . \ . \ . \ . \ . \ . \ . \ . \ (10)$$

Entonces las variables potencia, torque efectivo y velocidad promedio de rotación son cuantificables mediante las ecuaciones (11), (12) y (13), respectivamente. La expresión para el torque efectivo del servomotor es:

$$T_{ef} = \sqrt{\frac{\sum_j T_j^2 t_j}{t_{ciclo}}} \qquad . \ . \ . \ . \ . \ . \ . \ . \ . \ (11)$$

La velocidad promedio de rotación del servomotor se calcula como:

$$N_P = \frac{\frac{1}{2}\sum_j |N_{mi} + N_{mf}| \Delta t_j}{t_{ciclo}} \qquad . \ . \ . \ . \ . \ . \ . \ . \ . \ (12)$$

Donde:

N_{mi} : Velocidad inicial de rotación en Δt_j [RPM].
N_{mf}: Velocidad final de rotación en Δt_j [RPM].

Finalmente, la potencia del servomotor se obtiene como:

$$P_{SM} = 2\pi N_P T_{PK} \qquad . \ . \ . \ . \ . \ . \ . \ . \ . \ (13)$$

Los servomotores de los ejes X e Y se dimensionan en forma análoga, es decir, planteando los ciclos de trabajo y sus diagramas de cuerpo libre.

Dimensionamiento de Elementos Estructurales

Los elementos estructurales del prototipo SRC pueden ser seleccionados utilizando la siguiente metodología:

- Identificación de las cargas puntales máximas.
- La velocidad máxima requerida para el motor no debe ser mayor que la velocidad nominal del motor seleccionado.
- El torque efectivo y la velocidad promedio de rotación del servomotor, requeridos por el ciclo de trabajo, deben estar por debajo de la curva de operación.

6.7 Sistema de Control

Como se menciono en capitulo 3, el circuito que usaremos para el desarrollo de nuestro proyecto es el CPLD y más en específico el modelo Cy7C372i que a continuación se describen sus características:

a) 64 macro celdas distribuidas en cuatro bloque lógicos
b) 32 terminales de entrada / salida
c) 6 entradas dedicadas, incluyendo 2 terminales de reloj
d) Reprogramable en sistema (ISR®)
   Tecnología flash
   Interface JTAG
e) Alta velocidad
   $F_{MAX} = 125\ MHz$
   $T_{PD} = 10\ ns$
   $T_S = 5.5\ ns$
   $T_{CO} = 6.5\ ns$
f) Compatible con PCI

6.8 Descripción del Funcionamiento del SRC

El SRC se encuentra ubicado sobre una banda trasportadora de diversos tipos de productos, y donde el producto es depositado en una base, de lo que se va encargar el SRC es de clasificar el producto de acuerdo a 3 características estarán siendo verificadas por 3 sensores que se describen a continuación:

- Sensor 1 (e1): Este sensor se encargara mediante una báscula ubicada en la base del producto que podrá ser calibrado para responder a un determinado rango de peso si esta dentro del rango este sensor nos enviara una señal de ‘1’ y si esta fuera nos mandara una señal de ‘0’.

- Sensor 2 (e2): Este sensor estará conectado a un sensor de color, de la misma forma que el anterior podrá ser calibrado para responder dentro del rango que nosotros queramos dándonos como resultado un valor de ‘1’ y en el caso contrario ‘0’.

- Sensor 3 (e3): Este sensor es el mas elaborado ya que requiere de una cámara conectada a una computadora que estará registrando la forma del objeto para detectar las variaciones que pudiera tener con respecta a la forma natural de este y de la misma forma si se encuentra dentro de los limites dados de forma, nos enviara como resultado el valor de ‘1’ y el valor de ‘0’ en el caso opuesto.

De pendiendo de estos tres sensores es del como se clasificara el producto y por ende el movimiento que tiene que hacer el SRC.

Las salidas que tendremos serán s1 y s2 que según su configuración serán los movimientos de del SRC, como se muestra a continuación:

En el caso de tener las señales:	Indica que:
s1 = 1, s2 = 1	Quiere decir que el producto es de calidad alta
s1 = 0, s2 = 1	Quiere decir que el producto es de calidad media
s1 = 1, s2 = 0	Quiere decir que el producto es de calidad baja
s1 = 0, s2 = 0	Quiere decir que el producto es defectuoso

Esto es que por cada salida el SRC tendrá un movimiento diferente dentro de los ejes X y Y, esto para la clasificación de cada producto según sus características, el movimiento del eje Z debido a su naturaleza no se vera afectado en lo mas mínimo por cualquiera de estas salidas.

6.9 Entrenamiento de una red heteroasociativa para el SRC

El sistema se entrenara para la siguiente tarea:

En el caso de recibir:			Mandara la salida:	
e1	e2	e3	s1	s2
0	0	0	0	0
1	0	0	1	0
1	1	0	0	1
1	1	1	1	1

Vamos a entrenar la red mediante la ley de Hebb, para lo que tenemos un vector de entrada **e** = (e_1, e_2, e_3) y el vector de salida **s** = (s_1, s_2)

Aplicando nuestro algoritmo obtenemos:

Inicializar los pesos en cero.
Para el primer par de vectores de entrenamiento e : s, (0 0 0) : (0 0)

$x_i = e_i,\ y_j = s_j$

$x_1 = x_2 = x_3 = 0$

$y_1 = 0,\ y_2 = 0$

Todos los pasos se mantienen en cero.

Para el segundo par de vectores de entrenamiento e : s, (1 0 0) : (1 0)

$x_1 = 1,\ x_2 = 0, x_3 = 0$

$y_1 = 1,\ y_2 = 0$

$w_{11(nueva)} = w_{11(anterior)} + x_1 y_2 = 0 + (1)(1) = 1$

Los demás pasos se mantienen en cero.

Para el tercer par de vectores de entrenamiento e : s, (1 1 0) : (0 1)

$x_1 = 1,\ x_2 = 1, x_3 = 0$

$y_1 = 0,\ y_2 = 1$

$w_{12(nueva)} = w_{12(anterior)} + x_1 y_2 = 0 + (1)(1) = 1$
$w_{22(nueva)} = w_{22(anterior)} + x_2 y_2 = 0 + (1)(1) = 1$

Los demás pasos se mantienen en cero.

Para el último par de vectores de entrenamiento e : s, (1 1 1) : (1 1)

$x_1 = x_2 = x_3 = 1,$

$y_1 = 1,\ y_2 = 1$

$w_{11(nueva)} = w_{11(anterior)} + x_1 y_1 = 1 + (1)(1) = 2$

$$w_{21(nueva)} = w_{21(anterior)} + x_2 y_1 = 0 + (1)(1) = 1$$
$$w_{31(nueva)} = w_{31(anterior)} + x_3 y_1 = 0 + (1)(1) = 1$$

$$w_{12(nueva)} = w_{12(anterior)} + x_1 y_2 = 1 + (1)(1) = 2$$
$$w_{22(nueva)} = w_{22(anterior)} + x_2 y_2 = 1 + (1)(1) = 2$$
$$w_{32(nueva)} = w_{32(anterior)} + x_3 y_2 = 0 + (1)(1) = 1$$

La matriz de pesos queda así:

$$\boldsymbol{W} = \begin{bmatrix} 2 & 2 \\ 1 & 2 \\ 1 & 1 \end{bmatrix}$$

Vamos a probar que funcionan las entradas dadas, con la siguiente función de transferencia:

$$y_i = \begin{cases} 1; & si\ I > 0 \\ 0; & si\ I \leq 0 \end{cases}$$

Para el primer patrón de entrada,

x = (1 1 1)

$$I_1 = x_1 w_{11} + x_2 w_{21} + x_3 w_{31} = (1)(2) + (1)(1) + (1)(1) = 4$$

$$I_2 = x_1 w_{12} + x_2 w_{22} + x_3 w_{32} = (1)(2) + (1)(2) + (1)(1) = 5$$

Por lo tanto,

$y_1 = 1$, dado que I > 0

$y_2 = 1$, dado que I > 0

La respuesta es correcta.

Para el segundo patrón de entrada,

x = (1 1 0)

$$I_1 = x_1 w_{11} + x_2 w_{21} + x_3 w_{31} = (1)(2) + (1)(1) + (0)(1) = 3$$

$$I_2 = x_1 w_{12} + x_2 w_{22} + x_3 w_{32} = (1)(2) + (1)(2) + (0)(1) = 4$$

Por lo tanto,

$y_1 = 1$, dado que I > 0

$y_2 = 1$, dado que I > 0

La respuesta no es correcta.

Para el tercer patrón de entrada,

x = (1 0 0)

$$I_1 = x_1 w_{11} + x_2 w_{21} + x_3 w_{31} = (1)(2) + (0)(1) + (0)(1) = 2$$

$$I_2 = x_1 w_{12} + x_2 w_{22} + x_3 w_{32} = (1)(2) + (0)(2) + (0)(1) = 2$$

Por lo tanto,

$y_1 = 1$, dado que I > 0

$y_2 = 1$, dado que I > 0

La respuesta no es correcta.

Para el último patrón de entrada,

x = (0 0 0)

$$I_1 = x_1 w_{11} + x_2 w_{21} + x_3 w_{31} = (0)(2) + (0)(1) + (0)(1) = 0$$

$$I_2 = x_1 w_{12} + x_2 w_{22} + x_3 w_{32} = (0)(2) + (0)(2) + (0)(1) = 0$$

Por lo tanto,

$y_1 = 0$, dado que I = 0

$y_2 = 0$, dado que I = 0

La respuesta es correcta.

Como se puede apreciar el funcionamiento de las entradas no es el correcto, por lo cual nuestra red no cumple nuestro propósito, por lo que para solucionar este problema se procederá a colocar un decodificador con ciertas características especiales, el cual será programado por nosotros en VHDL y a continuación se explica su funcionamiento.

En el siguiente cuadro se muestra el funcionamiento completo de los sensores así como las salidas que nos interesan, además de los nuevos valores (NE) que se le asignaran por medio del decodificador:

E1	E2	E3	S1	S2	Salidas de Interés	NE1	NE2	NE3
0	0	0	0	0	◄	0	0	0
1	0	0	1	0	◄	1	0	0
0	1	0						
1	1	0	0	1	◄	0	1	0
0	0	1						
1	0	1						
0	1	1						
1	1	1	1	1	◄	0	0	1

Estos nuevos valores se los asigne para así tener menos valores de “1” y hacer que la red se pueda entrenar de manera correcta, de este modo nosotros solo ocuparemos los nuevos valores para entrenar la red.
En el caso del decodificador funcionara de la siguiente manera:

No.	Entrada			►	Salida		
Dato	E1	E2	E3	►	NE1	NE2	NE3
1	0	0	0	►	0	0	0
2	1	0	0	►	1	0	0
3	0	1	0	►	1	0	0
4	1	1	0	►	0	1	0
5	0	0	1	►	1	0	0
6	1	0	1	►	1	0	1
7	0	1	1	►	0	1	1
8	1	1	1	►	0	0	1

Como se puede apreciar en las líneas 1, 2, 6 y 7 los datos son iguales en estos caso nuestro dispositivo actuara como buffer, no así para los datos de las líneas 3, 4, 5 y 8, que en estas entradas donde actuara el decodificador con el siguiente algoritmo:

Si x = “010” entonces x <= “100”;
Si no, x = “110” entonces x <= “010”;
Si no, x = “001” entonces x <= “100”;
Si no, x = “111” entonces x <= “001”;
Si no, x = x;

A continuación se pone el programa del decodificador en VHDL programado en un gal 22v10:

```
1  library ieee;
2  use ieee.std_logic_1164.all;
3  entity deco is
4  port (a: in std_logic_vector (2 downto 0);
5        b: out std_logic_vector (2 downto 0));
6  end deco;
7  architecture decodificador of deco is
8    begin
9    process (a)
10   begin
11   b<="000";
12         if  a  =  "010"  then
13             b  <= "001";
14         elsif  a = "011" then
15                b <= "010";
16         elsif  a = "100" then
17                b <= "001";
18         elsif  a = "111" then
19                b <= "100";
20         else
21                b <= a;
22         end if;
23   end process;
24   end decodificador;
```

En el programa anterior se puede ver en la parte superior la declaración de la librería y paquete líneas 1 y 2.
De la línea 3 a la 6 se define la entidad, donde encontramos dos vectores de 3 bits uno d entrada y uno de salida.
De la línea 7 a la 24 esta la arquitectura que indica el funcionamiento del decodificador, como se puede ver solo en las entradas seleccionadas actúa, para las demás su función es la de buffer.

Ahora el sistema se entrenara para los nuevos valores:

En el caso de recibir:			Mandara la salida:	
e1	e2	e3	s1	s2
0	0	0	0	0
1	0	0	1	0
0	1	0	0	1
0	0	1	1	1

Vamos a entrenar la red mediante la ley de Hebb, para lo que tenemos un vector de entrada **e** = (e_1, e_2, e_3) y el vector de salida **s** = (s_1, s_2)

Aplicando nuestro algoritmo obtenemos:

Inicializar los pesos en cero.
Para el primer par de vectores de entrenamiento e : s, (0 0 0) : (0 0)

$x_i = e_i,\ y_j = s_j$

$x_1 = x_2 = x_3 = 0$

$y_1 = 0,\ y_2 = 0$

Todos los pasos se mantienen en cero.

Para el segundo par de vectores de entrenamiento e : s, (1 0 0) : (1 0)

$x_1 = 1,\ x_2 = 0, x_3 = 0$

$y_1 = 1,\ y_2 = 0$

$w_{11(nueva)} = w_{11(anterior)} + x_1 y_1 = 0 + (1)(1) = 1$

Los demás pasos se mantienen en cero.

Para el tercer par de vectores de entrenamiento e : s, (0 1 0) : (0 1)

$x_1 = 0,\ x_2 = 1, x_3 = 0$

$y_1 = 0,\ y_2 = 1$

$$w_{22(nueva)} = w_{22(anterior)} + x_2 y_2 = 0 + (1)(1) = 1$$

Los demás pasos se mantienen en cero.

Para el último par de vectores de entrenamiento e : s, (0 0 1) : (1 1)

$x_1 = x_2 = x_3 = 1,$
$y_1 = 1,\ y_2 = 1$

$$w_{31(nueva)} = w_{31(anterior)} + x_3 y_1 = 0 + (1)(1) = 1$$
$$w_{32(nueva)} = w_{32(anterior)} + x_3 y_2 = 0 + (1)(1) = 1$$

La matriz de pesos queda así:

$$\boldsymbol{W} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 1 & 1 \end{bmatrix}$$

Vamos a probar que funcionan las entradas dadas, con la siguiente función de transferencia:

$$y_i = \begin{cases} 1; & si\ I > 0 \\ 0; & si\ I \leq 0 \end{cases}$$

Para el primer patrón de entrada,

x = (0 0 0)

$$I_1 = x_1 w_{11} + x_2 w_{21} + x_3 w_{31} = (0)(1) + (0)(0) + (0)(1) = 0$$

$$I_2 = x_1 w_{12} + x_2 w_{22} + x_3 w_{32} = (0)(0) + (0)(1) + (0)(1) = 0$$

Por lo tanto,

$y_1 = 0$, dado que I = 0

$y_2 = 0$, dado que I = 0

La respuesta es correcta.

Para el segundo patrón de entrada,
x = (1 0 0)

$$I_1 = x_1 w_{11} + x_2 w_{21} + x_3 w_{31} = (1)(1) + (0)(0) + (0)(1) = 1$$

$$I_2 = x_1 w_{12} + x_2 w_{22} + x_3 w_{32} = (1)(0) + (0)(1) + (0)(1) = 0$$

Por lo tanto,

$y_1 = 1$, dado que I > 0

$y_2 = 0$, dado que I = 0

La respuesta es correcta.

Para el tercer patrón de entrada,

x = (0 1 0)

$$I_1 = x_1 w_{11} + x_2 w_{21} + x_3 w_{31} = (0)(1) + (1)(0) + (0)(1) = 0$$

$$I_2 = x_1 w_{12} + x_2 w_{22} + x_3 w_{32} = (0)(0) + (1)(1) + (0)(1) = 1$$

Por lo tanto,

$y_1 = 0$, dado que I = 0

$y_2 = 1$, dado que I > 0

La respuesta es correcta.

Para el último patrón de entrada,

x = (0 0 1)

$$I_1 = x_1 w_{11} + x_2 w_{21} + x_3 w_{31} = (0)(1) + (0)(0) + (1)(1) = 1$$

$$I_2 = x_1 w_{12} + x_2 w_{22} + x_3 w_{32} = (0)(0) + (0)(1) + (1)(1) = 1$$

Por lo tanto,

$y_1 = 1$, dado que I > 0

$y_2 = 1$, dado que I > 0

La respuesta es correcta.

Vamos a checar para un valor diferente a los que se tomaron para entrenar la red:

x = (1 0 1)

$$I_1 = x_1 w_{11} + x_2 w_{21} + x_3 w_{31} = (1)(1) + (0)(0) + (1)(1) = 2$$

$$I_2 = x_1 w_{12} + x_2 w_{22} + x_3 w_{32} = (1)(0) + (0)(1) + (1)(1) = 1$$

Por lo tanto,

$y_1 = 1$, dado que I > 0

$y_2 = 1$, dado que I > 0

La respuesta es la mas parecida, tomando en cuenta el tipo de entradas. En este caso aunque el color estaba fuera de rango el sistema opto por mandar el producto a la categoría de alta calidad.

A continuación se muestran los programas para entrenar la red en VHDL:

```
 1 use std.textio.all;
 2
 3 entity het is end;
 4
 5 architecture red of het is
 6  file archivo_entrada: text open read_mode is "patron.in";
 7  file archivo_salida: text open write_mode is "peso.out";
 8  type ponderaciones is array (0 to 2, 0 to 1) of integer;
 9  type entrada is array (0 to 2) of integer;
10  type salida is array (0 to 1) of integer;
11 begin
12  process
13   variable buf_in, buf_out: line;
14   variable w: ponderaciones : = ((0,0),(0,0),(0,0));
15   variable x: entrada;
16   variable y: salida;
17  begin
18   while not endfile (archivo_entrada) loop
19    readline (archivo_entrada, buf_in);
20    for i in 0 to 2 loop
21     read(buf_in, x(i));
22    end loop;
```

```
23    read(buf_in,y(0));
24    read(buf_in,y(1));
25    for i in 0 to 2 loop
26     for j in 0 to 1 loop
27      w(i,j): = w(i,j) + x(i)*y(j);
28     end loop;
29    end loop;
30    for i in 0 to 2 loop
31     for j in 0 to 1 loop
32      write(buf_out, w(i,j));
33      write(buf_out, " ");
34     end loop;
35     wait;
36  end process;
37 end red;
```

A continuación se muestra el programa en VHDL para utilizar la red:

```
 1 entity redrobot is
 2 port (x1, x2, x3 : in integer range 0 to 1;
 3       y1, y2 : out integer range 0 to 1);
 4 end;
 5
 6 architecture controb of redrobot is
 7  type ponderaciones is array (0 to 2, 0 to 1) of integer;
 8  begin
 9  process (x1, x2, x3)
10   variable w: ponderaciones := ((1,0),(0,1),(1,1));
11   variable I1, I2: integer;
12  begin
13   I1:= x1*w(0,0) + x2*w(1,0) + x3*w(2,0);
14   I2:= x1*w(0,1) + x2*w(1,1) + x3*w(2,1);
15   if I1 > 0 then
16      y1 <= 1;
17   else
18      y1 <= 0;
19   end if;
20   if I2 > 0 then
21      y2 <= 1;
22   else
23      y2 <= 0;
24   end if;
25  end process;
26 end controb;
```

Como se puede ver una ves teniendo los valores de la matriz de ponderaciones el programa que utilizaremos para el control de nuestro robot es muy sencillo, solo queda el conectar las salidas del decodificador a la entrada de la red y obtendremos nuestros resultados. Como se muestra en la figura 6.4.

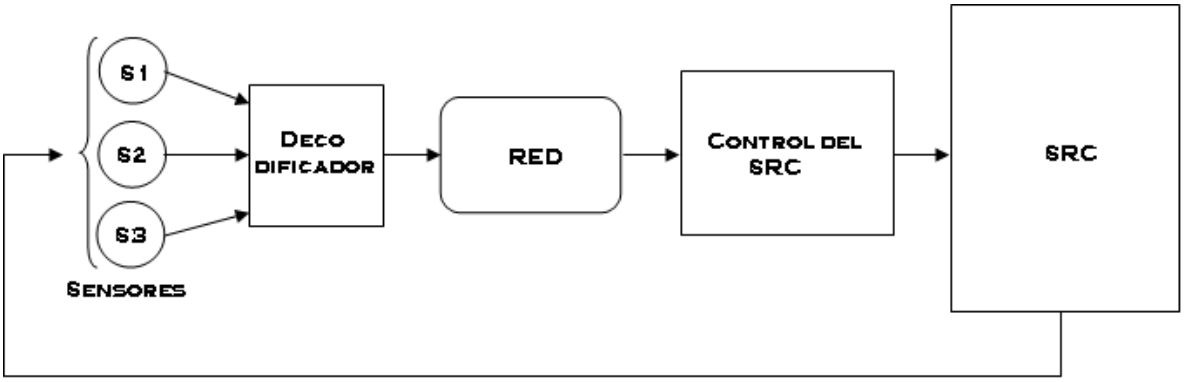


Figura 6.4 Diagrama de proceso de control del SRC

CONCLUSIONES

A lo largo de este trabajo me encontré con gran cantidad de problemas debido a la escasa información en español del tema, teniendo que recurrir ha cierta literatura anglosajona para poder seguir con el trabajo.

Este trabajo estaba orientado al desarrollo de un control “inteligente” para un robot con el mínimo de componentes y costo, además de hacerlo por medio de un lenguaje fácil de aprender y hasta cierto punto flexible, analizando esta situación puedo decir que la plataforma utilizada es la ideal para nuestros fines ya que los micro controladores que nos pudieran servir para resultados iguales o similares son más complicados de utilizar y de la misma forma a un costo mayor, esto resulta muy importante debido a que la mayoría de los que se llegan a interesar por el tema al encontrarse con elementos complejos y a alto costo merman sus aspiraciones a continuar desarrollando en el área, más sin embargo si se encuentran con un sistema fácil de implementar y con un costo accesible, sirve de escalón para posteriormente pasar a un nivel más complejo.

Con esto también debo mencionar que debido a la manera que se avanza en el área de la tecnología en estos tiempos cada vez tenemos a nuestra disposición una mayor variedad de opciones con las cuales podemos desarrollar proyectos cada vez de mejor calidad y complejidad a un costo razonable, además que cuentan con gran variedad de lenguajes en los que se pueden programar cada vez más amigables.

Una segunda parte en la que esta orientada el trabajo es la de dar a conocer de una forma clara el comportamiento de una red neuronal y sus unidades así como sus características, objetivo que tomo por cumplido debido a la conjunción de información de diversas fuentes y su explicación sencilla, con lo que resulta sencillo entender lo referente a las neuronas y sus redes, desde sus características físicas hasta las características en su comportamiento.

Este objetivo me resulta aun más importante que el anterior debido a que es mas fácil que una persona interesada en abundar en el tema de las redes neuronales artificiales, lo haga teniendo información clara, ya que la cantidad de tiempo que se lleva una persona en recabar información para su trabajo es considerable, esto debido en gran parte a lo complejo de la forma en la que se estructura la información.

Tomando en consideración lo anterior me queda claro que los objetivos se cumplieron.

BIBLIOGRAFÍA

1 [1] "REDES NEURONALES ARTIFICIALES", José R. Hilera y Victor J Martinez. 2000. Alfaomega. Madrid. España

2 VHDL El arte de programar sistemas digitales. David G. Maxinez, Jessica Alcalá, CECSA

3 Sistemas Inteligentes Artificiales. Fritz, Walter; García Martínez, Ramón y Marsiglio, Antonio (1990). Buenos Aires.

4 Máquinas que piensan: *Robots y circuitos con microcontroladores*, por Francisco Carabaza Piñeiro

5 Redes Neuronales Artificiales. José R. Hilera, Víctor J.Martínez (2000)

6 Russell, Stuart y Norvig, Peter (1996): Inteligencia Artificial: un enfoque moderno. México, Editorial Prentice Hall Hispanoamericana.

7 Manual DEGEM® SYSTEMS. Entrenamiento del CIM-2000 Mechatronics; Manual de aprendizaje de la estación de control central.

8 Barrientos A., Peñin L.P., Balaguer C.y Aracil R., (1997), *Fundamentos de Robótica*, McGraw Hill.

9 Manual CPLD Cy7C372i

10 Descripción de circuitos digitales mediante VHDL, Francisco Javier Ferrero Martín, Universidad de Oviedo

11 VHDL: Lenguaje Estándar de Diseño Electrónico. Lluís Terés, Yago Torroja, Serafín Olcoz, Eugenio Villar. McGraww-Hill. 1998

12 José R. Hilera, Víctor J.Martínez (2000) - Redes Neuronales Artificiales

13 Fiszelew, A. y García Martínez, R. 2002. Generación Automática de Redes Neuronales con Ajuste de Parámetros Basado en Algoritmos Genéticos.