3. APLICACIONES DE LAS REDES NEURONALES A INGENIERÍA ELÉCTRICA

Las aplicaciones que se estudian en el capítulo 3, representan una serie de problemas típicos en Ingeniería Eléctrica, los cuales ya han sido resueltos por métodos tradicionales. El objetivo de emplear Redes Neuronales en su solución es ofrecer una alternativa novedosa que simplifica y permite comparar la eficiencia de este nuevo enfoque respecto a las metodológicas tradicionales, a través de conocimientos matemáticos de un nivel aceptable para comprender el fundamento y desarrollo del problema a resolver.

3.1 DETECCIÓN DE OBSTÁCULOS POR MEDIO DE UN ROBOT

3.1.1 Descripción del problema.

Un robot es un dispositivo automático que realiza acciones especificas, que dependen de las necesidades del proceso en que se encuentre involucrado, en este caso se tiene un robot que cuenta con cuatro sensores de proximidad en distintas ubicaciones que permanentemente detectan si hay objetos que se

encuentren a una distancia superior o inferior a la preestablecida, con base en esto se decide si dar marcha adelante o atrás a cada uno de los dos motores que posee; en las lecturas de los sensores podrían darse 16 posibles combinaciones ($16=2^4$) y para cada combinación cada uno de los dos motores podría dar marcha adelante o marcha atrás.

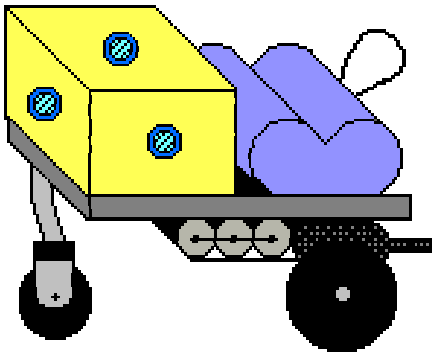

Figura 3.1.1 Robot

El comportamiento del robot lo describe la tabla 3.1.2, cuando los sensores detecten un objeto que se encuentra a una distancia inferior a la predeterminada se dirá que el objeto se encuentra cerca y esto se representa por medio de un 1 y cuando se detecte un objeto que se encuentra a una distancia mayor que la predeterminada se dirá que el objeto esta lejos lo cual se indica con un –1; dependiendo de estas lecturas los motores podrán dar marcha adelante, lo que se representara por un 1 o dar marcha atrás con un –1.

S1	S2	S3	S4	M1	M2
1	1	1	1	-1	-1
-1	1	1	1	-1	1
1	1	-1	-1	1	-1
-1	-1	-1	-1	1	1
1	-1	1	1	1	-1
1	1	-1	1	-1	1
1	1	1	-1	1	-1

Tabla 3.1.1 Comportamiento del robot

3.1.2 Justificación del tipo de red.

Este tipo de problema generalmente es resuelto suministrándole al robot una base de datos que contiene todas las posibles situaciones que se podrían presentar y sus respectivas soluciones, en este caso se necesitaría almacenar las respuestas para ambos motores ante las 16 posibles combinaciones en las lecturas de los sensores, cuando el número de variables de entrada y el número de salidas es mucho mayor, la cantidad de datos necesarios para especificar cada posible situación crece indefinidamente, debido a esto se requerirían dispositivos con gran capacidad de almacenamiento; *en contraste, una red neuronal puede entrenarse con un número representativo de patrones y aprender el comportamiento del sistema utilizando dispositivos de menos capacidad de almacenamiento y costo.*

Una red tipo Perceptrón puede ser entrenada con patrones de cualquier dimensión en la entrada y en la salida con datos binarios, por la simplicidad del problema este tipo de red es la mas adecuada.

Para garantizar que el problema puede ser resuelto por una red neuronal tipo Perceptrón se debe comprobar que los patrones de entrenamiento son linealmente separables, para esto se deben plantear las desigualdades generadas por cada patrón de entrenamiento, en este caso cada patrón de cuatro dimensiones generara dos desigualdades (una por cada salida), estas desigualdades no deben contradecirse, si esto ocurriera el problema no podría ser resuelto por una red tipo Perceptrón de una sola capa y deberá buscarse otro tipo de solución.

Debido a la naturaleza bipolar de la salida, la función de transferencia a utilizar es *hardlims* (ver sección 1.3.2), la cual se rige por las siguientes condiciones.

$$\begin{Bmatrix} n \geq 0 & a = 1 \\ n < 0 & a = -1 \end{Bmatrix} \qquad (3.1.1)$$

La salida de la red está dada por la ecuación 3.1.2, (ver sección 2.1)

$$n = (\boldsymbol{W}\boldsymbol{p} + b) \qquad (3.1.2)$$

Aplicando esta ecuación a cada patrón de entrenamiento se tienen las desigualdades de la tabla 3.1.2, las cuales se satisfacen plenamente, lo que implica que el problema es linealmente separable y puede ser resuelto por una red tipo Perceptron

$$p_1\begin{cases}W_{11}+W_{12}+W_{13}+W_{14}+b_1<0\\W_{21}+W_{22}+W_{23}+W_{24}+b_2<0\end{cases}\qquad p_2\begin{cases}-W_{11}+W_{12}+W_{13}+W_{14}+b_1<0\\-W_{21}+W_{22}+W_{23}+W_{24}+b_2\geq 0\end{cases}$$

$$p_3\begin{cases}W_{11}+W_{12}-W_{13}-W_{14}+b_1\geq 0\\W_{21}+W_{22}-W_{23}-W_{24}+b_2<0\end{cases}\qquad p_4\begin{cases}-W_{11}-W_{12}-W_{13}-W_{14}+b_1\geq 0\\-W_{21}-W_{22}-W_{23}-W_{24}+b_2\geq 0\end{cases}$$

$$p_5\begin{cases}W_{11}-W_{12}+W_{13}+W_{14}+b_1\geq 0\\W_{21}-W_{22}+W_{23}+W_{24}+b_2<0\end{cases}\qquad p_6\begin{cases}W_{11}+W_{12}-W_{13}+W_{14}+b_1<0\\W_{21}+W_{22}-W_{23}+W_{24}+b_2\geq 0\end{cases}$$

$$p_7\begin{cases}W_{11}+W_{12}+W_{13}-W_{14}+b_1\geq 0\\W_{21}+W_{22}+W_{23}-W_{24}+b_2<0\end{cases}$$

Tabla 3.1.2 Desigualdades que garantizan que el problema sea linealmente separable

3.1.3 Entrenamiento de la red.

A la red se le presentaran 7 patrones de la tabla 3.1.1, para los cuales dependiendo de las lecturas de los sensores se le dirá al robot que hacer específicamente y luego se probará la red con los casos restantes para comprobar la capacidad de generalización de la red neuronal ante un patrón nunca antes visto.

Los estados de lecturas de los sensores y de operación de los motores fueron designados con 1 y –1, puesto que para la convergencia del proceso de entrenamiento resulta más ventajosos propagar valores de 1 y –1 que de 1 y 0.

Debido a la naturaleza de la salida y de la entrada de la red, la función de transferencia apropiada es *hardlims*, la cual trabaja con valores bipolares.

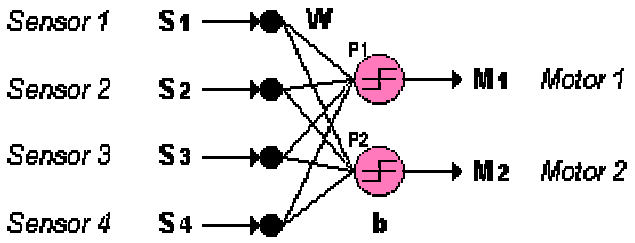

Figura 3.1.2 Red tipo Perceptrón

Se creará una red de 4 entradas con una neurona tipo Perceptrón para cada salida, teniendo así una salida bidimensional, los pesos iniciales aleatorios de la red se muestran en la tabla 3.1.3

Wi=net.IW{1,1}

	S1	S2	S3	S4
M1	0.8636	-0.1627	0.0503	0.3443
M2	-0.0680	0.6924	-0.5947	0.6762

bi=net.b{1}

M1	1
M2	1

Tabla 3.1.3 Pesos Iniciales

El siguiente código crea una red tipo Perceptrón con función de transferencia *hardlims*, dos neuronas en la salida, utiliza como patrones de entrenamiento las lecturas de los cuatro sensores almacenados en ***p*** y como patrones objetivo o salidas deseadas las acciones de ambos motores almacenados en el vector ***t***.

```
net=newp([-1 1;-1 1;-1 1;-1 1],2,'hardlims');
net.adaptParam.passes=200;
Wi;
[net,a,e]=adapt(net,P,t);
Wf=net.IW{1,1};
bf=net.b{1};
```

Los pesos finales de la red entrada que satisface todos los patrones de entrada y salida son:

Wf=net.IW{1,1}

	S1	S2	S3	S4
M1	0.8636	-4.1627	0.0503	-3.6557
M2	-4.0680	0.6924	-4.5947	4.6762

bf=net.b{1}

M1	1
M2	1

Tabla 3.1.4 Pesos finales red entrenada

La red fue simulada para la totalidad de combinaciones posibles de entrada para comprobar que no exista error en el aprendizaje de los patrones de entrenamiento y para observar su capacidad de generalización en los casos restantes

```
S1=[1 -1]; S2=[1 -1]; S3=[1 -1]; S4=[1 -1];
P=combvec(S1,S2,S3,S4)
net=newp([-1 1;-1 1;-1 1;-1 1],2,'hardlims');
Wf;
t=sim(net,P);
```

La respuesta de la red a todos los patrones de entrenamiento fue exitosa, como puede observarse en la tabla 3.1.5

	S1	S2	S3	S4	M1	M2
P1	**1**	**1**	**1**	**1**	**-1**	**-1**
P2	**-1**	**1**	**1**	**1**	**-1**	**1**
P3	**1**	**1**	**-1**	**-1**	**1**	**-1**
P4	**-1**	**-1**	**-1**	**-1**	**1**	**1**
P5	**1**	**-1**	**1**	**1**	**1**	**-1**
P6	**1**	**1**	**-1**	**1**	**-1**	**1**
P7	**1**	**1**	**1**	**-1**	**1**	**-1**

Tabla 3.1.5 Simulación de la red para los Patrones de Entrenamiento

La red fue simulada para las posibles combinaciones restantes obteniéndose los siguientes resultados:

	S1	S2	S3	S4	M1	M2
C1	-1	-1	1	1	1	1
C2	-1	1	-1	1	-1	1
C3	1	-1	-1	1	1	1
C4	-1	-1	-1	1	1	1
C5	-1	1	1	-1	-1	-1
C6	1	-1	1	-1	1	-1
C7	-1	-1	1	-1	1	-1
C8	-1	1	-1	-1	-1	1
C9	1	-1	-1	-1	1	-1

Tabla 3.1.6 Simulación de la red para las nuevas combinaciones

Las combinaciones que no hacían parte del set de entrenamiento, al ser presentadas a la red fueron aproximadas al patrón del set de entrenamiento aprendido con menor distancia euclidiana.

Para las combinaciones C1, C2 y C4 la red decidió dar marcha adelante a ambos motores; para la combinación C5 la red decidió dar marcha atrás a ambos motores, para las combinaciones C6, C7 y C9 la red decidió dar marcha adelante

a M1 y marcha atrás a M2 y para las combinaciones C2 y C8 la red decidió dar marcha atrás a M1 y marcha adelante a M2

Una red tipo Perceptrón de una sola capa es una buena solución a un problema que involucre patrones linealmente separables, en el caso de contar con patrones que no son linealmente separables se tiene la alternativa de utilizar una red Perceptrón multicapa o cambiar definitivamente de red, nótese que una red Perceptrón multicapa puede solucionar el problema de separabilidad lineal a medida que aumenta el número de capas de la red.

La capacidad de generalización de las redes neuronales juega un papel importante cuando las posibles combinaciones de patrones de entrada son tantas que resultaría imposible especificarle a un dispositivo que hacer en cada caso, puesto que la red se entrena con un número de patrones representativo y no con la totalidad de ellos, ahorrando tiempo de cómputo en la solución del problema.

En las aplicaciones desarrolladas con redes neuronales juega un papel importante la tolerancia a fallas que las caracteriza, pues en caso de fallar uno o varios sensores (como en este caso) la red siempre producirá una salida que en la mayoría de los casos es la mas acertada, debido a que la red después de un proceso de aprendizaje exitoso esta en capacidad de generalizar el comportamiento del sistema.

3.2 CONTROL DE GIRO DE UN MOTOR DE INDUCCIÓN DE JAULA DE ARDILLA

3.2.1 Descripción del problema.

Un motor de inducción tiene físicamente el mismo estator de una máquina sincrónica con diferente construcción de rotor, existen dos tipos diferentes de rotores de motor de inducción que se pueden colocar dentro del estator, rotor jaula de ardilla y rotor devanado; para esta aplicación se estudiará el rotor de jaula de ardilla, el cual consiste en una serie de barras conductoras colocadas en ranuras talladas en la cara del rotor y con sus extremos puestos en corto circuito. Este diseño hace referencia a un rotor de jaula de ardilla, porque los conductores parecen las ruedas de ejercicio en donde suelen jugar las ardillas o los ratones de laboratorio, ejemplos de este tipo de motor pueden verse en las figuras 3.2.1.1 y 3.2.1.2.

Figura 3.2.1.1 Corte típico de un motor jaula de ardilla pequeño

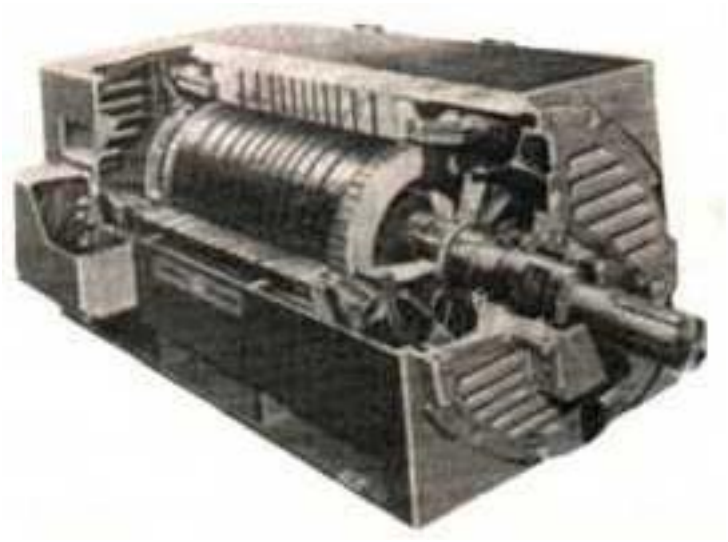

Figura 3.2.1.2 Corte típico de un motor jaula de ardilla grande

En la figura 3.2.2 se muestra un motor de inducción jaula de ardilla, en el un sistema de voltajes trifásicos ha sido aplicado al estator y un conjunto de corrientes trifásicas del estator circula en sus embobinados. Estas corrientes producen un campo magnético $\mathbf{B_s}$, el cual está girando en sentido contrario al de las manecillas del reloj. La velocidad de rotación del campo magnético está dada por

$$n_{sinc} = \frac{120 f_e}{P} \qquad (3.2.1)$$

Donde f_e es la frecuencia del sistema en Hz y P es el número de polos de la máquina. Este campo magnético rotatorio $\mathbf{B_s}$, pasa sobre las barras del rotor y les induce un voltaje. El voltaje inducido en una barra de rotor dada se obtiene mediante la ecuación

$$e_{ind} = (\boldsymbol{v}x\boldsymbol{B}) \bullet \boldsymbol{L} \qquad (3.2.2)$$

Donde $\boldsymbol{v}$= velocidad de las barras del rotor con relación al campo magnético

$\boldsymbol{B}$= densidad de flujo magnético del estator

$\boldsymbol{L}$= longitud de la barra del rotor

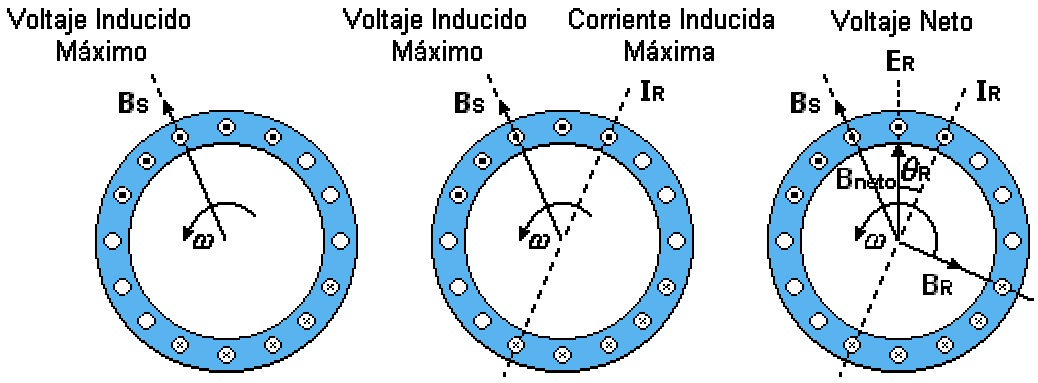

Figura 3.2.2 Desarrollo del momento de torsión en un motor de inducción

El movimiento relativo del rotor con relación al campo magnético del estator es lo que produce el voltaje inducido en una barra del rotor. La velocidad de las barras superiores del rotor, con relación al campo magnético es hacia la derecha, de tal forma que el voltaje inducido en las barras superiores estará hacia fuera de la figura y el voltaje inducido en las varillas inferiores estará hacia adentro de la figura. Esto determina un flujo de corriente hacia fuera de las barras superiores y hacia adentro de las barras inferiores, sin embargo como el conjunto del rotor es

inductivo y su corriente pico se atrasa con relación a su voltaje pico (parte central de la figura 3.2.2), el flujo de corriente del rotor produce un campo magnético $\mathbf{B_R}$. Por último, como el momento de torsión inducido en la máquina se expresa por:

$$\tau_{ind} = k * \boldsymbol{B_R} \boldsymbol{x} \boldsymbol{B_S} \qquad (3.2.3)$$

el momento resultante es contrario al sentido de las manecillas del reloj, por lo cual el rotor se acelera en esta dirección.

No obstante, la velocidad del motor tiene un límite superior finito. Si el rotor del motor de inducción girara a velocidad sincrónica, entonces sus barras permanecerían estacionarias con relación al campo magnético y no habría inducción de voltaje; si fuera igual a cero, entonces no habría corriente ni campo magnético en el rotor; sin campo magnético en éste, el momento de torsión inducido sería cero porque el rotor se frenaría como consecuencia de las pérdidas por fricción.

Un motor puede, en esta forma, acelerarse hasta cerca de la velocidad sincrónica, pero jamás podrá alcanzar exactamente la velocidad sincrónica.

En esta aplicación sugerida en Delgado [8], se requiere entrenar una red neuronal que sintetice el accionamiento correspondiente a la inversión del sentido de giro de un motor de las condiciones descritas anteriormente. En la figura 3.2.3 se hace

una representación gráfica de la secuencia de contactos que deben accionarse para dar paso a una inversión en el giro del motor.

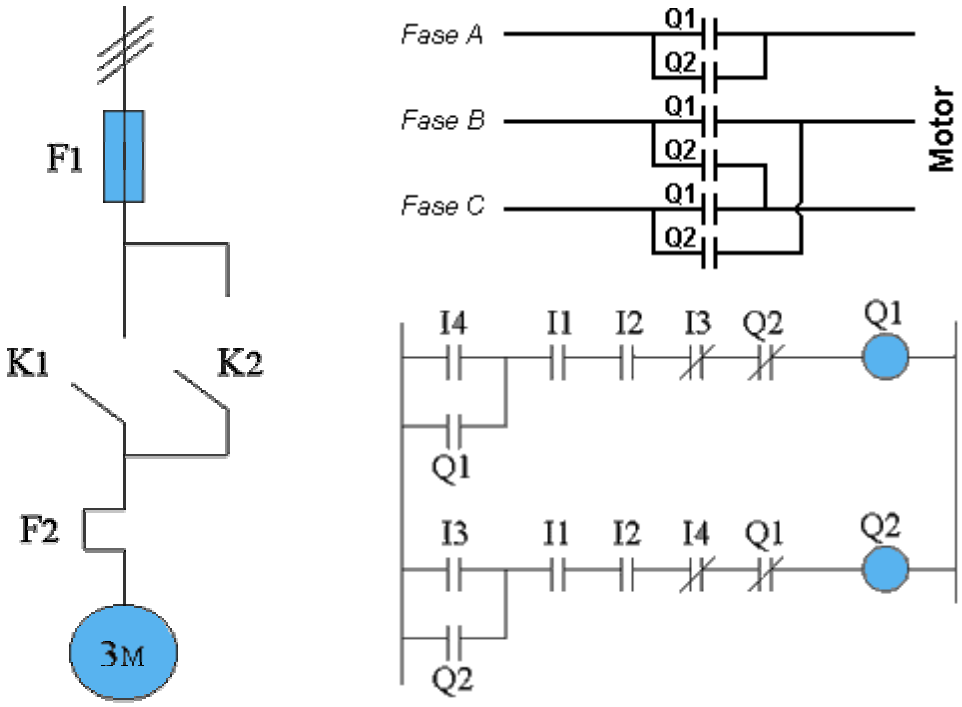

Figura 3.2.3 Diagrama Eléctrico y de Contactos

Cada elemento en la figura anterior representa:

F1: Fusible circuito principal

F2: Relé térmico (I1)

K1: Contactor derecha (Q1)

K2: Contactor izquierda(Q2)

S1: Pulsador de paro (I2)

S2: Pulsador izquierda (I3)

S3: Pulsador derecha (I4)

Las ecuaciones lógicas del circuito son:

$$Q_1 = (Q_1 - I_4) * I_1 * I_2 * \overline{I_3} * \overline{Q_2} \qquad (3.2.4)$$

$$Q_2 = (Q_2 + I_3) * I_1 * I_2 * \overline{I_4} * \overline{Q_1} \qquad (3.2.5)$$

Los valores de entrada a la red fueron determinados por Delgado [8] los cuales cubren las posibles combinaciones de contactores de la figura 3.2.3, de tal modo que su interconexión determinará el sentido de giro para el motor trifásico. Los estados de cada uno de los contactores se agrupan en la tabla 3.2.1, en donde 1 representa el contactor se encuentra activo y –1 representa que el contactor está inactivo.

P	P1	P2	P3	P4	P5	P6	P7	P8	P9	P10	P11	P12
I1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
I2	-1	-1	-1	-1	-1	-1	-1	-1	1	1	1	1
I3	-1	-1	-1	-1	1	1	1	1	-1	-1	-1	-1
I4	-1	-1	1	1	-1	-1	1	1	-1	-1	1	1
	-1	1	-1	1	-1	1	-1	1	-1	1	-1	1

P	P13	P14	P15	P16	P17	P18	P19	P20	P21	P22	P23	P24	P25
I1	-1	1	-1	-1	1	1	1	1	1	1	1	1	1
I2	1	1	1	1	-1	-1	-1	-1	-1	-1	-1	-1	1
I3	1	1	1	1	-1	-1	-1	-1	1	1	1	1	-1
I4	-1	-1	1	1	-1	-1	1	1	-1	-1	1	1	-1
	-1	1	-1	1	-1	1	-1	1	-1	1	-1	1	-1

Tabla 3.2.1 Entradas a la red

La quinta fila de la tabla 3.2.1 representa un sensor que se ha implementado para determinar el estado de giro en que se encuentra el motor al momento de accionamiento para cambio. Este sensor es necesario para determinar si el cambio debe realizarse o no, es decir, si dada una configuración cualquiera el motor se encuentra girando hacia la derecha y el contactor Q_1, que da la orden de cambiar a derecha y que para la red neuronal es representada por un 1, es accionado, la red no tendrá que dar al motor ninguna orden de cambio; si por el contrario para esta misma configuración es accionado el contartor Q_2, que da la orden de cambiar a izquierda y que para la red es representado por –1, la red inmediatamente debe dar la orden de cambio de giro, y el motor debe pasar de girar hacia la derecha a girar hacia la izquierda.

Siendo 1 el valor que representa el giro del motor por derecha y –1 el valor que representa el giro por izquierda, los valores esperados de salida se visualizan en la tabla 3.2.2:

t	t1	t2	t3	t4	t5	t6	t7	t8	t9	t10	t11	t12
	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1

t13	t14	t15	t16	t17	t18	t19	t20	t21	T22	t23	t24	t25
-1	1	-1	-1	1	1	1	1	1	1	1	1	1

Tabla 3.2.2 Salidas esperadas de la red

3.2.2 Justificación del tipo de red.

En Ingeniería Eléctrica hay muchos tipos de problemas cuya solución depende sólo de dos estados: on - off, si - no, 1-0, alto - bajo, o como en el caso que se estudia izquierda - derecha, este tipo de problemas divide el espacio solución en dos regiones, cada una de ellas limitada por un hiperplano que determina el límite donde empieza una y termina la otra.

La regla de aprendizaje del Perceptrón, es hábil para dividir el espacio de entrada en dos regiones, más que una línea existe un hiperplano que separa físicamente el espacio en cada una de las categorías de entrada, matemáticamente este hiperplano se expresa así:

$$\boldsymbol{w}^T \boldsymbol{p} + b = 0 \qquad (3.2.6)$$

Los problemas que permiten que un hiperplano de estas características separe sus categorías de entrada en dos regiones diferentes son llamados problemas linealmente separables y pueden ser solucionados por una red tipo Perceptrón.

Expresar los datos de la tabla 3.2.1 gráficamente para saber si los valores de entrada a la red son linealmente separables, es imposible debido al orden de estos datos los cuales generan una gráfica en R^5, así que es necesario emplear la técnica alternativa del capítulo 2, en la cual por medio de desigualdades se

determina la separabilidad lineal de un conjunto de vectores dados y se comprueba la factibilidad de emplear una red Perceptrón en su solución.

3.2.3 Entrenamiento de la red.

El proceso de entrenamiento de esta red fue realizado por medio del Matlab, utilizando las funciones de la herramienta de redes neuronales. Como patrones de entrada a la red se emplearon los datos de la tabla 3.2.1, y sus salidas asociadas corresponden a la tabla 3.2.2.

La red fue generada por medio de la función *newp*, de acuerdo a la sintaxis referida en el anexo 1 y escogiendo la función de transferencia *hardlims* de acuerdo con la naturaleza de los datos de salida

```
net=newp([-1 1;-1 1;-1 1;-1 1;-1 1],1,'hardlims');
```

Los valores iniciales para la matriz de peso W se generaron aleatoriamente y el valor inicial para la ganancia b, se escogió arbitrariamente como 1

```
net.IW{1,1}=rands(1,5);
net.b{1}=[1];
```

Los comandos básicos para entrenar una red Perceptrón se presentan a continuación; mediante los valores fijados se logró un buen desempeño de la red

```
net.adaptParam.passes=50;
[net,a,e]=adapt(net,p,t);
```

Los resultados obtenidos se agrupan en la siguiente tabla:

Pesos =

6.0056	2.4189	1.8578	1.6092	1.3793

Ganancia =

-1

Tabla 3.2.3 Valores finales de los parámetros de la red finalizado el proceso de entrenamiento

El error entregado por esta red fue cero, demostrando no sólo la efectividad de la red tipo Perceptrón para clasificar patrones linealmente separables, sino su facilidad de implementación, ya que sólo requiere tener clarificados la naturaleza de los datos de entrada a la red y la respuesta asociada a cada uno de ellos.

3.3 FILTRO ADAPTIVO

3.3.1 Descripción del problema.

Sin duda la principal aplicación de la red Adaline está en el campo del procesamiento de señales, en concreto para el diseño y realización de filtros que eliminen el ruido de señales portadoras de información. Como filtro adaptivo se ha utilizado esta red en numerosas aplicaciones, destacándose su uso en filtros de ecualización adaptivos en modems de alta velocidad y canceladores adaptivos del eco para filtrado de señales en comunicaciones telefónicas de larga distancia y comunicaciones vía satélite. Una de las primeras aplicaciones de redes neuronales que tuvo éxito en la industria fue el Adaline para la eliminación de ecos en circuitos telefónicos.

Por otro lado, los filtros adaptivos se pueden usar para predecir el valor futuro de una señal a partir de su valor actual basándose en un aprendizaje en el que se emplea como entrada el valor retardado de la señal actual y como salida esperada, el valor actual de la señal, el filtro intentará minimizar el error entre su salida y la señal anterior. Una vez el filtro predice correctamente la señal actual, basándose en la señal anterior, se puede utilizar directamente la actual como entrada sin el retardo, el filtro realizará una predicción del valor futuro de la señal.

Otro ejemplo es el filtro adaptivo utilizado para modelar las respuestas de un sistema basándose en las señales de entrada, en este caso las entradas al filtro

son las mismas que las del sistema, ajustando los pesos durante el aprendizaje en función de la diferencia entre su salida y la del sistema, éste será el modelo de filtro adaptivo utilizado en esta aplicación.

Existen sistemas en los cuales es posible determinar la fuente de ruido, por ejemplo cuando son los mismos elementos de medición los que introducen señales de ruido a la señal original; este ruido es ocasionado por los niveles de voltaje y de frecuencia del sistema alimentador de los aparatos, el cual es imprescindible en el proceso.

En la presente aplicación, una señal de ruido ha sido introducida por los cables de conexión de los aparatos y se ha mezclado con la señal que se desea medir, lo que en este caso es crítico, ya que no se conoce la forma exacta de la señal de interés y por tanto no es posible determinar cuál debe ser la señal correcta de salida.

Aprovechando que la fuente de ruido ha sido completamente determinada, se utilizará este conocimiento para implementar un procedimiento de filtrado que por medio de un filtro adaptivo permita recuperar la señal original.

La figura 3.3.1, es un esquema de la disposición que se dará al filtro dentro del contexto general que se ha dispuesto para solucionar el problema. Las variables son:

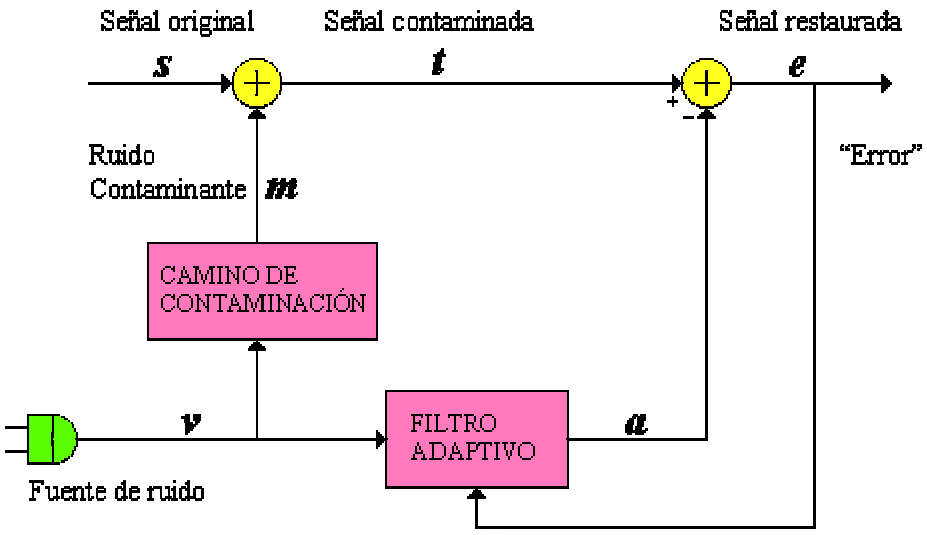

Figura 3.3.1 Diagrama de Bloques de un filtro adaptivo

s: Señal de interés, es decir es la señal que se desea medir, aunque se conoce su naturaleza, su forma es indeterminada.

v: Fuente de poder de los instrumentos de medida, por sus características de potencia y frecuencia es una fuente de ruido.

m: La señal *v*, es indispensable en el proceso de medición ya que sin ella los aparatos de medida no funcionarían; *v* es la fuente detectada de ruido, pero su contribución se limita a la parte inducida en los cables de conexión, que se convierten en el camino de contaminación; de esta forma la señal *m* es la fuente de ruido que afecta realmente la señal original *s*.

t: Señal resultante de la suma de la señal de interés *s* y de *m* que es el porcentaje efectivo de ruido que afecta la medición.

a: Salida del filtro.

e: Diferencia entre la señal contaminada t y la señal que entrega el filtro adaptivo, si el filtro realiza un buen trabajo, esta señal debe ser la señal restaurada sin ruido. Durante el proceso de aprendizaje la señal e servirá como referencia para determinar el desempeño del filtro.

La señal ***v*** es una señal por lo general de 60Hz, que en este caso alimenta el filtro adaptivo, los parámetros del filtro deben ajustarse para minimizar la señal de error ***e***, de esta forma y de acuerdo con el esquema de la figura 3.3.1 podría pensarse que la salida esperada del filtro ***a*** es la señal contaminada ***t***, pero debe recordarse que la única fuente efectiva de ruido es la señal ***m*** por lo tanto, lo que realmente se espera del filtro es que reproduzca sólo la parte de ***t*** que afecta la medición ($t = s+m$), o sea ***m***. Por consiguiente la salida del filtro ***a*** debe ser cada vez más cercana a ***m*** para que la señal de error ***e***, se aproxime a la señal no contaminada ***s***.

La entrada al filtro es el valor de la fuente de ruido, que en este caso se asumirá como una señal senoidal uniformemente distribuida entre -0.2 y 0.2 para conservarla dentro de límites observables; la frecuencia de la onda es de 60 Hz, y la frecuencia de muestreo es de 180 Hz.

$$v(k) = 1.2 sin\left[\frac{2\pi k}{3}\right] \qquad (3.3.1)$$

El estudio de los procesos de filtrado se ha basado en un análisis en el dominio de la frecuencia, determinado por las series de Fourier, así para un sistema de filtrado dado, una señal de entrada $f(t)$ que produce una salida $r(t)$ característica del sistema en el dominio del tiempo, se obtendrá un análogo en el dominio de la frecuencia tal que, $F(w)$ producirá una señal de salida $F(w)H(w)$.

En la figura 3.3.2, se observa como el sistema actúa como un filtro de las diferentes componentes de frecuencia. Para garantizar que el filtrado sea exitoso, el filtro debe atenuar igualmente todas las componentes de frecuencia, la respuesta debe ser una réplica de la señal de entrada no en magnitud, pero si en forma; en general puede haber un retraso de tiempo asociado con esta réplica, por lo tanto, se habrá filtrado exitosamente una señal $f(t)$, si la respuesta es $k * f(t - t_0)$; la respuesta es una réplica de la entrada con magnitud k veces la señal original y un retraso de t_0 segundos.

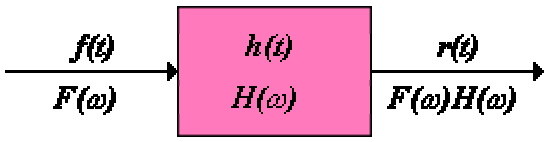

Figura 3.3.2 Representación de un sistema de filtrado en el dominio del tiempo y en el dominio frecuencial

Para el filtro adaptivo $v(t)$ es la señal de entrada al filtro y la respuesta debe ser $k * v(t - t_0)$ que en el dominio de la frecuencia, equivale a desfasar la función original 90^0 y multiplicarla por un factor k, el cual se escogió como 1/10 para simplificar el proceso de visualización.

Como se indicó anteriormente, la salida del filtro debe ser la señal m para que al final del proceso pueda obtenerse la señal restaurada, por lo tanto la forma de la señal m será:

$$m(k) = 0.12 sin\left[\frac{2\pi k}{3} + \frac{\pi}{2}\right] \qquad (3.3.2)$$

Para tener una idea clara de la relación entre m y v, la figura 3.3.3 ilustra la proporción de la señal original de ruido y de la señal efectiva de ruido que afecta la medición.

En la gráfica de la parte superior de la figura 3.3.3, se ve como v es una señal senoidal de magnitud 1.2, mientras que la gráfica inferior muestra la señal m que alcanza solo la décima parte de v con una magnitud de 0.12 y desfasada 90^0 con respecto a la señal v, éstas dos ondas representan los patrones de entrenamiento de la red, siendo v la entrada y m la salida deseada.

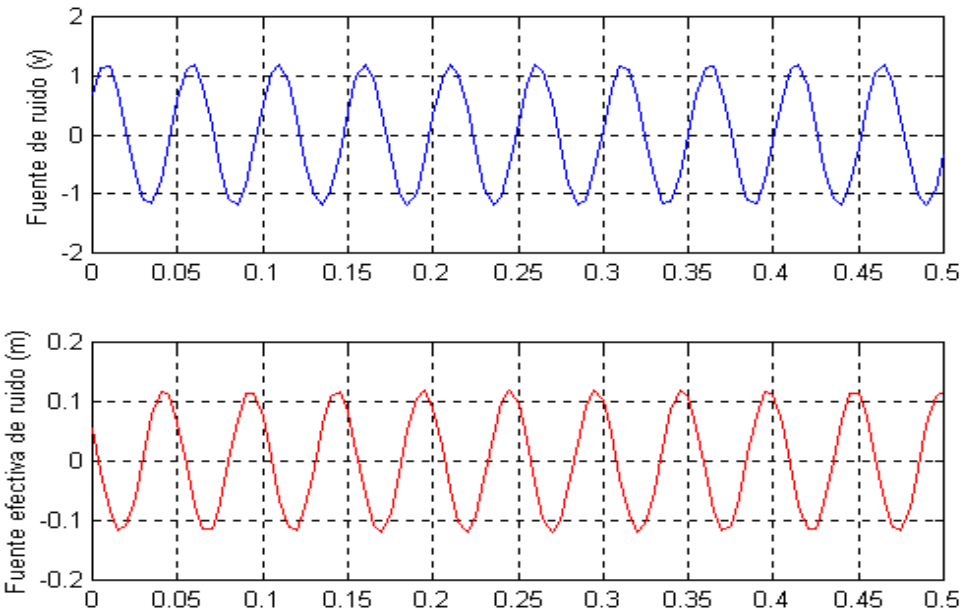

Figura 3.3.3 Señal original de ruido y señal que afecta el proceso de medición

3.3.2 Justificación del tipo de red.

Como puede notarse, la red Adaline tiene la estructura básica del Perceptrón, la diferencia es que su función de transferencia es del tipo lineal, pero por poseer una estructura similar presenta la misma limitación de la red tipo Perceptrón de poder resolver solo problemas linealmente separables.

A pesar de sus limitaciones innatas, la red Adaline es ampliamente usada en los procesos de filtrado combinándola con retardos en línea a su entrada, que mejoran la calidad del filtrado

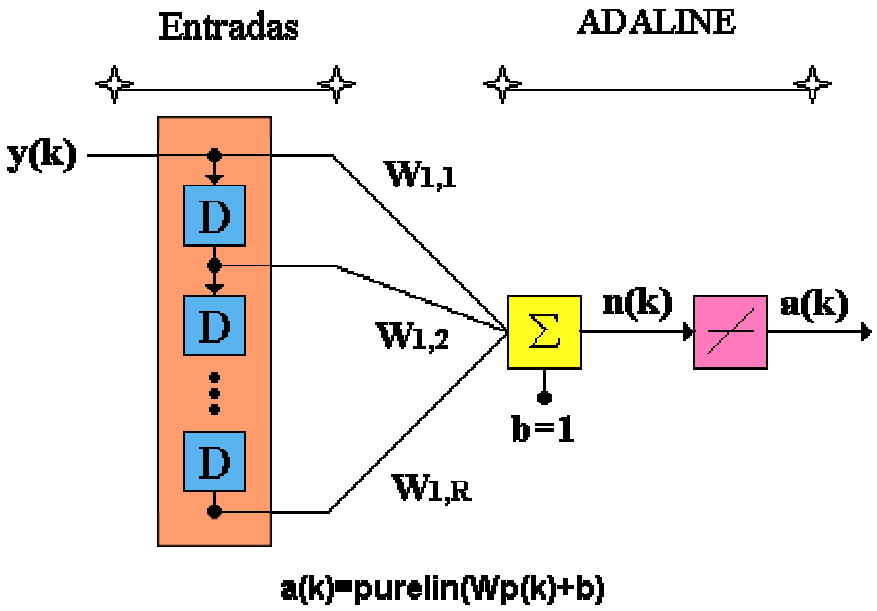

Figura 3.3.4 Estructura de un filtro adaptivo con una red Adaline

Cuando no es posible determinar la forma de onda de la señal original, ya que este es precisamente el propósito de la medición, un filtro adaptivo se hace bastante útil, pues como su nombre lo dice se adapta a cualquier forma de señal que se le presente, sobrepasando así las limitaciones de los filtros tradicionales.

3.3.3 Entrenamiento de la red.

El objetivo del algoritmo diseñado en Matlab es lograr obtener la señal *m* a la salida del filtro, minimizando el error medio cuadrático hasta reproducir una buena copia de la señal original *s*.

La red es creada con la función de las herramientas de redes neuronales *newlin,* que genera una nueva red tipo Adaline, a la cual se le han introducido cuatro

retardos, para conformar el filtro adaptivo que filtrará la señal de entrada. El valor de los retardos fue escogido por prueba y error, escogiendo aquel número que presentará un mejor rendimiento para la red.

```
net=newlin([-1,2],1);
net.inputWeights{1,1}.delays=[0 1 2 3 4];
```

Los valores de los pesos iniciales y de las ganancias de la red son inicializados aleatoriamente, con ayuda de la función *rands*

```
net.IW{1,1}=rands(1,5);
net.b{1}=[0];
pi={1 2 3 4};
```

Los valores correspondientes a la entrada y a la salida de la red se obtuvieron evaluando la ecuación (3.3.1) para valores desde $\frac{2\pi}{3}$ hasta 4π. La red se entrenó para 5000 iteraciones, mediante las cuales el filtro logró una excelente aproximación de la función original.

```
net.adaptParam.passes=5000;
[net,y,E,pf,af]=adapt(net,p,T,pi);
```

Para entrenar la red se generaron 101 puntos, algunos de los cuales pueden verse en la siguiente tabla:

	1	2		23		51	52		78		101
Entrada a la red (***v***)	0.6967	1.1345	....	1.1926	....	0.2119	0.8583	....	-1.1165	....	-0.3137
Valor esperado (***m***)	0.0561	-0.0159	....	-0.0653	....	0.0961	0.0365	....	0.0107	....	0.1176
Valor entregado (***a***)	-0.2592	-0.2379	....	-0.0385	....	0.0766	0.0293	....	0.0163	....	0.1098

W1=net.IW{1,1}

	I1	I2	I3	I4	I5
N1	-0.0405	-0.0127	-0.0319	-0.0389	-0.0097

b1=net.b{1}

N1	0.0032

Mse 0.0022

Tabla 3.3.1 Resultados del proceso de entrenamiento

Para poder juzgar el trabajo realizado por el filtro, la figura 3.3.5 muestra la señal original y la señal restaurada, que permiten comprobar las bondades del filtro adaptivo.

En la figura 3.3.5 se observa como la señal e (verde) reproduce la señal original s (naranja) logrando una excelente aproximación, a partir de la cual puede retomarse el proceso de medición, pues en este punto cualquier tratamiento que se le haga a la señal de interés es altamente confiable, ya que ésta se encuentra libre de ruido.

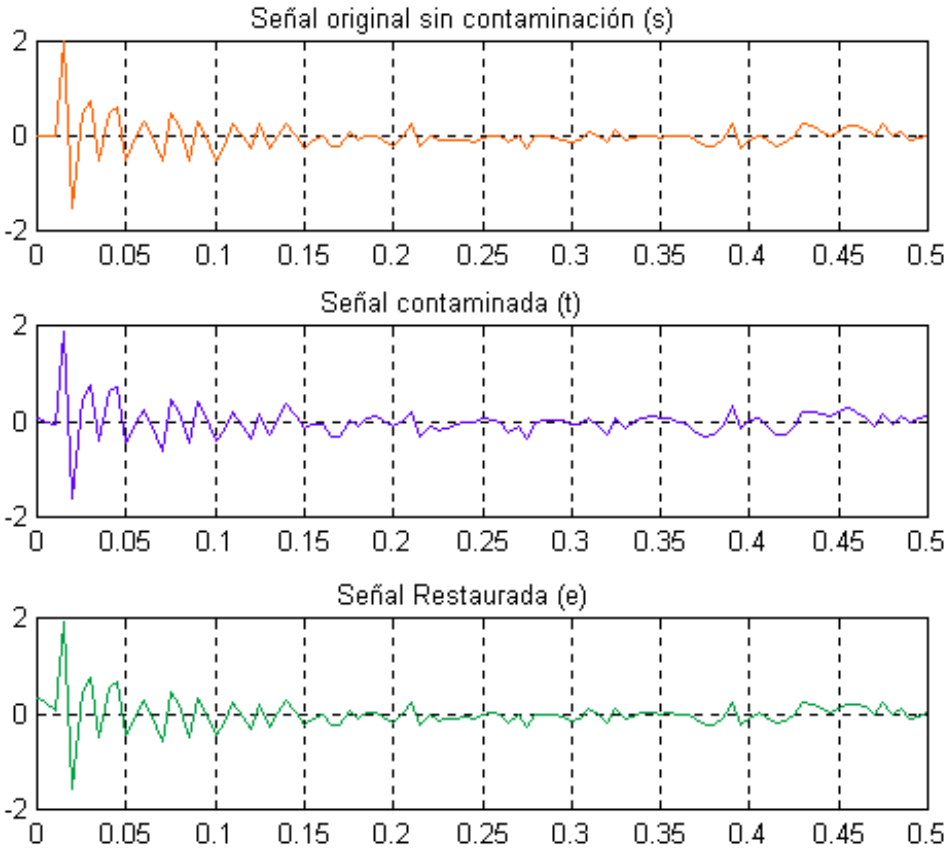

Figura 3.3.5 Señal recuperada por el filtro

Al final del proceso iterativo el máximo error entregado por la red equivale a 1.41e-07, este es un valor bastante aceptable teniendo en cuenta que el error difícilmente será cero, puesto que el algoritmo LMS emplea un valor aproximado para el gradiente, que como se dijo en el capítulo 2 recibe el nombre de gradiente instantáneo, en lugar del valor real para realizar la actualización de los pesos; este valor del gradiente es una versión distorsionada del verdadero gradiente que ocasionará que los pesos sigan variando suavemente, a pesar de que el error medio cuadrático haya alcanzado el mínimo valor posible.

Es importante diferenciar entre el valor del error del filtro adaptivo, el cual mide su desempeño y corresponde al error medio cuadrático descrito anteriormente, y e la señal de error del sistema en general, con la cual se espera reproducir la señal original sin contaminación s, trabajo que depende a su vez del desempeño del filtro.

En esta red todos los parámetros están muy ligados, por ejemplo en el esquema general de la figura 3.3.1 se observa como la salida e (correspondiente a la diferencia entre a la salida del filtro y m la señal que se esperaba que este entregará para poder reproducir la señal original s a la salida del sistema), realimenta el filtro adaptivo convirtiéndose en un factor decisivo en el proceso de actualización de los pesos de la red, por otro lado la dimensión del vector de pesos tiene una influencia directa en el tiempo necesario de entrenamiento, por lo que generalmente se debe tomar un compromiso entre este aspecto y la aceptabilidad de la solución (normalmente se mejora el error aumentando el número de pesos).

El valor del parámetro α tiene una gran influencia sobre el entrenamiento. Si α es demasiado grande, es posible que la convergencia no se produzca debido a que se darán altos en torno al mínimo sin alcanzarlo. Si α es demasiado pequeño, se alcanzará convergencia pero a costa de una etapa de aprendizaje más larga.

3.4 PREDICCIÓN DE CONSUMO DE CARGA

3.4.1 Descripción del problema

Un sistema de energía eléctrica debe abastecer de energía a todos los puntos de carga con una buena calidad del servicio. Por lo tanto un sistema eléctrico confiable, el cual asegura buena calidad, debe contar con las siguientes características:

- Entregar energía en forma continua a todos los puntos de carga.
- Los límites de la frecuencia y la tensión deben estar dentro de valores tolerables.
- El sistema debe operar en la medida de lo posible, con costos mínimos y con un mínimo de alteraciones ambientales o ecológicas.

Estas características pueden adquirirse por medio de una planeación exhaustiva del sistema, que permita conocer no sólo su estado actual, sino también las medidas que deben adoptarse para condiciones futuras.

Una de las herramientas útiles en el planeamiento de un sistema eléctrico es la predicción del consumo de carga, la cual permite conocer de antemano la necesidad de expansión del sistema; la finalidad de la predicción siempre será el

mejoramiento del servicio, convirtiéndose en uno de los primeros pasos en cualquier proceso de planeamiento de un sistema eléctrico.

Al hablar de predicción de carga, resulta útil aclarar que como carga se asume todo equipo que demanda energía del sistema de energía eléctrica, tales como lámparas, electrodomésticos, motores eléctricos, hornos eléctricos, etc. De esta manera se tienen varios tipos de carga:

- Motores en general
- Equipos de calentamiento
- Equipos electrónicos
- Equipo de iluminación

Desde el punto de vista del sistema de energía eléctrica, las cargas pueden ser separadas en tres (3) grupos funcionales:

- Cargas domiciliarias
- Cargas industriales
- Cargas comerciales

Estas cargas presentan características muy diferentes con relación al tamaño, simetría (1Φ o 3Φ), constancia de la carga y el período de funcionamiento.

La predicción de consumo de carga refleja las necesidades futuras de una población; esta previsión debe ser lo más ajustada a la realidad, ya que unos valores inferiores a los reales causarán deficiencias en la prestación del servicio en el futuro y un pronóstico de necesidades superior al real, motiva la inversión prematura en instalaciones que no tendrán un aprovechamiento inmediato.

La proyección del suministro de energía se hace con base en el consumo, aplicando porcentajes de pérdidas que pueden obtenerse de un análisis de los registros históricos (que normalmente se presentan en forma estadística), o por similitud con otros sistemas. En general las pérdidas tienden a disminuir a causa de las mejoras progresivas, que se introducen en los sistemas de transmisión, subtransmisión y distribución. En forma similar al consumo de energía, la proyección de la demanda pico se obtiene sumando las demandas máximas coincidentes en hora pico [37].

En la tabla 3.4.1 se observa el comportamiento típico de consumo de carga de una población, a la cual se realizó un seguimiento hora a hora durante una semana tipo. Se ha tomado el día lunes como 1 y en su orden el día domingo como 7, se asumió la una de la mañana como la hora 0 y las doce de la noche como la hora 23. Los datos correspondientes al consumo de la población se encuentran en (kW).

HORA	0	1	2	3	4	5	6	7	8	9	10	11
Lunes	1,2300	1,0889	1,0289	0,9879	0,9879	1,1050	1,3729	1,6649	1,6649	2,1569	2,3230	2,3659
Martes	1,6049	1,4389	1,3631	1,3559	1,3439	1,3890	1,5699	1,7750	2,0180	2,1900	2,3359	2,3630
Miércoles	1,6630	1,4689	1,3890	1,3751	1,3611	1,4140	1,6040	1,8009	2,0739	2,2301	2,3649	2,3990
Jueves	1,7299	1,5159	1,4320	1,3931	1,3909	1,4310	1,6140	1,8170	2,0989	2,2260	2,3810	2,3741
Viernes	1,7129	1,4569	1,3461	1,2880	1,2331	1,1911	1,1570	1,1700	1,2139	1,3370	1,4799	1,5740
Sábado	1,7130	1,2821	1,1820	1,1220	1,0961	1,1059	1,1710	1,2751	1,4121	1,5450	1,7110	1,7410
Domingo	1,4140	1,3250	1,2249	1,2239	1,1240	1,0191	0,9989	0,9989	0,9790	1,0150	1,1271	1,2271

HORA	12	13	14	15	16	17	18	19	20	21	22	23
Lunes	2,3731	2,2311	2,1560	2,2080	2,2949	2,3741	2,5000	2,4340	2,3560	2,0000	1,9890	1,8080
Martes	2,3359	2,1560	2,0799	0,1651	2,2551	2,3671	2,4770	2,4310	2,3540	2,2100	1,7085	1,7000
Miércoles	2,3580	2,2000	2,1231	2,1749	2,2049	2,3349	2,4640	2,3780	2,4140	2,0040	1,8582	1,7071
Jueves	2,3021	2,1459	2,0581	2,0809	2,1651	2,2380	2,2820	2,1540	2,1020	1,9950	1,9040	1,8590
Viernes	1,5951	1,5771	1,5629	1,5320	1,5440	1,6380	1,7310	1,7480	1,7921	1,8321	1,8620	1,7930
Sábado	1,7129	1,6200	1,1570	1,5831	1,6251	1,6251	1,8950	1,9040	1,9310	1,9360	1,9580	1,748
Domingo	1,2950	1,3130	1,2909	1,2600	1,2669	1,3631	1,1530	1,6020	1,6440	1,6150	1,5030	1,4990

Tabla 3.4.1 Datos de consumo de carga de una población durante una semana

Estos datos son una recopilación de un promedio histórico de diferentes años, del consumo de la población escogida como muestra, con ellos podemos determinar la tendencia del consumo de los usuarios de esta población. El objetivo es entrenar una red neuronal que aprenda los datos anteriores y a partir de ellos esté en capacidad de predecir lo que sucederá en cualquier día de la semana a una hora determinada en años futuros.

El comportamiento de estos usuarios se visualiza en la figura 3.4.1, en donde puede notarse las tendencias que pueden llegar a causar congestión en la central en caso de no preverse con anticipación:

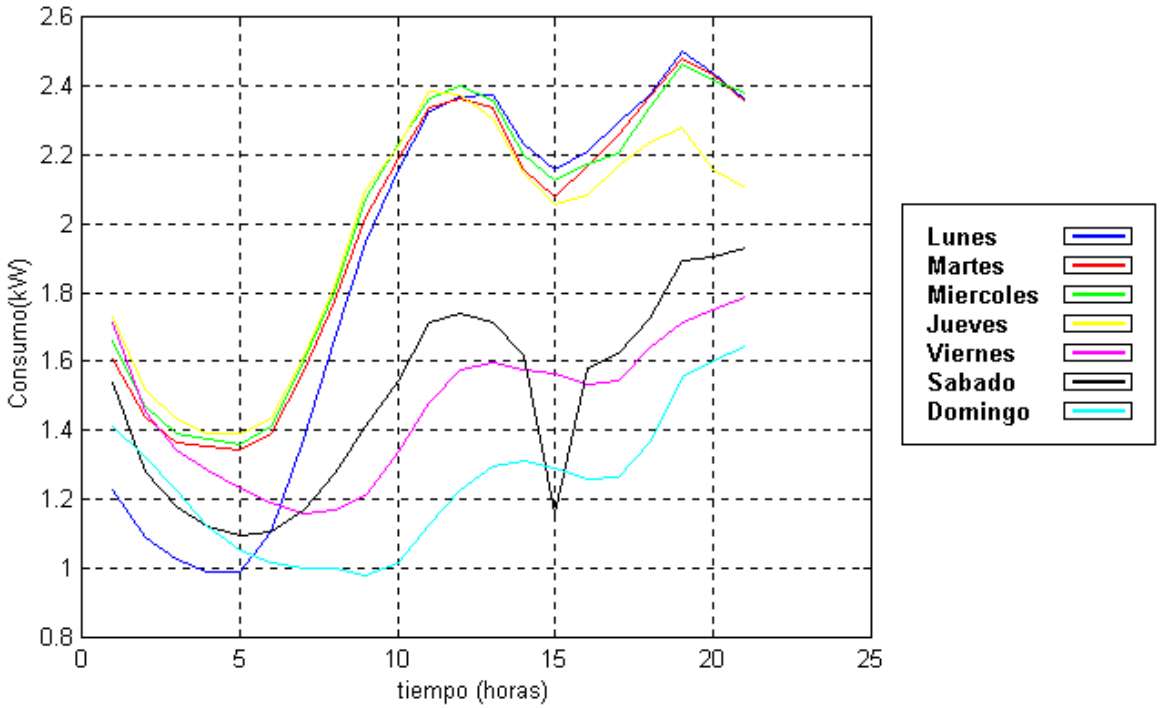

Figura 3.4.1 Curvas de Carga

3.4.2 Justificación del tipo de red.

El algoritmo Backpropagation es un algoritmo de aprendizaje supervisado, el cual necesita conocer cuál es la salida esperada (columnas interiores de la tabla 3.4.1) asociada a cada una de las entradas (columnas referentes al día y a la hora en la tabla 3.4.1), que actualiza pesos y ganancias siguiendo la regla de pasos descendientes descrita en la sección 2.3.3

$$W^m(k+1) = W^m(k) - \alpha s^m \left(a^{m-1}\right)^T \qquad (3.4.1)$$

$$b^m(k+1) = b^m(k) - \alpha s^m \qquad (3.4.2)$$

Una de las mayores ventajas de las redes multicapa, y en especial de la red Backpropagation, es que pueden aproximar cualquier función si se escoge una adecuada configuración para la red [16] y un adecuado número de neuronas en la capa oculta, escogencia que depende de la experiencia del desarrollador de la red. La red Backpropagation es un excelente aproximador de funciones, aunque es imposible determinar una configuración exacta de la red para cada aplicación.

El proceso de aprendizaje no es fijo para ninguna red neuronal, el éxito consiste en probar con diferentes configuraciones hasta obtener la respuesta deseada; para esta aplicación se escogió una red 2:12:8:1, es decir que para un vector de entrada de dos dimensiones y esperando una sola salida de red, se tienen 12 neuronas en la primera capa oculta y 8 neuronas en la segunda capa oculta.

Un esquema de la red puede observarse en la figura 3.4.2, la cual muestra las características de los datos de entrada y salida de la red. Allí puede verse que es necesario ingresar a la red el día y la hora para los cuales se desea conocer el valor de demanda pico de acuerdo a la convención de la tabla 3.4.1, las condiciones de entrada a la red pueden ser tan complejas como se quiera, es decir, la entrada puede convertirse fácilmente en un vector de cuatro componentes que incluya además del día y la hora, el mes y el año para los cuales se requiere predecir el valor de demanda; existen también otras opciones como convertir la entrada en un vector de tres componentes donde se incluya hora, día y tipo de carga para el cual se desea prever el consumo.

La elección de los patrones de entrada debe realizarse dependiendo de las necesidades explícitas que se tengan en el momento de hacer la predicción de carga, de la forma en que vaya a procesarse la información de salida de la red y de la cantidad y calidad de la información disponible.

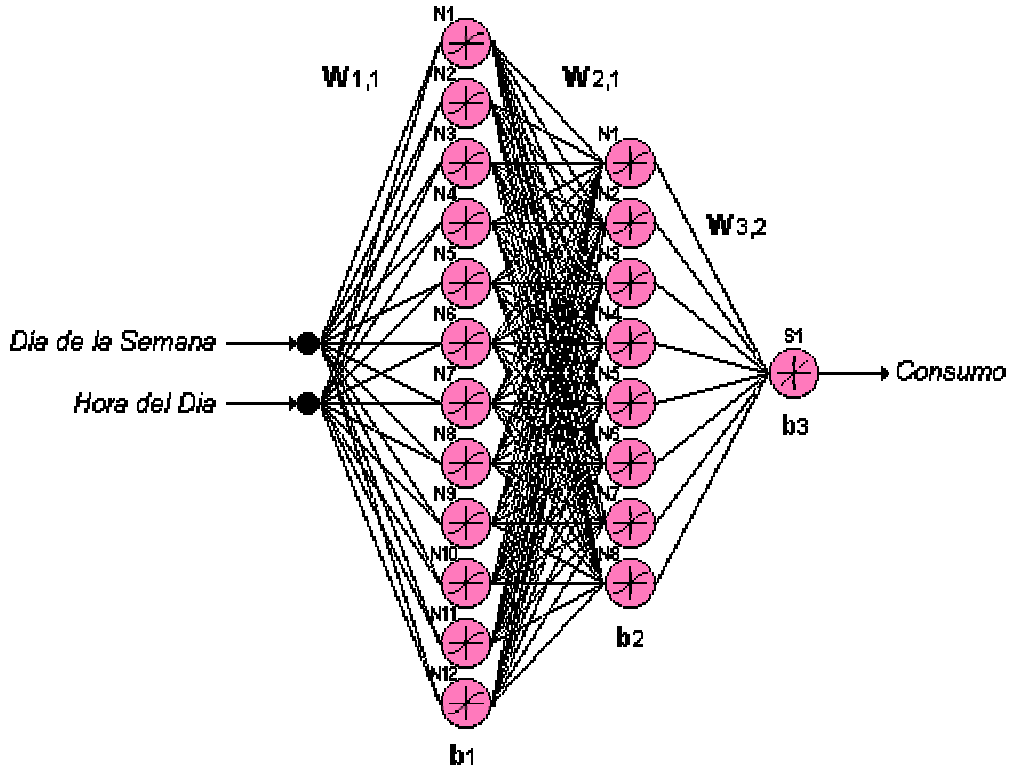

Figura 3.4.2 Red Backpropagation 2:12:8:1 para predicción del consumo de carga

Cualquier cambio que se realice en los patrones de entrenamiento exige una codificación diferente del vector de entrada y a su vez cambia las condiciones generales de la red, pero el proceso de entrenamiento sigue siendo igual. En esta aplicación se utilizó la configuración de la figura 3.4.2 porque el objetivo es mostrar el funcionamiento general de una red Backpropagation, mostrando las bondades de este tipo de red para aproximar funciones (en este caso las curvas de carga de

la población en estudio) y su gran capacidad para predecir comportamientos futuros de patrones nunca antes presentados en la etapa de aprendizaje.

3.4.3 Entrenamiento de la red

El código de entrenamiento para esta red, es desarrollado con base en la herramienta de redes neuronales del Matlab. La red es creada mediante el comando *newff* para creación de redes Backpropagation, con las siguientes características:

```
net=newff([0 1;0 1],[12,10,1],
         {'tansig','tansig','purelin'},'trainlm');
```

Los valores de iniciación de la matriz de pesos se generaron aleatoriamente; después de varias pruebas, los parámetros que determinan el entrenamiento se fijaron en los siguientes valores, mediante los cuales se alcanzó el rendimiento óptimo de la red.

```
net.trainParam.show =10;
net.trainParam.epochs =1000;
net.trainParam.goal =1e-5;
net.trainParam.lr=0.075;
net.trainParam.mem_reduc =1;
net.trainParam.min_grad =1e-12;
```

Una explicación del significado de los comandos anteriores, puede encontrarse en el anexo A.

Los valores de entrada a la red se agruparon en el vector de dos entradas $\boldsymbol{p}$, (tabla 3.4.1), las cuales describen el día y la hora en que desea saberse el valor de consumo. La red se entrenó sólo con 180 valores de entrada de los 206 que conforman todo el set de entrenamiento, estos valores fueron normalizados porque de esa forma se obtuvo una mejor generalización de la red en el proceso de aprendizaje; el proceso de normalización consiste en dividir cada uno de los valores de demanda de la tabla 3.4.1 por el valor de demanda máximo, de tal forma que el mayor valor de entrada a la red será uno:

$$Demanda\ \ normalizada = \frac{Valor\ de\ la\ demanda}{Demanda\ \ máxima}$$

El valor esperado, el valor de la potencia pico en un instante determinado, es representado por el escalar t

```
t=t';
[net,tr]=train(net,p,t);
```

Los restantes 26 valores de los patrones de entrenamiento se tomaron como valores de prueba del rendimiento de la red y se agruparon en el vector $\boldsymbol{p}_1$, siendo $\boldsymbol{t}_1$ el vector de salidas esperadas.

```
p1=p1';
a = sim(net,p1);
```

Luego de varias iteraciones el error cayó por debajo de $2x10^{-5}$, el desempeño del error medio cuadrático puede observarse en la figura 3.4.3.

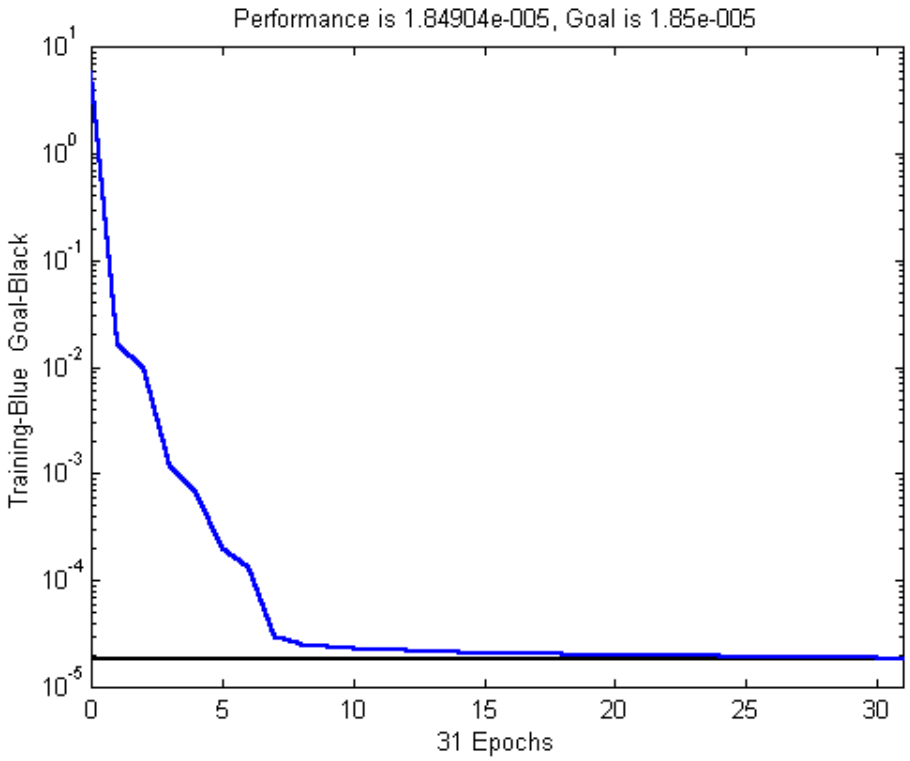

Figura 3.4.3 Iteraciones Vs Error en una predicción de consumo de carga

Los parámetros de la red entrenada se presentan en la tabla 3.4.2

W1=net.IW{1,1}

	1	**2**
N1	-21,9482	3,0064
N2	-0,4403	14,0061
N3	-30,1225	-14,4532
N4	6,8063	9,6593
N5	-0,6548	10,661
N6	-32,5167	-16,0543
N7	-31,5269	28,1193
N8	-26,5558	-10,0917
N9	2,4681	-26,1821
N10	7,306	1,6967
N11	-0,0242	9,3016
N12	-5,2355	0,211

W2=net.LW{2,1}

	1	2	3	4	5	6
N1	-0,3267	18,5006	1,4482	7,3519	5,56	-63,61
N2	-1,7789	-34,377	132,102	56,1894	-36,79	5,4474
N3	2,9001	-1,6303	31,2892	2,0873	-3,0091	-1,3828
N4	-0,8462	8,6876	4,0463	6,0138	2,3158	-0,2587
N5	12,0037	0,1605	60,885	148,871	2,5194	106,22
N6	-98,9806	105,356	115,699	0,5065	95,0003	-17,001
N7	-5,6485	136,978	15,7418	43,3306	-13,133	-23,351
N8	-0,7708	-80,889	1,397	100,628	5,1579	-0,1767

	7	8	9	10	11	12
N1	72,191	-27,627	42,4595	0,9163	-2,2968	57,8689
N2	180,2947	-253,47	-92,087	0,066	3,5167	-77,476
N3	3,3586	2,9584	1,2985	-0,0981	0,0207	-1,3115
N4	29,6843	0,8862	-10,697	0,4033	-1,1097	25,1813
N5	10,1653	0,8031	-30,127	0,2075	2,0099	3,7721
N6	46,6346	112,561	-28,56	6,8125	96,3854	172,322
N7	-58,043	-16,587	-10,875	-35,975	0,3041	74,7964
N8	94,3186	-1,8353	-1,4035	0,0731	-0,1007	33,8039

W3=net.LW{3,2}

	1	2	3	4	5	6	7	8
N1	-0.1029	-0.2205	1.1001	0.2796	0.2537	-1.0455	-0.0863	1.0944

b1=net.b{1}

N1	17,6647
N2	-12,2797
N3	36,1202
N4	-11,3385
N5	-4,2961
N6	29,8648
N7	-12,2397
N8	5,5126
N9	16,6252
N10	-2,721
N11	-0,2757
N12	4,0912

b2=net.b{2}

N1	35,736
N2	-58,2132
N3	-28,9712
N4	29,135
N5	7,4498
N6	-119,529
N7	20,4235
N8	84,9714

b3=net.b{3}

N1	-0.5760

Tabla 3.4.2 Resultados obtenidos con la red entrenada

En la tabla 3.4.3, se muestran seis de los valores de la simulación de la red; los valores de a_1 corresponden a los valores entregados por al red y los valores de t_1 corresponden a los valores esperados.

a_1=						
1,341	1,3925	1,1588	1,1699	1,2124	1,339	1,4776

t_1=						
1,3439	1,389	1,157	1,17	1,2139	1,337	1,4799

e=						
0,0029	0,0035	0,0018	0,0001	0,0015	0,002	0,0023

emáx=	emin =
0,0035	1,19E-04

Tabla 3.4.3 Valores de la simulación

Los comandos, que evaluaron el desempeño de la red son los siguientes, con ellos se calculó el error en la simulación

```
e=abs(t1-a1)
emax=max(e)
emin=min(e)
```

En la tabla 3.4.3 puede observarse el trabajo final de la red y la excelente labor de aproximación que realiza. Ilustrar el resultado para todos los patrones de prueba resulta un poco extenso, por lo tanto se escogieron los patrones más representativos, donde puede verse que sin ningún problema la red ha

aproximado patrones que no se le habían presentado durante el entrenamiento, garantizando el éxito de la red para realizar predicciones de consumo de carga.

De los datos entregados en el entrenamiento, el error más significativo equivale a 0.0035, es decir 3.5kW, valor muy aceptable teniendo en cuenta la disparidad de los datos de entrada, ya que como se observa de la figura 3.4.1 la población tiene un comportamiento bastante aleatorio, pues aunque se conserva la forma de onda, indicando que el comportamiento es similar todos los días de la semana, los valores de la demanda pico son muy diferentes cada día

Después de probar los algoritmos *traingd*, *traingda*, (ver anexo A) se comprobó que el algoritmo *trainlm*, correspondiente al método de Levenberg Marquardt garantizaba una alta velocidad de aprendizaje y una excelente generalización de los patrones de entrenamiento que no se le habían presentado inicialmente, y por eso se escogió para completar el proceso de entrenamiento.

El proceso de entrenamiento, involucró experimentar con muchos tipos de redes tratando de encontrar no solo una configuración óptima, si no también un algoritmo que además de rapidez garantizara estabilidad en el resultado final. La red escogida en la figura 3.4.2 es bastante robusta e involucra un gran número de parámetros (149 en total), sin embargo alcanzó convergencia en menos tiempo que otras redes escogidas en las cuales se trabajó con una sola capa oculta compuesta por un gran número de neuronas, de esta forma se confirma que no existe un procedimiento establecido para determinar el modelo de red que debe

emplearse en cada aplicación y que sólo con la práctica puede determinarse cuál es la configuración de red que garantiza mejores resultados, ésta aparente dificultad de las redes neuronales puede aprovecharse para resaltar una de sus grandes ventajas: La adaptabilidad de esta teoría a diferentes lenguajes de programación, de tal forma que con unos pocos comandos los parámetros de la red pueden ser transformados sin mayor problema, adoptando una configuración totalmente diferente.

Redes con el tipo de estructura, como la utilizada en esta aplicación pueden emplearse para predecir la necesidad de una reconfiguración del sistema en un día y una hora específica. Es posible también que los resultados de una red como ésta, se utilicen para adoptar estrategias de restauración del servicio en diferentes horas y diferentes días, pues al conocer los valores de demanda es fácil determinar comportamientos críticos y dar prioridad en el restablecimiento, dejando por fuera a la menor cantidad de usuarios posible.

Las diferentes aplicaciones para las que sea necesario implementar una predicción de consumo de carga, pueden derivarse fácilmente de ésta adaptando el código fuente a los patrones con los que se desea realizar la predicción, así como se mencionó anteriormente, lo único necesario es variar la estructura del vector de entrada $\boldsymbol{p}$ adecuándolo a la cantidad de entradas que se vayan a ingresar (hora, día, mes, año, tipo de carga) a la red y también establecer con claridad que tipo de información de salida se espera dependiendo del propósito de la predicción.

3.5 CONTROL DE VOLTAJE POR INYECCIÓN DE REACTIVOS EN UNA BARRA REMOTA

3.5.1 Descripción del problema. El objetivo de un sistema de potencia es suministrar a las barras de carga, potencia activa y reactiva para su consumo conservando los niveles de voltaje dentro de los limites establecidos. Sin embargo los niveles de tensión en las barras están determinados por las condiciones del sistema, no obstante es posible realizar un control en el nivel de tensión inyectando reactivos en una barra remota, para de este modo controlar los niveles de tensión. Cuando se desea especificar el nivel de voltaje en una barra de carga PQ, ésta barra se declara como tipo PQV, la barra donde se inyecta potencia reactiva que generalmente es de tipo PV se declara como tipo P.

Las ecuaciones básicas que describen el flujo de carga en un sistema de potencia son las siguientes:

$$P_k = V_k \sum_{m \in k} V_m \left(G_{km} Cos\theta_{km} + B_{km} Sen\theta_{km} \right) \qquad (3.5.1)$$

$$Q_k = V_k \sum_{m \in k} V_m \left(G_{km} Sen\theta_{km} - B_{km} Cos\theta_{km} \right) \qquad (3.5.2)$$

Para $k = 1, \ldots, Nb$ $\qquad$ $Nb \Longrightarrow$ Número de barras del sistema

La solución de las ecuaciones de flujo de potencia generalmente se realizan por el método de Newton-Raphson, donde se calculan las derivadas de P y Q de las ecuaciones (3.5.1) y (3.5.2) respecto a las variables V y θ. El sistema de ecuaciones no lineales en forma matricial se describe en la ecuación 3.5.3, donde el Jacobiano del sistema es la matriz después de la igualdad.

$$\begin{bmatrix} \Delta P \\ \Delta Q \end{bmatrix} = \begin{bmatrix} \frac{\partial P}{\partial \theta} & \frac{\partial P}{\partial V} \\ \frac{\partial Q}{\partial \theta} & \frac{\partial Q}{\partial V} \end{bmatrix} \begin{bmatrix} \Delta \theta \\ \Delta V \end{bmatrix} \qquad (3.5.3)$$

Para la solución de flujo de potencia cada barra aporta distintas ecuaciones e incógnitas según su tipo, como se puede ver en las tablas 3.5.1 y 3.5.2

• Barra tipo PV Ecuación: $\Delta P_k = P_k^{esp} - P_k^{cal}$ Incógnita: θ_k	• Barra tipo PQ Ecuaciones: $\Delta P_k = P_k^{esp} - P_k^{cal}$ $\Delta Q_k = Q_k^{esp} - Q_k^{cal}$ Incógnitas: V_k , θ_k

Tabla 3.5.1 Ecuaciones e incógnitas barras, tipo PV y PQ

Cuando se desea realizar control de voltaje por inyección de reactivos en una barra remota, las barras tipo P y PQV proporcionan las siguientes ecuaciones:

• Barra tipo P o de Control	• Barra tipo PQV o Controlada
Ecuación: $\Delta P_k = P_k^{esp} - P_k^{cal}$	Ecuaciones: $\Delta P_k = P_k^{esp} - P_k^{cal}$
	$\Delta Q_k = Q_k^{esp} - Q_k^{cal}$
Incógnitas: V_k, θ_k	Incógnita: θ_k

Tabla 3.5.2 Ecuaciones e incógnitas barras tipo P y PQV

En resumen las ecuaciones e incógnitas que aporta cada tipo de barra se sintetiza en el grupo de ecuaciones (3.5.4).

$$\begin{matrix} NPQV \\ NPQ \\ NPV \\ NP \end{matrix}\left\{\begin{matrix} \\ \\ \\ \\ \end{matrix}\right.\left[\begin{matrix} \\ \Delta P \\ \\ \\ \end{matrix}\right] = \left[\begin{matrix} \dfrac{\partial P}{\partial \theta} & \dfrac{\partial P}{\partial V} \\ & \\ \dfrac{\partial Q}{\partial \theta} & \dfrac{\partial Q}{\partial V} \end{matrix}\right]\left[\begin{matrix} \\ \Delta \theta \\ \\ \Delta V \\ \end{matrix}\right]\left\{\begin{matrix} NPQV \\ NPQ \\ NPV \\ NP \\ NPQ \\ NP \end{matrix}\right. \qquad (3.5.4)$$

Donde: $NP = NPQV \Rightarrow$ Número de barras con control remoto de voltaje.

$NPQ, NPV \Rightarrow$ Número de barras tipo PQ y PV respectivamente

Esta aplicación se llevará a cabo en un sistema de tres barras donde la barra 1 es una barra slack o de compensación Vθ, la barra 2 es una barra tipo P, desde donde se pretende controlar por inyección de reactivos el voltaje de la barra 3, la cual es una barra tipo PQV.

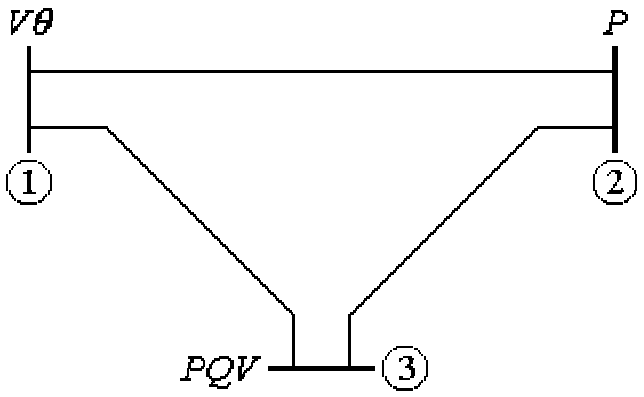

Figura 3.5.1 Sistema de 3 barras

El sistema tiene como potencia base Sb=100MVA y voltaje base Vb=230kV, el voltaje en el nodo slack se mantendrá en V1=1.015p.u.

Las ecuaciones no lineales que describen el comportamiento del flujo de potencia en el sistema de tres barras en forma matricial son las siguientes:

$$\begin{bmatrix} \Delta P_2 \\ \Delta P_3 \\ \Delta Q_3 \end{bmatrix} = \begin{bmatrix} \frac{\partial P_2}{\partial \theta_2} & \frac{\partial P_2}{\partial \theta_3} & \frac{\partial P_2}{\partial V_2} \\ \frac{\partial P_3}{\partial \theta_2} & \frac{\partial P_3}{\partial \theta_3} & \frac{\partial P_3}{\partial V_2} \\ \frac{\partial Q_3}{\partial \theta_2} & \frac{\partial Q_3}{\partial \theta_3} & \frac{\partial Q_3}{\partial V_2} \end{bmatrix} \begin{bmatrix} \Delta \theta_2 \\ \Delta \theta_3 \\ \Delta V_2 \end{bmatrix} \qquad (3.5.5)$$

$$\begin{bmatrix} \Delta P_2 \\ \Delta P_3 \\ \Delta Q_3 \end{bmatrix} = \begin{bmatrix} H_{22} & H_{32} & N_{22} \\ H_{32} & H_{33} & N_{32} \\ M_{32} & M_{33} & L_{22} \end{bmatrix} \begin{bmatrix} \Delta \theta_2 \\ \Delta \theta_3 \\ \Delta V_2 \end{bmatrix} \qquad (3.5.6)$$

$$H_{22} = -V_2 B_{22} - Q_2 \tag{3.5.7}$$

$$H_{23} = V_2 V_3 \left(G_{23} Sen\left(V_2 - V_3\right) - B_{23} Cos\left(V_2 - V_3\right)\right) \tag{3.5.8}$$

$$H_{32} = V_3 V_2 \left(G_{32} Sen\left(V_3 - V_2\right) - B_{32} Cos\left(V_3 - V_2\right)\right) \tag{3.5.9}$$

$$H_{33} = -V_3 B_{33} - Q_3 \tag{3.5.10}$$

$$N_{22} = \left(P_2 / V_2\right) + V_2 G_{22} \tag{3.5.11}$$

$$N_{32} = V_3 \left(G_{32} Cos\left(V_3 - V_2\right) + B_{32} Sen\left(V_3 - V_2\right)\right) \tag{3.5.12}$$

$$M_{32} = -V_3 V_2 \left(G_{32} Cos\left(V_3 - V_2\right) + B_{32} Sen\left(V_3 - V_2\right)\right) \tag{3.5.13}$$

$$M_{33} = -V_3^{\,2} G_{33} + P_3 \tag{3.5.14}$$

$$L_{32} = V_3 \left(G_{32} Sen\left(V_3 - V_2\right) - B_{32} Cos\left(V_3 - V_2\right)\right) \tag{3.5.15}$$

El proceso de cálculo se realiza de forma iterativa, cuando el sistema de ecuaciones ha convergido por debajo del error deseado se obtiene V_2, de las condiciones iniciales del flujo de carga se tienen V_1 y V_3 y con estos datos se calcula la potencia reactiva inyectada en la barra 2 (Q_2), que satisface las condiciones impuestas en la barra 3.

$$\begin{aligned} Q_2 = -V_2^2 B_{22} + V_2 V_1 \left(G_{21} Sen\left(V_2 - V_1\right) - B_{21} Cos\left(V_2 - V_1\right)\right) + \\ V_2 V_3 \left(G_{23} Sen\left(V_2 - V_3\right) - B_{23} Cos\left(V_2 - V_3\right)\right) \end{aligned} \tag{3.5.16}$$

Los datos de las líneas en p.u. son:

	Z serie	Y/2 paralelo
1-2	0.0229+j0.0400	j0.036
2-3	0.0118+j0.0333	j0.027
3-1	0.0046+j0.0267	j0.040

Tabla 3.5.3 Datos de las líneas del sistema de tres barras

Y bus en p.u.

	1	2	3
1	17.04606-j55.12630	-10.77946+j18.82875	-6.26660+j36.37354
2	-10.77946+j18.82875	22.37953-j41.94710	-12.60006+j23.18134
3	-6.26660+j36.37354	-12.60006+j23.18134	18.86667-j59.48788

Tabla 3.5.4 Ybus del sistema de tres barras

3.5.2 Justificación del tipo de red. El problema de flujos de potencia está regido por ecuaciones no lineales, las cuales según lo demostrado por Funahashi[16] pueden ser aprendidas por una red multicapa, en esta publicación el autor demuestra que siempre existe la red neuronal apropiada; este teorema es sólo de existencia pues prueba que la red existe pero no indica como construirla ni tampoco garantiza que la red aprenderá la función.

La red neuronal de propagación inversa o Backpropagation es una generalización para redes multicapa del algoritmo LMS. Ambas utilizan la técnica del error medio cuadrático, pero la red Backpropagation es más ampliamente utilizada para aproximación de funciones en especial no lineales.

3.5.3 Entrenamiento de la Red

3.5.3.1. Problema utilizando como entradas P3, Q3 y V3; para obtener Q2 utilizando 10 neuronas en la capa oculta

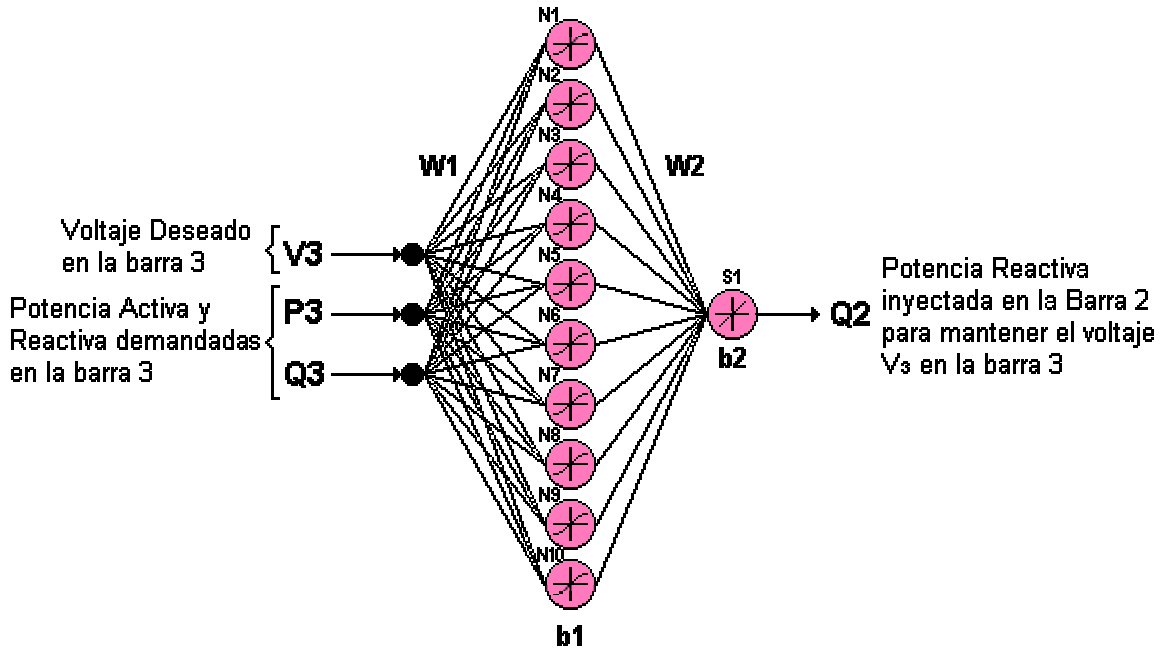

Figura 3.5.2 Red neuronal en configuración 3:10:1

La red será entrenada para determinar la cantidad de potencia reactiva en p.u. inyectada en la barra dos (Q_2) para satisfacer las condiciones impuestas por V_3, P_3 y Q_3, que son el voltaje, la potencia activa y reactiva demandada en la barra 3. Para generar el set de entrenamiento, se resolvieron las ecuaciones no lineales que describen el comportamiento del sistema por el método de Newton-Raphson considerando variaciones de V_3, P_3, Q_3; cada variable se incremento según lo mostrado en la tabla 3.5.3; se mantuvieron constantes el voltaje en el nodo slack V_1=1.015p.u. y la potencia activa demandada en la barra dos, P_2=0.7p.u.

	Valor Inicial	Valor Final	Incremento
V3	0.997	1.015	0.003
P3	0.700	0.850	0.030
Q3	0.250	0.400	0.030

Tabla 3.5.5 Patrones de entrenamiento

De estas variaciones se obtuvieron 252 combinaciones que conforman el vector de patrones de entrenamiento ***p***, que generan a su vez igual número de patrones objetivo o salidas deseadas ***t***.

P	P1	P2		P36		P78	P79		P140		P252
V3	0.9970	0.9970	....	0.9970	....	1.0030	1.0030	....	1.0060	....	1.0150
P3	0.8500	0.8500	....	0.7000	....	0.8500	0.8200	....	0.7000	....	0.7000
Q3	0.2500	0.2800	....	0.4000	....	0.4000	0.2500	....	0.2800	....	0.4000

t	t1	t2		t36		t78	t79		t140		t252
Q2	-0.7480	-0.6939	....	-0.5495	....	0.0685	-0.2244	....	0.0464	....	1.1281

Tabla 3.5.6 Patrones de entrenamiento de la red neuronal

Los pesos iniciales de la red fueron obtenidos de la función *initlay* que calcula los pesos iniciales con base al rango de los datos de entrada ***p***.

W1i=net.IW{1,1}

	I1	I2	I3
N1	-82.4449	-30.1178	24.7463
N2	251.4143	-20.6245	-16.7856
N3	-325.2046	-1.0347	9.6617
N4	325.4475	-9.5779	0.6391
N5	329.3532	-5.3503	-5.1664
N6	-142.9295	34.0779	12.7224
N7	64.3926	-27.4716	28.3361
N8	294.8647	-14.5486	-12.3960
N9	231.7537	21.5285	-19.5048
N10	-186.3720	-28.7647	-17.0220

b1i=net.b{1}

N1	101.2545
N2	-233.8294
N3	326.4934
N4	-321.1904
N5	-325.8389
N6	112.9068
N7	-51.6923
N8	-279.6544
N9	-241.1438
N10	212.2988

W2i=net.LW{2,1}

	N1	N2	N3	N4	N5	N6	N7	N8	N9	N10
S1	-0.1769	-0.4281	-0.2117	0.0060	0.4440	-0.3876	-0.7757	-0.1134	-0.0665	-0.9707

b2i=net.b{2}

S1	0.3281

Tabla 3.5.7 Pesos iniciales para la red neuronal

Se entrenará la red de la figura 3.5.2 con 3 distintos algoritmos de aprendizaje y luego se simulará cada red con los siguientes patrones de prueba:

	1	2	3	4	5	6	7	8
V3	1.0120	**0.9950**	1.0100	**1.0200**	1.0070	**1.0160**	1.0050	1.0100
P3	0.7320	0.8300	0.8250	0.8370	0.7520	**0.9120**	**0.8720**	0.8500
Q3	0.3710	0.3600	**0.2300**	**0.4100**	0.3230	0.2500	**0.4500**	**0.2020**
Q2	0.7992	-0.7359	0.3895	1.7141	0.2457	1.0476	0.3588	0.3497

	9	10	11	12	13	14	15	16
V3	1.0100	**1.0170**	1.0050	1.0120	1.0030	**0.9950**	**1.0200**	1.0100
P3	0.8430	**0.9000**	0.7500	0.8500	0.8200	**0.6500**	**0.9000**	0.8000
Q3	0.3570	**0.1500**	0.3100	**0.5000**	0.2800	**0.1500**	**0.2000**	0.3000
Q2	0.6390	0.9463	0.0349	1.1089	-0.1691	-1.1977	1.3368	0.5088

Tabla 3.5.8 Patrones de prueba de las RN's

Los valores de prueba que se encuentran en negrilla se encuentran por fuera del rango de entrenamiento y buscan probar la capacidad de extrapolación de la red, la mayoría de los datos de prueba que se encuentran dentro del rango de entrenamiento no coinciden exactamente con el valor utilizado en el entrenamiento para probar la capacidad de interpolación de la red.

- **Entrenamiento utilizando el algoritmo *trainrp***

El siguiente código entrena una red neuronal con 3 entradas, 10 neuronas en la capa oculta y 1 en la capa salida, utilizando la función de aprendizaje *trainrp* (Ver Anexo 1) y como objetivo un error medio cuadrático de $6x10^{-6}$.

```
net=newff([0.997 1.015;0.7 0.85;0.25 0.4],[10,1],
          {'tansig','purelin'},'trainrp');
net.trainParam.epochs=50000;
net.trainParam.goal=6e-6;
net.trainParam.min_grad=1e-10;
net.trainParam.lr=0.03;
net.trainParam.delt_inc=1.08;
net.trainParam.delt_dec=0.70;
net.trainParam.delta0=0.08;
net.trainParam.deltamax=50.0;
[net,tr]=train(net,P,t);
```

El error medio cuadrático objetivo fue fijado en $6x10^{-6}$, puesto que con la variación de los distintos parámetros no fue posible alcanzar un valor inferior; los mejores resultados se obtuvieron para una rata de aprendizaje de 0.03, para esta red un valor superior causa serias inestabilidades lo que a su vez ocasiona un proceso de aprendizaje bastante pobre (la red generaliza de manera incorrecta) y un valor inferior produce un aprendizaje bastante lento.

La figura 3.5.3 muestra la evolución del error medio cuadrático a través de 50000 iteraciones hasta alcanzar el error deseado.

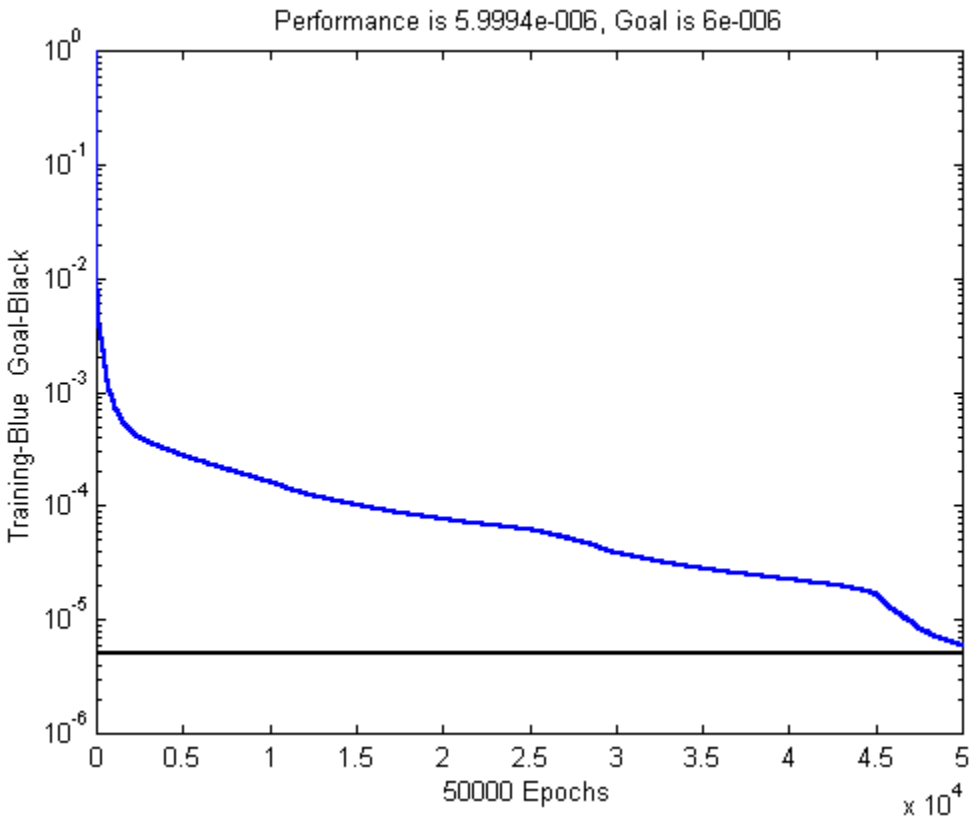

Figura 3.5.3 Error medio cuadrático utilizando *trainrp*

Luego de 50000 iteraciones el error medio cuadrático cayó por debajo de $6x10^{-6}$, los pesos y ganancias de la red entrenada son:

W1f=net.IW{1,1}

	I1	I2	I3
N1	-79.6128	-24.9124	34.8705
N2	252.3923	-15.6532	-27.2092
N3	-323.5910	-0.0197	0.3244
N4	322.9282	-0.2053	-0.1796
N5	325.8474	-0.0746	0.2680
N6	-133.9024	3.5832	15.9724
N7	59.2478	1.9485	7.0366
N8	275.5195	5.1249	21.3110
N9	234.3145	-18.7189	-245.3248
N10	-200.4902	-7.0055	-22.0806

b1f=net.b{1}

N1	112.1139
N2	-232.4849
N3	327.5410
N4	-324.6580
N5	-325.7568
N6	127.9533
N7	-63.8474
N8	-284.2129
N9	-129.0298
N10	212.8007

W2f=net.LW{2,1}

	N1	N2	N3	N4	N5	N6	N7	N8	N9	N10
S1	-0.0688	-0.0069	-0.2988	0.2533	0.2656	0.0251	0.2475	0.0563	-0.0031	-0.0346

b2f=net.b{2}

S1	0.2726

Tabla 3.5.9 Pesos finales de la RN

Para hallar la respuesta de la red (Q2Rn) ante los patrones de prueba, se simuló la red y para hallar el error en las aproximaciones se comparó con la respuesta producida por las ecuaciones del flujo de carga (Q2Fc).

	1	2	3	4	5	6	7	8
Q2Fc	0.7992	-0.7359	0.3895	1.7141	0.2457	1.0476	0.3588	0.3497
Q2Rn	0.7980	-0.6457	0.3799	1.3230	0.2549	1.0028	0.3215	0.3437
Error	0.0011	-0.0902	0.0097	0.3911	-0.0092	0.0448	0.0374	0.0060

	9	10	11	12	13	14	15	16
Q2Fc	0.6390	0.9463	0.0349	1.1089	-0.1691	-1.1977	1.3368	0.5088
Q2Rn	0.6333	0.8745	0.0268	0.9868	-0.1672	-0.9485	1.0116	0.4964
Error	0.0057	0.0718	0.0081	0.1220	-0.0019	-0.2492	0.3252	0.0123

Tabla 3.5.10 Simulación de la RN con los patrones de prueba

Los errores más altos se presentaron en los patrones de prueba 4 (2), 12 (1), 14 (3) y 15 (3), donde los valores entre paréntesis son el número de datos extrapolados del set de entrenamiento; algunos resultados son muy próximos a los valores que producirían las ecuaciones de flujo de carga que describen el comportamiento del sistema, pero en general el aprendizaje fue muy lento (50000 iteraciones) y la aproximación es muy deficiente.

- Entrenamiento utilizando el algoritmo *trainbfg*

El siguiente código entrena una red neuronal con 3 entradas, 10 neuronas en la capa oculta y 1 en la salida, utilizando la función de aprendizaje *trainbfg* (Ver Anexo 1) y como objetivo un error medio cuadrático de $1.35x10^{-8}$.

```
net=newff([0.997 1.015;0.7 0.85;0.25 0.4],[10,1],
          {'tansig','purelin'},'trainbfg');
net.trainParam.epochs=1800;
net.trainParam.goal=1.35e-8;
net.trainParam.min_grad=1e-10;
net.trainParam.searchFcn='srchcha'
net.trainParam.scal_tol=20;
net.trainParam.alpha=0.001;
net.trainParam.beta=0.1;
net.trainParam.delta=0.01;
net.trainParam.gama=0.1;
net.trainParam.low_lim=0.1;
net.trainParam.up_lim=0.5
[net,tr]=train(net,P,t);
```

El error medio cuadrático fue fijado en $1.35x10^{-8}$, por debajo de este valor la matriz Jacobiana del algoritmo se vuelve singular o sin inversa y por este motivo se presenta un estancamiento en el proceso de aprendizaje, pero antes de que esto suceda los resultados obtenidos presentan un error medio cuadrático bastante pequeño, produciendo una muy buena aproximación y generalización de las funciones deseadas

La figura 3.5.4 muestra la evolución del error medio cuadrático a través de 1523 iteraciones hasta alcanzar el error deseado.

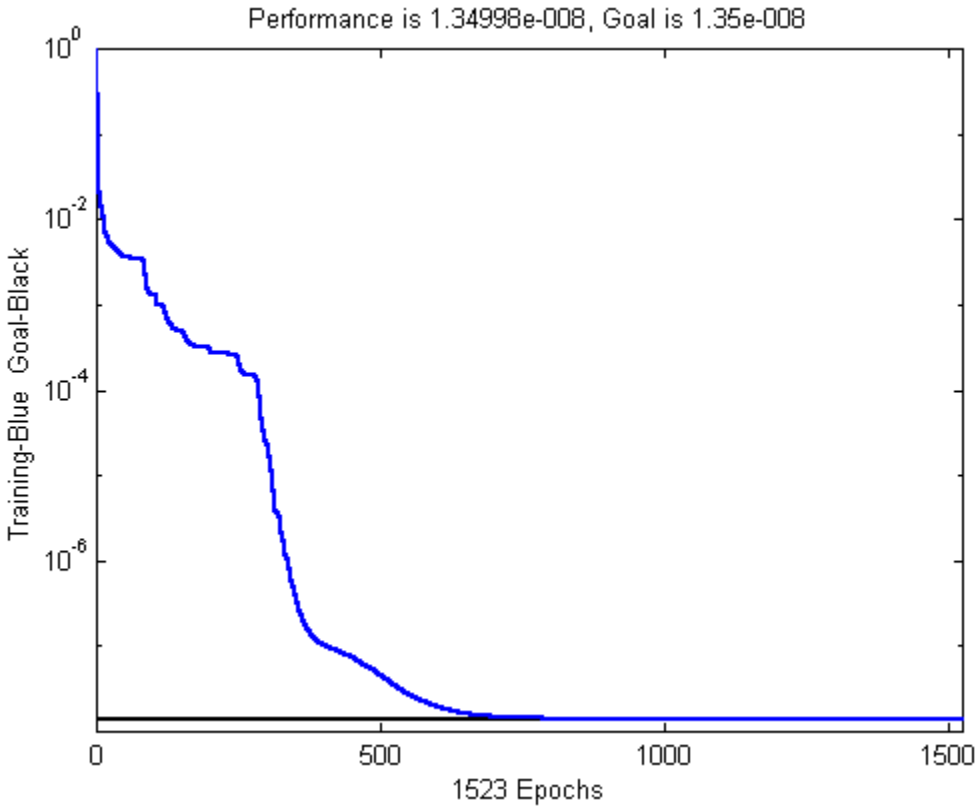

Figura 3.5.4 Error medio cuadrático utilizando *trainbfg*

Luego de 1523 iteraciones el error medio cuadrático cayó por debajo de 1.35×10^{-8}, alcanzado el error deseado, los pesos y ganancias de la red entrenada son:

W1f=net.IW{1,1}

	I1	I2	I3
N1	314.7873	292.0234	160.5730
N2	311.7056	-134.0357	138.9636
N3	-84.6597	-0.4099	-1.4174
N4	26.3003	0.1433	0.5297
N5	109.2728	0.6771	2.6136
N6	-63.4637	-296.9911	231.7073
N7	-75.4292	-98.1638	5.7991
N8	643.5915	286.1720	167.6307
N9	257.6940	33.3274	-21.1659
N10	-220.3097	125.5978	-512.3378

b1f=net.b{1}

N1	492.3094
N2	-159.4016
N3	87.8088
N4	-26.8251
N5	-109.7182
N6	208.1393
N7	-188.8066
N8	66.2581
N9	-215.0041
N10	171.3954

W2f=net.LW{2,1}

	N1	N2	N3	N4	N5	N6	N7	N8	N9	N10
S1	-35.0242	-5.1142	-0.3124	3.4025	0.0932	0.0000	29.3423	-617.2979	-545.3703	0.2596

b2f=net.b{2}

S1	1233.1

Tabla 3.5.11 Pesos finales de la RN entrenada con *trainbfg*

Para hallar la respuesta de la red (Q2Rn) ante los patrones de prueba, se simuló la red y para hallar el error en las aproximaciones se comparó con la respuesta producida por las ecuaciones del flujo de carga (Q2Fc).

	1	2	3	4	5	6	7	8
Q2Fc	0.7992	-0.7359	0.3895	1.7141	0.2457	1.0476	0.3588	0.3497
Q2Rn	0.7992	-0.7356	0.3893	1.7148	0.2459	1.0474	0.3582	0.3492
Error	-0.0000	-0.0002	0.0002	-0.0007	-0.0001	0.0001	0.0007	0.0005

	9	10	11	12	13	14	15	16
Q2Fc	0.6390	0.9463	0.0349	1.1089	-0.1691	-1.1977	1.3368	0.5088
Q2Rn	0.6390	0.9454	0.0351	1.1073	-0.1690	-1.1897	1.3380	0.5089
Error	-0.0000	0.0008	-0.0001	0.0015	-0.0001	-0.0080	-0.0011	-0.0001

Tabla 3.5.12 Simulación de la RN con los patrones de prueba

Como se observa esta red tiene una amplia capacidad de generalización tanto para interpolar como para extrapolar, el error más grande se produjo en el patrón de prueba 15 el cual tiene 3 valores extrapolados con respecto al set de entrenamiento.

- Entrenamiento utilizando el algoritmo *trainlm*

El siguiente código entrena una red neuronal con 3 entradas, 10 neuronas en la capa oculta y 1 en la salida, utilizando la función de aprendizaje *trainlm* y como objetivo un error medio cuadrático de $1x10^{-8}$.

```
net=newff([0.997 1.015;0.7 0.85;0.25 0.4],[10,1],
          {'tansig','purelin'},'trainlm');
net.trainParam.epochs=700;
net.trainParam.goal=1e-8;
net.trainParam.lr=0.03;
net.trainParam.mem_reduc=1;
net.trainParam.min_grad=1e-10;
[net,tr]=train(net,P,t);
```

Se utilizó una rata de aprendizaje de 0.03, para este valor se obtuvieron los mejores resultados de error medio cuadrático en el menor número de iteraciones, para valores diferentes a esta rata de aprendizaje se obtuvieron valores superiores en el error medio cuadrático para el mismo número de iteraciones.

No fue necesario fraccionar por submatrices el cálculo de la matriz Jacobiana del algoritmo de entrenamiento, lo cual en algunos casos se hace indispensable por razones computacionales cuando el conjunto de patrones de entrenamiento es muy extenso, pues el Jacobiano de este algoritmo tiene dimensiones QxN donde Q es el número de patrones de entrenamiento y N es el número de pesos y ganancias de la red.

La figura 3.5.5 muestra la evolución del error medio cuadrático a través de 601 iteraciones hasta alcanzar el error deseado.

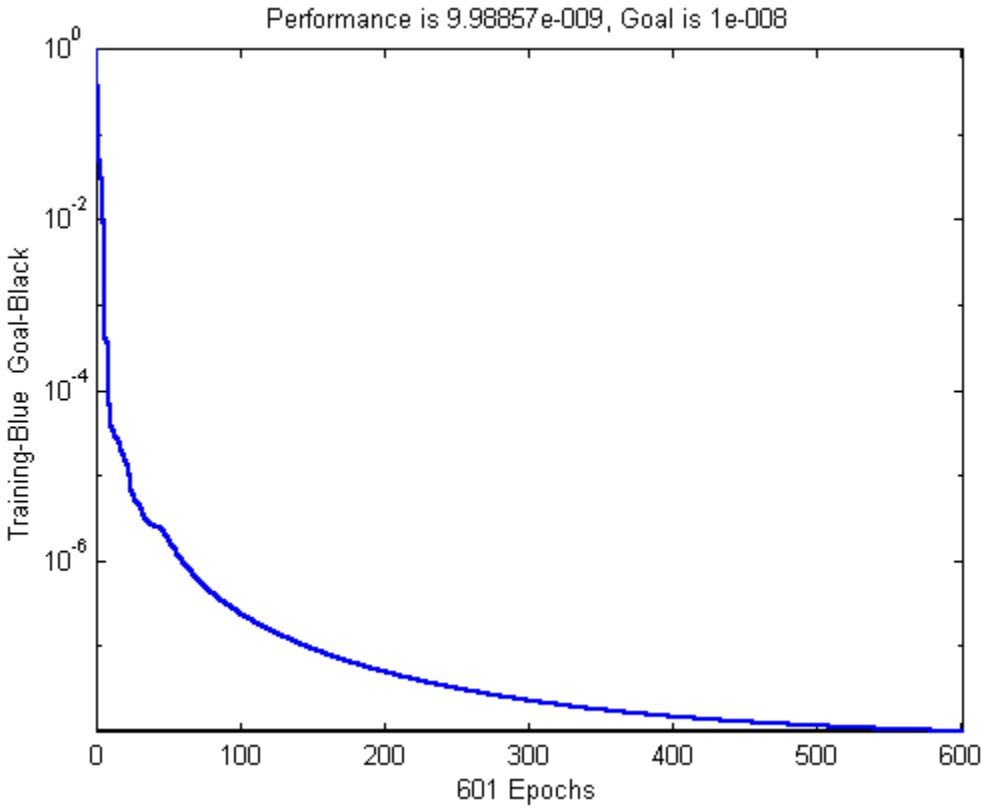

Figura 3.5.5 Error medio cuadrático utilizando *trainlm*

Luego de 601 iteraciones el error medio cuadrático cayó por debajo de $1x10^{-8}$, alcanzado el error deseado, los pesos y ganancias de la red entrenada son:

W1f=net.IW{1,1}

	I1	I2	I3
N1	-88.3684	-9.3759	-10.4875
N2	249.1384	-17.7323	-11.8856
N3	-289.7534	-1.2117	-8.7486
N4	333.5132	-34.3139	1.4535
N5	320.8474	1.9620	9.7026
N6	-143.5382	31.1396	23.1280
N7	13.7085	0.0753	0.2782
N8	298.9814	-14.0175	-39.8990
N9	241.2621	-9.1133	-10.0929
N10	-193.7428	0.4029	2.1040

b1f=net.b{1}

N1	105.0244
N2	-236.6178
N3	296.1395
N4	-312.9180
N5	-324.2801
N6	113.2598
N7	-14.1171
N8	-274.3762
N9	-232.4621
N10	195.5529

W2f=net.LW{2,1}

	N1	N2	N3	N4	N5	N6	N7	N8	N9	N10
S1	-0.3137	-0.0002	0.0016	0.0002	0.0043	0.0001	6.9859	0.0001	0.0007	-0.0068

b2f=net.b{2}

S1	1.7103

Tabla 3.5.13 Pesos y ganancias RN entrenada con *trainlm*

Para hallar la respuesta de la red (Q2Rn) ante los patrones de prueba, se simuló la red y para hallar el error en las aproximaciones se comparó con la respuesta producida por las ecuaciones del flujo de carga (Q2Fc)

	1	2	3	4	5	6	7	8
Q2Fc	0.7992	-0.7359	0.3895	1.7141	0.2457	1.0476	0.3588	0.3497
Q2Rn	0.7992	-0.7361	0.3896	1.7055	0.2459	1.0469	0.3598	0.3497
Error	-0.0001	0.0002	-0.0001	0.0086	-0.0001	0.0007	-0.0010	0.0000

	9	10	11	12	13	14	15	16
Q2Fc	0.6390	0.9463	0.0349	1.1089	-0.1691	-1.1977	1.3368	0.5088
Q2Rn	0.6387	0.9443	0.0350	1.1132	-0.1692	-1.1856	1.3291	0.5087
Error	0.0003	0.0020	-0.0000	-0.0043	0.0001	-0.0120	0.0078	0.0000

Tabla 3.5.14 Simulación de la RN con los patrones de prueba

Como se observa, esta red tiene una amplia capacidad de generalización para interpolar como para extrapolar, los errores más grandes se produjeron en los patrones de prueba 4 y 15 los cuales tienen 2 y 3 valores extrapolados respectivamente con relación al set de entrenamiento; este algoritmo produjo resultados tan buenos como los obtenidos con el algoritmo *trainbfg* pero en menor número de iteraciones y sin problemas de singularidad de la matriz Jacobiana en el proceso de aprendizaje.

3.5.3.2 Problema utilizando como entradas P3, Q3, V3 y P2; para obtener Q2 utilizando 12 neuronas en la capa oculta

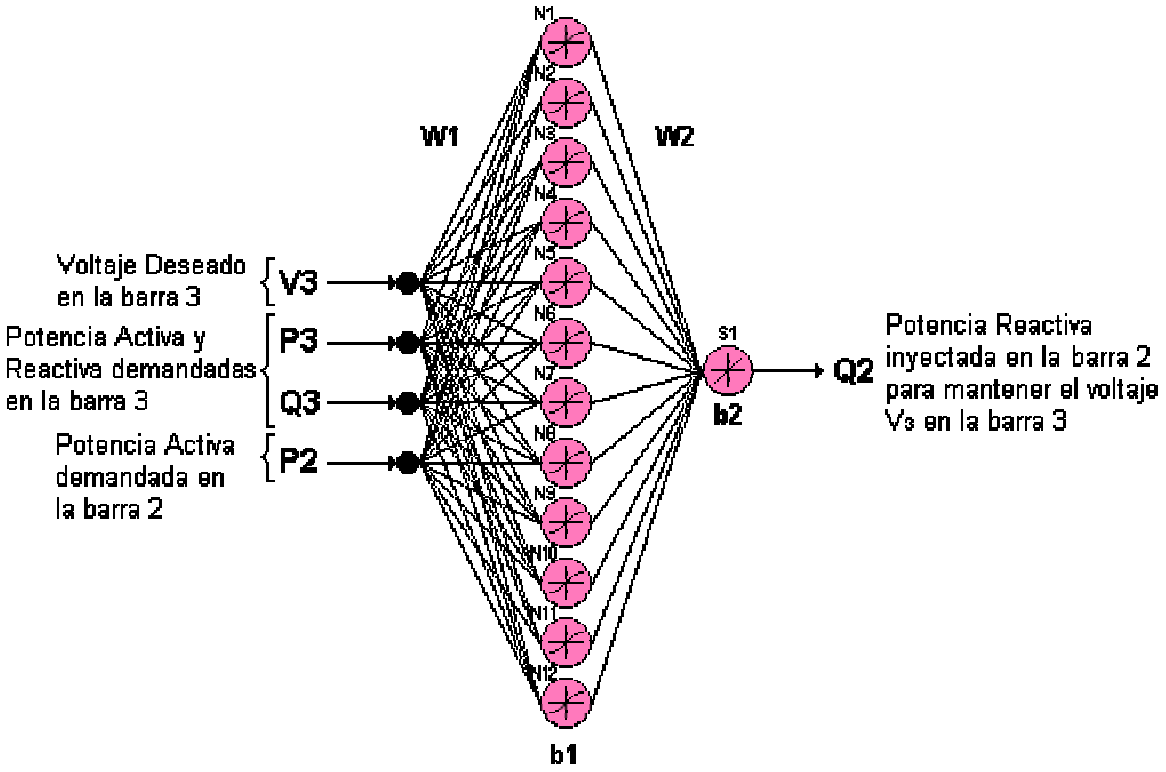

Figura 3.5.6 Red Neuronal en configuración 4:12:1

La red será entrenada para determinar la cantidad de potencia reactiva en p.u. inyectada en la barra 2 (Q_2) para satisfacer las condiciones impuestas por V_3, P_3 Q_3 y P_2, que son el voltaje, la potencia activa y reactiva deseados en la barra 3 y la potencia activa demandada en la barra 2. Para generar el set de entrenamiento se resolvieron las ecuaciones no lineales que describen el comportamiento del sistema por el método de Newton-Raphson considerando variaciones de V_3, P_3, Q_3 , P_2 y sus posibles combinaciones en los rangos mostrados en la tabla 3.5.13, el voltaje en el nodo slack V_1=1.015p.u se conservo constante.

	Valor Inicial	Valor Final	Incremento
V3	0.997	1.015	0.0045
P3	0.700	0.850	0.0500
Q3	0.250	0.400	0.0500
P2	0.600	0.750	0.0500

Tabla 3.5.15 Patrones de entrenamiento

De estas variaciones se obtuvieron 320 patrones de entrenamiento ***p***, que generaron a su vez igual número de respuestas esperadas t

P	p1	p2		p75		p148	p149		p263		p320
V3	0.9970	0.9970	....	1.0015	....	1.0060	1.0060	....	1.0150	....	1.0150
P3	0.8500	0.8500	....	0.8500	....	0.8000	0.8000	....	0.8500	....	0.7000
Q3	0.2500	0.2500		0.3500		0.2500	0.3000		0.3000		0.4000
P2	0.6000	0.6500	....	0.7000	....	0.7500	0.6000	....	0.7000	....	0.7500

t	t1	t2		t75		t148	t149		t263		t320
Q2	-0.7800	-0.7641	....	-0.1617	....	0.0577	0.1008	....	1.0139	....	1.1456

Tabla 3.5.16 Patrones de entrenamiento de la red neuronal

Los pesos iniciales de la red fueron obtenidos de la función *initlay* que calcula los pesos iniciales con base en el rango de los datos de entrada

W1i=net.IW{1,1}

	I1	I2	I3	I4
N1	214.9557	18.4104	8.3064	11.5648
N2	85.1487	-25.6540	-20.9690	2.1943
N3	-84.5134	12.2602	12.9213	28.0521
N4	-203.3228	18.7339	16.1259	-0.8630
N5	77.5282	-22.7461	-7.6400	-23.3397
N6	130.9888	27.2670	-5.8177	13.5142
N7	-68.2392	4.5660	10.2405	31.8477
N8	-207.9645	6.7104	17.1889	15.6131
N9	-63.1775	-25.1240	-12.1456	19.2571
N10	272.6824	6.2926	-9.6419	-1.9396
N11	129.7327	22.8632	5.0148	-20.4160
N12	250.8019	-11.8106	4.7112	11.8144

b1f=net.b{1}

N1	-243.6249
N2	-62.5758
N3	54.0424
N4	186.5500
N5	-42.8385
N6	-160.3749
N7	40.0478
N8	187.1758
N9	72.7921
N10	-273.0942
N11	-133.9472
N12	-250.0536

W2i=net.LW{2,1}

	N1	N2	N3	N4	N5	N6
S1	-0.2693	-0.7199	0.1335	0.6460	0.3479	0.9989

	N7	N8	N9	N10	N11	N12
S1	0.9233	-0.8823	-0.2794	0.0970	-0.4765	0.1947

b2f=net.b{2}

S1	-0.9014

Tabla 3.5.17 Pesos iniciales para la red neuronal

Se entrenará la red de la figura 3.5.6 con dos distintos algoritmos de aprendizaje y luego se simulará cada red con los siguientes patrones de prueba:

	1	2	3	4	5	6	7	8
V3	1.0120	**0.9950**	1.0100	**1.0200**	1.0070	**1.0160**	1.0050	1.0100
P3	0.7320	0.8300	0.8250	0.8370	0.7520	**0.9120**	**0.8720**	0.8500
Q3	0.3710	0.3600	**0.2300**	**0.4100**	0.3230	0.2500	**0.4500**	**0.2020**
P2	**0.5300**	**0.5500**	0.6200	0.6100	**0.7580**	0.7420	0.6320	**0.5840**
Q2	0.7416	-0.7836	0.3626	1.6821	0.2653	1.0623	0.3359	0.3108

	9	10	11	12	13	14	15	16
V3	1.0100	**1.0170**	1.0050	1.0120	1.0030	**0.9950**	**1.0200**	1.0100
P3	0.8430	**0.9000**	0.7500	0.8500	0.8200	**0.6500**	**0.9000**	0.8000
Q3	0.3570	**0.1500**	0.3100	**0.5000**	0.2800	**0.1500**	**0.2000**	0.3000
P2	0.6500	**0.7800**	0.6200	**0.5800**	**0.5500**	0.6000	0.6800	0.7000
Q2	0.6220	0.9743	0.0085	1.0673	-0.2180	-1.2288	1.3298	0.5088

Tabla 3.5.18 Patrones de prueba de las RN's

Los valores de la tabla 3.5.18 que se encuentran en negrilla se encuentran por fuera del rango de entrenamiento y con ellos se busca probar la capacidad de extrapolación de la red, la mayoría de los datos de prueba que se encuentran dentro del rango de entrenamiento no coinciden exactamente con el valor utilizado en el entrenamiento para probar la capacidad de interpolación de la red.

- Entrenamiento utilizando el algoritmo *trainbfg*

El siguiente código entrena una red neuronal con 4 entradas, 12 neuronas en la capa oculta y 1 en la salida, utilizando la función de aprendizaje *trainbfg* (Ver Anexo 1) y como objetivo un error medio cuadrático de $2.15x10^{-8}$.

```
net=newff([0.997 1.015;0.7 0.85;0.25 0.4;0.6 0.75],[12,1],
          {'tansig','purelin'},'trainbfg');
net.trainParam.epochs=1500;
net.trainParam.goal=5e-8;
net.trainParam.min_grad=1e-10;
net.trainParam.searchFcn='srchcha'
net.trainParam.scal_tol=20;
net.trainParam.alpha=0.001;
net.trainParam.beta=0.1;
net.trainParam.delta=0.01;
net.trainParam.gama=0.1;
net.trainParam.low_lim=0.1;
net.trainParam.up_lim=0.5;
[net,tr]=train(net,P,t);
```

En el entrenamiento de la red con este algoritmo se presentaron de nuevo inconvenientes en la invertibilidad de la matriz Jacobiana del algoritmo en el proceso iterativo, por lo cual el error medio cuadrático objetivo fue fijado en $5x10^{-8}$, por debajo de este valor la matriz Jacobiana del algoritmo se vuelve singular y presenta los inconvenientes antes descritos, pero antes de que esto suceda los resultados obtenidos proveen una muy buena aproximación.

La figura 3.5.7 muestra la evolución del error medio cuadrático a través de 1066 iteraciones hasta alcanzar el error deseado.

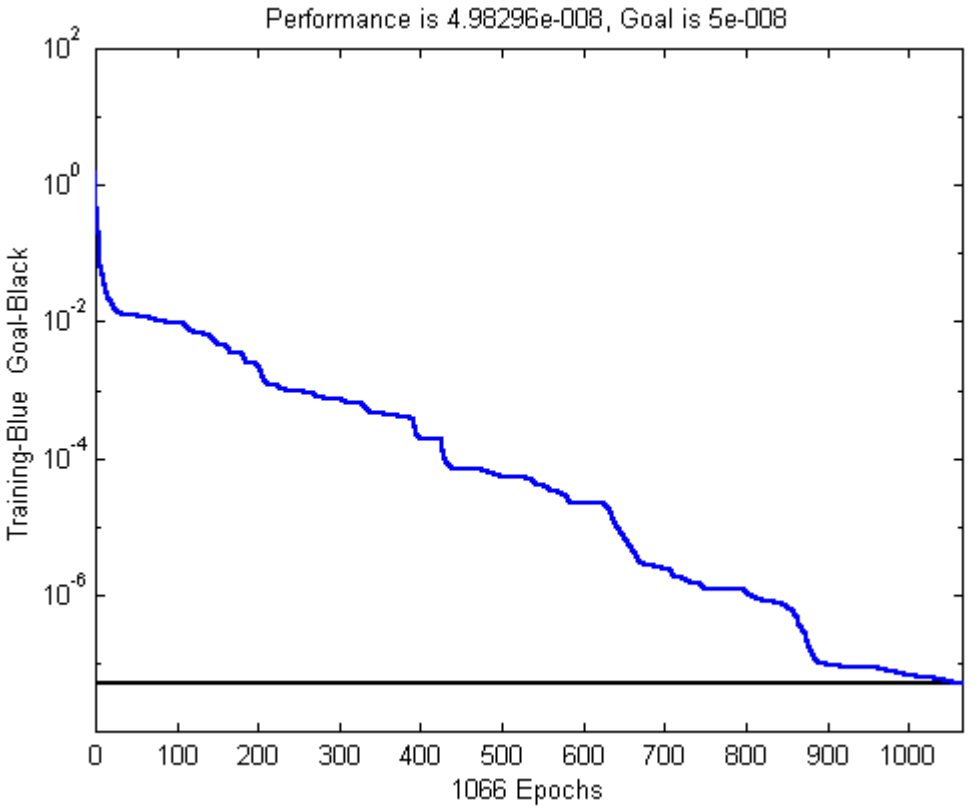

Figura 3.5.7 Error medio cuadrático utilizando *trainbfg*

Luego de 1066 iteraciones el error medio cuadrático cayó por debajo de $5x10^{-8}$, alcanzado el error deseado, los pesos y ganancias de la red entrenada son:

W1f=net.IW{1,1}

	I1	I2	I3	I4
N1	224.4737	1.1518	4.1945	0.7659
N2	102.8281	19.5357	-69.0280	-18.4298
N3	132.9784	173.1409	67.2342	171.2176
N4	-783.1072	-316.8951	-154.8182	-441.7653
N5	-151.4010	-190.8937	-109.9698	-159.8299
N6	425.0669	259.6459	85.3641	183.3998
N7	361.8879	328.3253	140.2923	276.1773
N8	47.0275	204.3169	40.6780	175.2289
N9	-45.6292	-0.2511	-0.9316	-0.1655
N10	268.0715	1.2620	4.5158	0.8389
N11	82.6771	4.2783	13.3543	-42.3772
N12	243.0169	1.3006	4.7068	0.8643

b1f=net.b{1}

N1	-231.1824
N2	-48.5460
N3	272.5634
N4	-385.5031
N5	-271.9678
N6	132.7761
N7	464.9202
N8	445.8823
N9	46.1722
N10	-272.7277
N11	-177.9131
N12	-248.8931

W2f=net.LW{2,1}

	N1	N2	N3	N4	N5	N6
S1	0.2130	10.4271	10.5727	-2.6097	27.1693	-906.5415

	N7	N8	N9	N10	N11	N12
S1	794.0887	-14.8412	-1.9511	0.0293	-0.4926	0.0775

b2f=net.b{2}

S1	130.1609

Tabla 3.5.19 Pesos y ganancias RN entrenada con *trainbfg*

Para hallar la respuesta de la red (Q2Rn) ante los patrones de prueba, se simuló la red y para hallar el error en las aproximaciones se comparó con la respuesta producida por las ecuaciones del flujo de carga (Q2Fc)

	1	2	3	4	5	6	7	8
Q2Fc	0.7416	-0.7836	0.3626	1.6821	0.2653	1.0623	0.3359	0.3108
Q2Rn	0.7412	-0.7841	0.3629	1.5963	0.2658	1.0627	0.3350	0.3112
Error	0.0004	0.0005	-0.0003	0.0858	-0.0005	-0.0004	0.0010	-0.0003

	9	10	11	12	13	14	15	16
Q2Fc	0.6220	0.9743	0.0085	1.0673	-0.2180	-1.2288	1.3298	0.5088
Q2Rn	0.6221	0.9742	0.0083	1.0634	-0.2184	-1.2087	1.3310	0.5088
Error	-0.0002	0.0001	0.0003	0.0040	0.0004	-0.0201	-0.0012	-0.0001

Tabla 3.5.20 Simulación de la RN con los patrones de prueba.

Como se observa, esta red tiene una amplia capacidad de generalización para interpolar como para extrapolar, los errores más grandes se produjeron en los patrones de prueba 4 y 14 los cuales tienen 2 y 3 valores extrapolados respectivamente en relación al set de entrenamiento.

- Entrenamiento utilizando el algoritmo *trainlm*

El siguiente código entrena una red neuronal con 4 entradas, 12 neuronas en la capa oculta y 1 en la salida, utilizando la función de aprendizaje *trainbfg* y como objetivo un error medio cuadrático de $1x10^{-8}$.

```
net=newff([0.997 1.015;0.7 0.85;0.25 0.4;0.6 0.75],[12,1],
          {'tansig','purelin'},'trainlm');
net.trainParam.epochs=700;
net.trainParam.goal=1e-8;
net.trainParam.lr=0.03;
net.trainParam.max_fail=5;
net.trainParam.mem_reduc=1;
net.trainParam.min_grad=1e-10;
[net,tr]=train(net,P,t);
```

Después de realizar varias pruebas, se decidió utilizar una rata de aprendizaje de 0.03, para este valor se obtuvieron los mejores resultados en el entrenamiento de la red, llegando a un valor muy pequeño en el error medio cuadrático a través de pocas iteraciones.

Como en el caso anterior de entrenamiento con este algoritmo, no fue necesario fraccionar por submatrices el cálculo de la matriz Jacobiana, pues los patrones de entrenamiento y el número de parámetros de la red son estrictamente los necesarios

La figura 3.5.8 muestra la evolución del error medio cuadrático a través de 362 iteraciones hasta alcanzar el error deseado.

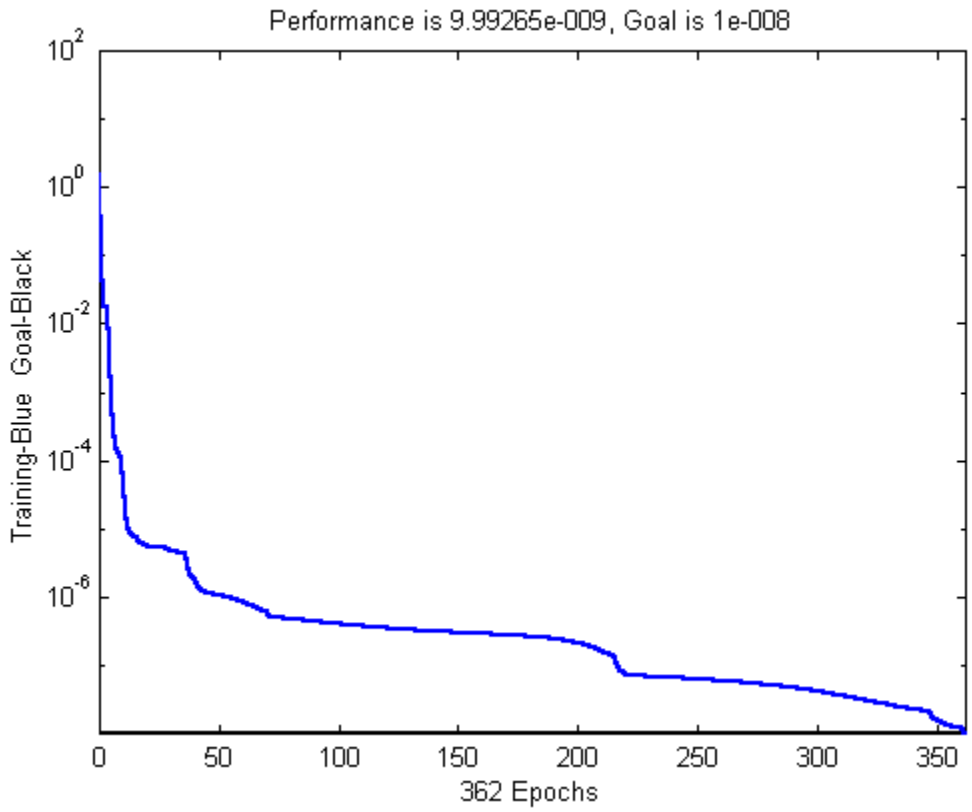

Figura 3.5.8 Error medio cuadrático utilizando *trainlm*

Luego de 362 iteraciones el error medio cuadrático cayó por debajo de $1x10^{-8}$, alcanzado el error deseado, los pesos y ganancias de la red entrenada son:

W1f=net.IW{1,1}

	I1	I2	I3	I4
N1	217.3050	1.1784	4.3958	0.7495
N2	86.3682	-28.0087	-14.8760	2.4348
N3	-82.5289	9.5925	36.1293	22.6072
N4	-196.4316	15.3102	-1.1168	-1.8863
N5	73.2020	-19.7579	-4.5863	-25.2722
N6	144.0374	0.7851	2.9423	0.5168
N7	-66.5832	7.3605	5.8988	24.5765
N8	-191.3809	-1.0280	-3.8772	-0.6760
N9	-70.7928	-0.3863	-1.4178	-0.2546
N10	263.8718	9.4256	8.6166	1.3268
N11	126.0300	11.5735	16.9349	2.5421
N12	246.1641	-2.9151	-7.5082	-1.6426

b1f=net.b{1}

N1	-221.8890
N2	-60.9291
N3	50.6335
N4	187.3245
N5	-49.0199
N6	-145.1983
N7	41.1724
N8	194.3911
N9	72.9238
N10	-275.1674
N11	-139.0503
N12	-244.2148

W2f=net.LW{2,1}

	N1	N2	N3	N4	N5	N6
S1	0.0595	-0.0001	0.0001	-0.0001	-0.4521	0.4525

	N7	N8	N9	N10	N11	N12
S1	0.0003	-0.1357	-1.3382	0.0004	0.0024	0.0023

b2f=net.b{2}

S1	0.1047

Tabla 3.5.21 Pesos y ganancias RN entrenada con *trainlm*

Para hallar la respuesta de la red (Q2Rn) ante los patrones de prueba, se simuló la red y para hallar el error en las aproximaciones se comparó con la respuesta producida por las ecuaciones del flujo de carga (Q2Fc).

	1	2	3	4	5	6	7	8
Q2Fc	0.7416	-0.7836	0.3626	1.6821	0.2653	1.0623	0.3359	0.3108
Q2Rn	0.7406	-0.7842	0.3624	1.6512	0.2652	1.0621	0.3357	0.3105
Error	0.0010	0.0007	0.0002	0.0308	0.0001	0.0002	0.0002	0.0004

	9	10	11	12	13	14	15	16
Q2Fc	0.6220	0.9743	0.0085	1.0673	-0.2180	-1.2288	1.3298	0.5088
Q2Rn	0.6222	0.9730	0.0089	1.0680	-0.2182	-1.1664	1.3250	0.5089
Error	-0.0002	0.0013	-0.0004	-0.0006	0.0002	-0.0624	0.0048	-0.0002

Tabla 3.5.22 Simulación de la RN con los patrones de prueba

De la tabla 3.5.22 se observa que esta red tiene una amplia capacidad de generalización para interpolar como para extrapolar, los errores más grandes se produjeron en los patrones de prueba 4 (2), 14 (3) y 15 (3) donde los valores entre paréntesis son el número de datos extrapolados en relación con el set de entrenamiento; este algoritmo produjo resultados tan buenos como los obtenidos

con el algoritmo *trainbfg* pero en menor número de iteraciones y sin problemas de singularidad de la matriz Jacobiana en el proceso de aprendizaje.

Para entrenar una red neuronal se debe invertir cierta cantidad de tiempo en la elección del tipo de red, conjunto de patrones de entrenamiento, topología de la red, proceso de entrenamiento; la inversión de esta cantidad de tiempo se ve recompensada cuando la red ha sido entrenada, pues la velocidad de la respuesta de la red es muy grande y la confiabilidad en las respuestas es muy alta.

Los sistemas de potencia poseen diferentes tópicos en los cuales se dificulta la solución de las complejas ecuaciones que los describen, una red neuronal entrenada correctamente con abundante y representativa información puede controlar las condiciones de operación de un sistema de potencia, con la precisión y acierto de un operador experto.

Utilizando redes neuronales se podrían entrenar dispositivos inteligentes para lograr despacho económico en mercado abierto y obtener flujos óptimos en el sistema de potencia sin necesidad de recurrir a resolver el sistema de ecuaciones no lineales que describen este de difícil solución.

3.6 RECONFIGURACIÓN DE UN ALIMENTADOR PRIMARIO

3.6.1 Descripción del problema.

En la figura 3.6.1 vemos el sistema de distribución que será analizado en este ejemplo, el sistema cuenta con tres alimentadores, 14 barras, 13 ramas del árbol y 3 ramas de enlace.

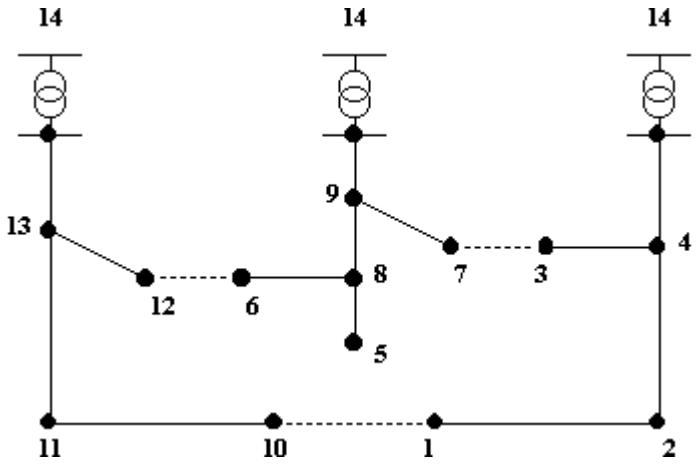

Figura 3.6.1 Sistema de 14 nodos

Cada uno de los nodos representa la carga concentrada como una inyección de potencia, o la división de diferentes ramas y/o ubicación de un seccionador.

Las ramas o líneas continuas interconectan los nodos, transportando la energía demandada y conformando así el árbol; para completar el sistema las líneas punteadas representan líneas desenergizadas formando el coárbol.

Cuando una rama del árbol se intercambia con una rama del coárbol, conservando la estructura radial, la topología del sistema cambia, y a esta maniobra se le llama reconfiguración de la red de distribución.

Bajo condiciones de operación normal la reconfiguración es útil para balancear cargas, evitando sobrecorrientes en las líneas o sobrecargas en los transformadores, mejora los niveles de tensión y reduciendo las pérdidas de potencia activa del sistema; ante aumentos de demanda, la reconfiguración es una herramienta útil en la planeación y diseño de nuevas configuraciones óptimas.

En este ejemplo el objetivo principal es utilizar la reconfiguración como una herramienta para reducción de pérdidas, para ello es necesario correr un flujo de carga antes y después de la maniobra, lo que nos permitirá conocer los valores de las pérdidas de potencia y comprobar así la eficiencia de la operación.

Las pérdidas de potencia activa totales del sistema pueden calcularse como sigue:

$$Pa = \sum_{i=1}^{n} r_i \left[\frac{P_i^2 + Q_i^2}{V_i^2} \right]$$

Donde:

Pa: Pérdidas activas totales del sistema

Pi: Flujo de potencia activa por la rama i

Qi: Flujo de potencia reactiva por la rama i

Vi: Magnitud del voltaje de recibo por la rama i

n :Número de ramas del sistema

El objetivo es encontrar una configuración óptima del sistema de distribución, es decir encontrar todas aquellas posibles configuraciones que con base en la figura 3.6.1 minimicen las pérdidas; nuestra función objetivo (F.O) es entonces:

$$F.O = min\left(\sum_{i=1}^{n} r_i \left[\frac{P_i^2 + Q_i^2}{V_i^2}\right]\right)$$

La reconfiguración óptima debe cumplir con las siguientes restricciones:

- Conservar la topología radial de la red sin que se presenten aislamientos del sistema

$$S_{i-1} = S_i + S_{Li} \quad (i=2,3,4...n)$$

Donde:

S_{i-1}: Flujo de potencia que sale del nodo i -1

S_i : Flujo de potencia que sale del nodo i

S_{Li}: Potencia de carga en el nodo i-ésimo

n : Número de nodos del sistema

- No sobrepasar las restricciones de voltaje mínimo en la carga

$$V_i \geq V_i^{min} \ (i=1,2,3...,n)$$

Donde:

V_i : Voltaje en el nodo i

V_i^{min} : Voltaje mínimo permitido para el nodo i

n : Número de nodos del sistema

- Cumplir con las especificaciones de carga máxima en los transformadores y de corriente máxima que circule por cada rama

$$S_i \leq S_i^{max} \quad (i= 1,2,3,...,m)$$

Donde:

S_i : Flujo de carga por la rama i, o potencia demandada para el transformador i

S_i^{max} : Capacidad máxima de la rama i, o capacidad máxima del transformador i

n : Número de ramas del alimentador o número de transformadores

En el proceso de encontrar la configuración óptima, pueden diferenciarse dos técnicas, aquellas exhaustivas que estudian todas las combinaciones de estados

del sistema, y otras que proporcionen una ruta heurística que se acerque de manera rápida al estado objetivo.

Una heurística es una técnica que aumenta la eficiencia de un proceso de búsqueda, en ocasiones pueden causar que una ruta sea pasada por alto, pero en promedio, mejoran la calidad de las rutas que exploran. Al usar buenas heurísticas se esperan soluciones a problemas difíciles, que aunque no siempre son las óptimas, podría decirse que son subóptimas.

Un gran número de las metodologías que se han utilizado para resolver el problema de la reconfiguración de alimentadores primarios, se ha basado en técnicas heurísticas; de estos trabajos se destacan: Cinvalar [7], Baran M.E y Wu F.F [4], Goswami K.S y Basu S.K [20], Chen C.S y Cho M [6] y Gallego R.[18], metodología que se empleará en esta aplicación.

En la figura 3.6.1, las líneas punteadas representan los enlaces del sistema para la configuración inicial. Los datos de la estructura del sistema pueden verse en la tabla 3.6.1, cuyos valores base son: Vb=23Kv y Sb=100MVA,

Nodo de	Nodo de	Resistencia	Reactancia	Demanda nodo de recibo	
Envío	Recibo	de rama (pu)	De rama(pu)	P(MW)	Q(MVAR)
14	13	0.0075	0.1	2	1.6
13	12	0.08	0.11	3	1.5
13	11	0.09	0.18	2	0.8
11	10	0.04	0.04	1.5	1.2
14	9	0.11	0.11	4	2.7
9	8	0.08	0.11	5	3
9	7	0.11	0.11	1	0.9
8	6	0.11	0.11	0.6	0.1
8	5	0.08	0.11	4.5	2
14	4	0.11	0.11	1	0.9
4	3	0.09	0.12	1	0.7
4	2	0.08	0.11	1	0.9
2	1	0.04	0.04	2.1	1
12	6	0.04	0.04		
7	3	0.04	0.04		
10	1	0.09	0.12		

Tabla 3.6.1 Estructura del sistema de catorce nodos

Como no existen curvas de carga que simulen el comportamiento de este sistema, se supondrá un agrupamiento según perfiles de demanda de igual tendencia, de acuerdo con el procedimiento empleado por Baran M.E [4]

Grupo	Nodos que conforman el grupo
Grupo #1	6-10-11-13
Grupo #2	4-8-9-12
Grupo #3	1-2-3-5-7

Tabla 3.6.2 Clasificación de las cargas según su tipo

Cada grupo representa un perfil de demanda típico, el grupo #1 Residencial, grupo #2 Comercial y grupo #3 Industrial. Para discretizar las curvas de carga en niveles, aunque estas no se conozcan se pueden utilizar una discretización por

niveles representativos que son un porcentaje de la demanda pico, este criterio es utilizado principalmente por N.I SANTOSO y O.T TAN [39] quienes consideraron que los niveles de carga que deben tenerse en cuenta ocurren para el 50%, 70%, 85% y 100% de la demanda pico. Según lo anterior para el nodo #1, las variaciones de la demanda a tenerse en cuenta son:

Valor de la demanda al 100%	2.0MW+j1.6MVAR
Valor de la demanda al 85%	1.7MW+j1.36MVAR
Valor de la demanda al 70%	1.4MW+j1.12MVAR
Valor de la demanda al 50%	1.0MW+j0.8MVAR

Tabla 3.6.3 Niveles representativos de la demanda pico para el nodo #1

Por lo tanto para tres grupos de demanda, cada uno de ellos con cuatro niveles de carga, el número total de combinaciones de carga es $\mathbf{4^3 = 64}$, que serán los datos con los cuales se entrenará la red, este criterio de discretización puede verse ampliamente en FLOREZ O. y SALAZAR H [11], de donde se extrajo la siguiente tabla que ilustra las 64 combinaciones posibles:

Combinación	Nivel grupo 1	Nivel grupo 2	Nivel grupo3
1	1,00	1,00	1,00
2	1,00	1,00	0,85
3	1,00	1,00	0,70
4	1,00	1,00	0,50
5	1,00	0,85	1,00
6	1,00	0,85	0,85
7	1,00	0,85	0,70
8	1,00	0,85	0,50
9	1,00	0,70	1,00
10	1,00	0,70	0,85

11	1,00	0,70	0,70
12	1,00	0,70	0,50
13	1,00	0,50	1,00
14	1,00	0,50	0,85
15	1,00	0,50	0,70
16	1,00	0,50	0,50
17	0,85	1,00	1,00
18	0,85	1,00	0,85
19	0,85	1,00	0,70
20	0,85	1,00	0,50
21	0,85	0,85	1,00
22	0,85	0,85	0,85
23	0,85	0,85	0,70
24	0,85	0,85	0,50
25	0,85	0,70	1,00
26	0,85	0,70	0,85
27	0,85	0,70	0,70
28	0,85	0,70	0,50
29	0,85	0,50	1,00
30	0,85	0,50	0,85
31	0,85	0,50	0,70
32	0,85	0,50	0,50
33	0,70	1,00	1,00
34	0,70	1,00	0,85
35	0,70	1,00	0,70
36	0,70	1,00	0,50
37	0,70	0,85	1,00
38	0,70	0,85	0,85
39	0,70	0,85	0,70
40	0,70	0,85	0,50
41	0,70	0,70	1,00
42	0,70	0,70	0,85
43	0,70	0,70	0,70
44	0,70	0,70	0,50
45	0,70	0,50	1,00
46	0,70	0,50	0,85
47	0,70	0,50	0,70
48	0,70	0,50	0,50
49	0,50	1,00	1,00
50	0,50	1,00	0,85
51	0,50	1,00	0,70
52	0,50	1,00	0,50
53	0,50	0,85	1,00
54	0,50	0,85	0,85
55	0,50	0,85	0,70
56	0,50	0,85	0,50
57	0,50	0,70	1,00
58	0,50	0,70	0,85
59	0,50	0,70	0,70

60	0,50	0,70	0,50
61	0,50	0,50	1,00
62	0,50	0,50	0,85
63	0,50	0,50	0,70
64	0,50	0,50	0,50

Tabla 3.6.4 Número total de combinaciones

El procedimiento para determinar la topología que disminuye las pérdidas para los 64 datos de entrada es [11]:

1. Leer los datos del sistema original
2. Leer una fila de los niveles de carga, es decir una fila de la tabla 3.6.4
3. Multiplicar la demanda pico de cada barra, por el nivel correspondiente leído en el numeral anterior.
4. Ejecutar el algoritmo de Gallego R [18], con los valores de carga establecidos en tres.
5. Ejecutar un flujo de carga radial, con la configuración encontrada en cuatro para determinar la efectividad del reconfigurador.
6. Repetir los pasos anteriores para cada una de las 64 combinaciones

Mediante este procedimiento se encontró que las dos topologías que garantizaban una disminución en las pérdidas de potencia activa son las ilustradas en la tabla 3.6.5:

Número de Configuración	***Ramas del Coárbol***		
1	9 - 7	8 - 6	10 - 1
2	11 - 10	9 - 7	8 - 6

Tabla 3.6.5 Configuraciones óptimas después de evaluar las 64 combinaciones

Para formar el set de entrenamiento los niveles de carga para cada nodo se representan por un vector de cuatro elementos el cual tendrá un uno en la posición que corresponda al nivel de carga, por consiguiente el set de entrenamiento estará conformado por una matriz de 54 filas, a las 52 primeras corresponden los niveles de carga en forma binaria y los dos últimos a la configuración que disminuye las pérdidas; cada columna es un patrón de entrenamiento. De esta forma las entradas a la red serán siempre 0 o 1, los patrones de entrenamiento de la red se presentan en la tabla 3.6.6

		Ejemplo 1										Ejemplo 64							
Fila1	Barra #1	1	1	1	1	1	1	1	1	.....		0	0	0	0	0	0	0	0
Fila2		0	0	0	0	0	0	0	0	.....		0	0	0	0	0	0	0	0
Fila3		0	0	0	0	0	0	0	0	.....		0	0	0	0	0	0	0	0
Fila4		0	0	0	0	0	0	0	1	.....		1	1	1	1	1	1	1	1
Fila5	Barra #2	1	1	1	1	1	1	1	1	.....		0	0	0	0	0	0	0	0
Fila6		0	0	0	0	0	0	0	0	.....		0	0	0	0	0	0	0	0
Fila7		0	0	0	0	0	0	0	0	.....		0	0	0	0	0	0	0	0
Fila8		0	0	0	0	0	0	0	0	.....		1	1	1	1	1	1	1	1
Fila9	Barra #3	1	1	1	1	1	1	1	1	.....		0	0	0	0	0	0	0	0
Fila10		0	0	0	0	0	0	0	0	.....		0	0	0	0	0	0	0	0
Fila11		0	0	0	0	0	0	0	0	.....		0	0	0	0	0	0	0	0
Fila12		0	0	0	0	0	0	0	0	.....		1	1	1	1	1	1	1	1
Fila41	Barra #11	0	1	0	0	0	1	0	1	0	.....	0	1	0	0	0	1	0	0
Fila42		0	0	1	0	0	0	0	0	1	.....	0	0	1	0	0	1	0	0
Fila43		0	0	0	1	0	0	0	0	0	.....	0	0	0	1	0	0	0	1
Fila44		1	0	0	0	1	0	0	0	0	.....	1	0	0	0	1	0	0	0
Fila45	Barra #12	0	1	0	0	0	1	0	0	0	.....	0	1	0	0	0	1	0	0
Fila46		0	0	1	0	0	0	1	0	0	.....	0	0	1	0	0	0	1	0
Fila47		0	0	0	1	0	0	0	1	0	.....	0	0	0	0	1	0	0	1
Fila48		1	0	0	0	1	0	0	0	1	.....	1	0	0	0	1	0	0	0
Fila49	Barra #13	1	0	0	0	1	0	0	0	0	.....	0	1	1	0	0	0	1	0
Fila50		0	0	1	0	0	0	1	0	0	.....	0	0	1	0	0	0	1	0
Fila51		0	0	1	0	0	0	0	1	0	.....	0	0	0	1	0	0	0	1
Fila52		1	1	0	0	1	0	0	0	1	.....	1	1	1	1	1	1	1	0
Configuración #1		1	1	1	1	1	1	1	1	1	.....	0	0	0	0	0	0	0	0
Configuración #2		0	0	0	0	0	0	0	0	0	.....	1	1	1	1	1	1	1	1

Tabla 3.6.6 Matriz de entrenamiento

3.6.2 Justificación del tipo de red.

El objetivo es entrenar una red neuronal competitiva y más explícitamente una red LVQ que para cada una de las 64 combinaciones generadas encuentre la mejor configuración, esta configuración consistirá en una de las dos especificadas en la tabla 3.6.5.

El entrenamiento de la primera capa de la red LVQ, la cual es una capa competitiva y por lo tanto de aprendizaje no supervisado, determinará las subclases dentro de las cuales se ubicará cada una de las combinaciones, estas subclases son cuatro (determinadas arbitrariamente).

El entrenamiento de la segunda capa de la red LVQ, la cual es una capa lineal de aprendizaje supervisado, requiere conocer las clases en las cuales se ubicará cada una de las salidas de la primera capa, estas clases son las dos topologías encontradas en el set de entrenamiento.

Un esquema general de la red que solucionará la aplicación es:

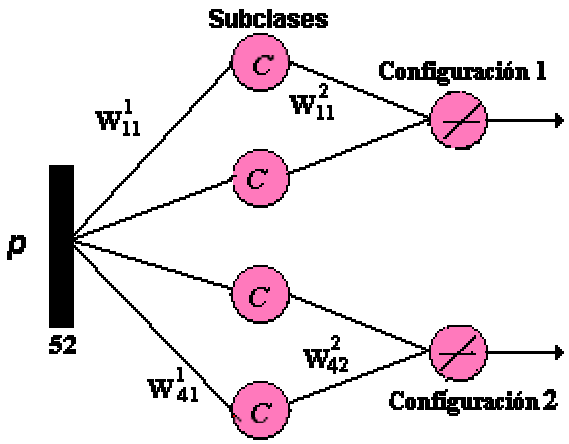

Figura 3.6.2 Esquema general de la red LVQ para reconfiguración

El vector de entrada ***p*** corresponde a cada una de las columnas de la tabla 3.6.6, las que conforman un vector de entrada de dimensiones 1x52, debido a que la reconfiguración se realizará para 13 nodos, cada uno con cuatro niveles de carga.

La función de transferencia *compet* retorna un vector de salida formado por ceros a excepción de la neurona ganadora, aquella cuyos pesos están más cercanos al vector de entrada y que ocasiona una salida de uno, esto es, las neuronas de esta capa compiten por ser la subclase ganadora. El vector de salida de la segunda capa, el cual depende de la subclase ganadora de la primera capa, determina la configuración óptima para cada patrón de entrada, de tal manera que si se obtiene un uno en la primera fila, la configuración óptima será la configuración número uno correspondiente a las ramas de coárbol 9-7, 8-6, 10 -1 y si el uno es obtenido en la segunda fila, la configuración óptima corresponde al coárbol 11-10, 9 -7, 8 – 6.

La disposición de las dos capas del algoritmo LVQ en notación compacta es la que se muestra en la figura 3.6.2.

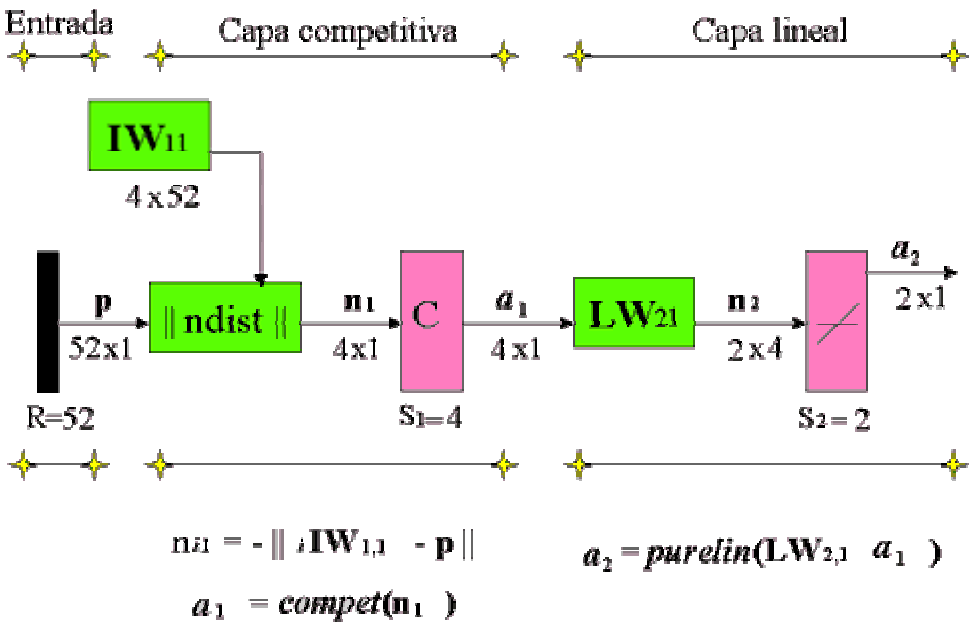

Figura 3.6.3 Red LVQ

La salida de la capa competitiva a_1 determina una posible configuración, la que red sospecha que es la configuración indicada a través del aprendizaje no supervisado, sin embargo la decisión de la mejor configuración es determinada en la capa dos, mediante un aprendizaje supervisado.

De esta forma la red LVQ ofrece una gran ventaja, puesto que en su primera capa la red tiene la libertad de determinar la "mejor" configuración, la cual es corroborada por la segunda capa, esto presenta dos mejoras, primero, un menor tiempo computacional en su entrenamiento y segundo, ofrece la flexibilidad para

que la red pueda generar sus propias representaciones (configuraciones) en la capa competitiva, la cual en el mejor de los casos puede ser una configuración completamente desconocida.

3.6.3 Entrenamiento de la red.

El código del algoritmo para esta aplicación fue desarrollado en Matlab 5.3, por medio de comandos generales sin utilizar la herramienta de redes neuronales, esto con el fin de utilizar los datos de la tabla 3.6.5 sin transformaciones; es necesario aclarar que para el entrenamiento de la red se emplearon las funciones de transferencia adecuadas para una red tipo competitiva.

Como datos de entrenamiento se tomaron 36 columnas de las 64 disponibles en la tabla 3.6.5, los valores restantes fueron escogidos como patrones de prueba para comprobar la capacidad de generalización de la red.

Los datos iniciales de los pesos fueron generados aleatoriamente con valores entre –1 y 1, por medio de la función *rands*

```
W1i=rands(52,36);
dim=size(W1);
cw=dim(2);
```

Los valores de W_2 (dos últimas filas de la tabla 3.6.5) representan las clases a las que corresponden los vectores de entrada en el algoritmo LVQ; en este caso, cada una de ellas muestra la configuración apropiada que debe adoptarse dependiendo de las condiciones impuestas por los datos de entrada.

La rata de aprendizaje, se varió hasta encontrar un valor óptimo de 0.45

```
alfa=0.45;
```

Los comandos mediante los cuales se realiza la validación y actualización de los pesos son los siguientes:

```
for i=1:cw
   for j=1:cw
      n(j,1)=-norm(W1(:,j)-p(:,i));
   end
  a1=compet(n);
    a2=W2*a1;
    e(:,i)=a2-W2(:,i);

    if e(:,i)==0
       W1(:,i)=W1(:,i)+alfa*(p(:,i)-W1(:,i));
       else
       l=find(a1);
       W1(:,l)=W1(:,l)-alfa*(p(:,i)-W1(:,l));
    end
end
```

Después de ejecutar este algoritmo para *seis iteraciones* la red alcanzó convergencia, obteniendo los siguientes valores para la matriz de pesos de la capa competitiva:

W1=net.IW{1,1}

	1	**2**	**3**	**4**	.....	**48**	**49**	**50**	**51**	**52**
N1	1,1229	0,8898	1,1379	1,052	.....	0,0232	-0,111	0,0271	-0,0448	0,0603
N2	-0,0713	0,0111	-0,353	-0,2442	.....	-0,0773	-0,1375	-0,2203	0,0168	-0,0654
N3	-0,1078	-0,1675	-0,0779	-0,1233	.....	0,9166	-0,0185	0,0619	0,9126	-0,0326
N4	-0,0037	0,0525	0,1002	0,0298	.....	-0,0532	0,9624	1,0164	-0,0679	0,9667

Tabla 3.6.7 Pesos de la capa competitiva

Los pesos para la segunda capa se visualizan en la tabla 3.6.7, los cuales como se esperaba, corresponden a la salida deseada de la red.

W2=net.LW{2,1}

	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**18**
N1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1
N2	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0

	19	**20**	**21**	**22**	**23**	**24**	**25**	**26**	**27**	**28**	**29**	**30**	**31**	**32**	**33**	**34**	**35**	**36**
N1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
N2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Tabla 3.6.8 Pesos de la capa lineal

El algoritmo LVQ no tiene ganancias, aún así la red alcanza un eficiente desempeño clasificando correctamente todos los patrones, es decir encontrando la configuración más adecuada para cada una de las condiciones del sistema, esta

evaluación del desempeño de la red se hace con base en los flujos de carga ejecutados para las diferentes combinaciones y comparándolo con la configuración óptima determinada por la red, ya que al ser éste tipo de red un híbrido entre aprendizaje supervisado y competitivo no es posible determinar un error en su entrenamiento, pues el vector de salida para la primera capa no está especificado.

Esta red tiene grandes ventajas con respecto a otras que podrían también realizar esta aplicación, una de ellas es el tiempo de cómputo, el cual es más corto para una red LVQ, comparado con el tiempo que invierte por ejemplo una Backpropagation en alcanzar convergencia para el mismo número de patrones de entrenamiento [11].

Otra importante característica es la configuración de la red, pues el número de neuronas de cada una de las dos capas queda determinado por las mismas condiciones del problema. Además su capacidad de generalización no se ve afectada por la cantidad de patrones de entrada a la red.

3.7 IDENTIFICACION DE UN SISTEMA DINÁMICO NO LINEAL

3.7.1 Descripción del Problema: La identificación de un sistema consiste en determinar una función que relacione las variables de entrada con las variables de salida, esta identificación tradicionalmente se hace a través de técnicas de regresión y con un conocimiento previo de la naturaleza del sistema.

Clásicamente el comportamiento de un sistema se describe a través de sus ecuaciones de estado, las cuales son un conjunto de ecuaciones diferenciales no lineales de primer orden, donde el objetivo es encontrar las funciones f y g mostradas en las siguientes ecuaciones:

$$\frac{\partial x(t)}{\partial t} = f[x(t), u(t)] \qquad (3.7.1)$$

$$y(t) = g[x(t)] \qquad (3.7.2)$$

En la identificación de un sistema pueden conocerse previamente el orden del sistema y desconocer de forma precisa sus parámetros o desconocer totalmente el orden del sistema y sus parámetros; la identificación de un sistema mediante redes neuronales no pretende encontrar las funciones f y g, sino un mapa de la relación entrada - salida con base a los patrones con que fue entrenada.

El sistema que se quiere identificar es el péndulo invertido propuesto por Kuschewski et al. (1993), mostrado en la figura 3.7.1

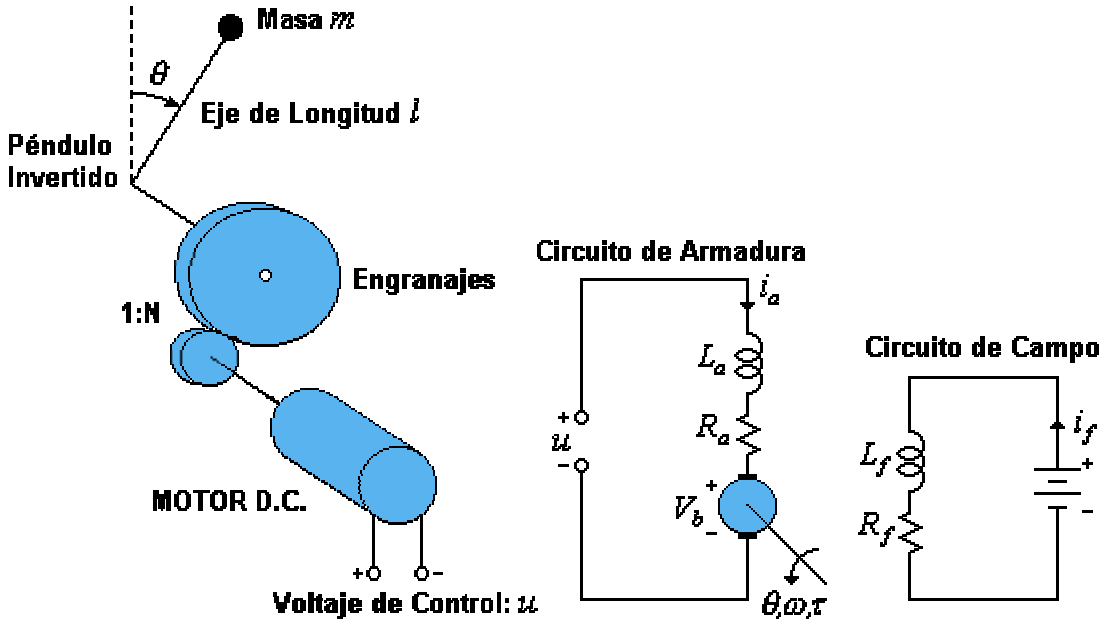

Figura 3.7.1 Péndulo Invertido

Las ecuaciones diferenciales no lineales que describen el comportamiento del sistema son:

$$ml^2\ddot{\theta} + mgl\ Sen\theta = \tau_m \qquad (3.7.3)$$

$$L_{fa}\frac{di_a}{dt} + R_{fa}i_{fa} + V_b = u \qquad (3.7.4)$$

$$\tau_m = N_t K_m i_a \qquad (3.7.5)$$

$$V_b = N_t K_b \theta_m \qquad (3.7.6)$$

K_m: Constante de Torque del motor = 0.1NM/A

K_b: Constante de Fuerza Contra electromotriz = 0.1Vs/rad

N_t: Relación de Transformación entre los piñones = 10

g = 9.8 m/s^2 l = 1m m = 1 kg R_a = 1 Ω L_a = 1 mH

Con estos parámetros, las ecuaciones de estado que representan el comportamiento del Péndulo Invertido están dadas por:

$$\begin{bmatrix} \dot{X}_1 \\ \dot{X}_2 \\ \dot{X}_3 \end{bmatrix} = \begin{bmatrix} X_2 \\ -9.8Sen(X_1) + X_3 \\ -10X_2 - 10X_3 \end{bmatrix} \begin{bmatrix} X_1 \\ X_2 \\ X_3 \end{bmatrix} \quad (3.7.7)$$

$$y = \begin{bmatrix} 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} X_1 \\ X_2 \\ X_3 \end{bmatrix} \quad (3.7.8)$$

Las variables de estado del sistema son:

$$X_1 = \theta,\ X_2 = \frac{d\theta}{dt},\ X_3 = i_a \quad (3.7.9)$$

3.7.2 Justificación del tipo de red.

Según lo demostrado en la sección 2.6.1 del capitulo 2, una red neuronal dinámica recurrente puede identificar el comportamiento de un sistema dinámico descrito

por ecuaciones diferenciales no lineales si se tiene abundante y representativa información de sus datos de entrada y salida.

Según lo demostrado en la sección 2.6.2 un sistema dinámico autónomo ($u=0$) puede ser identificado una red dinámica multicapa, utilizando en la capa estática un algoritmo tipo backpropagation

3.7.3 Entrenamiento de la red.

3.7.3.1 Identificación del Sistema Autónomo (u=0). Utilizando una red dinámica multicapa: Se identificará la dinámica del péndulo invertido autónomo considerando nulo el voltaje aplicado.

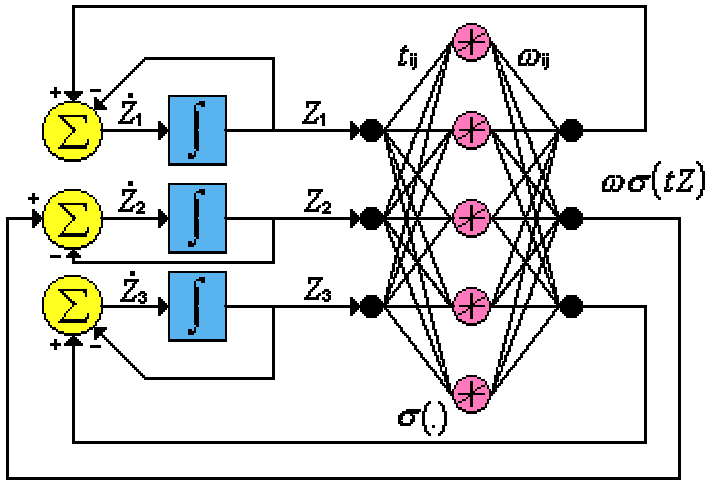

Figura 3.7.2 Red multicapa

Para encontrar los pesos tij y wij de la red recurrente multicapa, se entrenará la capa estática con un algoritmo de propagación inversa y luego se implementará la red de la figura 3.7.2 para identificar plenamente la dinámica del sistema autónomo.

El set de entrenamiento se generó con variaciones en el ángulo, la velocidad, la corriente y todas sus posibles combinaciones en los siguientes rangos:

	Valor Inicial	Valor Final	Incremento	Conversión
Ang	-30º	30º	12º	pi/180º
Vel	-115º	46º	115º	pi/180º
u	0	8	1.6	-

Tabla 3.7.1 Rangos del set de entrenamiento

Los patrones de entrenamiento esperados se calculan como el valor actual del patrón de entrada más el valor de cada una de sus derivadas como se explicó en la sección 2.6.2, el valor de las derivadas de las variables de estado se obtiene al evaluar su valor actual en el conjunto de ecuaciones que describe el comportamiento de la planta.

P	P1	P2		P42		P144	P145		P180		P216
Z1	-0.5236	-0.3142	....	0.5236	....	0.5236	-0.5236	....	0.5236	....	0.5236
Z2	-2.0071	-2.0071	....	-2.0071	....	2.0071	-2.0071	....	2.0071	....	2.0071
Z3	0.0000	0.0000	....	1.6000	....	4.8000	6.4000	....	6.4000	....	8.0000

t	t1	t2		t42		t144	t145		t180		t216
Z1+dZ1	-2.5307	-2.3213	...	-1.4835	...	2.5307	-2.5307	...	2.5307	...	2.5307
Z2+dZ2	2.8929	1.0212	...	-5.3071	...	1.9071	9.2929	...	3.5071	...	5.1071
Z3+dZ3	20.0713	20.0713	...	5.6713	...	-63.2713	-37.5287	...	-77.6713	...	-92.0713

Tabla 3.7.2 Patrones de entrenamiento

Para la capa estática tipo backpropagation los pesos iniciales son:

W1i=net.IW{1,1}

	Z1	Z2	Z3
N1	2.4024	-1.0466	-0.0833
N2	-2.3909	0.2432	-0.4966
N3	2.6014	0.2716	-0.4746
N4	2.2506	-0.6173	-0.4285
N5	-2.7268	0.1526	-0.4748

W2i=net.LW{2,1}

	N1	N2	N3	N4	N5
Z1	-0.2321	-0.9293	-0.9685	0.1738	0.2629
Z2	0.3662	0.2248	-0.9673	-0.8848	0.4353
Z3	-0.8143	0.2171	-0.6199	-0.2649	0.3853

Tabla 3.7.3 Valores iniciales de los Parámetros de la Capa Estática

El siguiente código crea una red de 3 entradas, 5 neuronas en la capa oculta y 3 salidas y la entrena con el algoritmo *trainlm* (Ver Anexo A):

```
net=newff(minmax(P),[5 3],{'tansig' 'purelin'},'trainlm');
net.trainParam.epochs=1000;
net.trainParam.goal=1e-06;
net.trainParam.lr=0.03
net.trainParam.mem_reduc=1
[net,tr]=train(net,P,t);
```

Luego de 500 iteraciones el error medio cuadrático cae por debajo de $1x10^{-6}$, los valores de los parámetros de la red luego del proceso de entrenamiento se muestran en la tabla 3.7.2

tij=W1f=net.IW{1,1}

	Z1	Z2	Z3
N1	0.7305	0.0000	0.0000
N2	0.0002	0.0211	0.0188
N3	-0.0003	-0.0212	-0.0189
N4	-0.0003	-0.0287	-0.0256
N5	-0.0078	-0.0074	0.0004

wij=W2i=net.LW{2,1}

	N1	N2	N3	N4	N5
Z1	-0.0004	10.1756	7.9010	-1.1967	-127.9416
Z2	-13.2698	16.1031	-28.6495	-24.1701	15.6964
Z3	0.0000	-217.4789	187.7513	218.6130	-13.3227

Tabla 3.7.4 Valores finales de los Parámetros de la Capa Estática

Según lo demostrado en la sección 2.6.2, el Jacobiano de la red multicapa en el origen se calcula como:

$$J_M = -I_N + WT \qquad (3.7.10)$$

Reemplazando en la ecuación 3.7.10, los valores de la tabla 3.7.4 el Jacobiano de la red multicapa es como se sigue:

JM

0.0002	10.260	0.0231
-9.7995	0.5254	1.4680
-0.0490	-14.7521	-14.2328

Tabla 3.7.5 Jacobiano de la Red Multicapa

Para una identificación exitosa los valores propios del Jacobiano de la red multicapa mostrado en la tabla 3.7.5 deben ser muy próximos a los valores propios del sistema descritos en la ecuación 3.7.10

$$\begin{aligned} \lambda_1 &= -0.5207 + j3.2816 \\ \lambda_2 &= -0.5207 - j3.2816 \\ \lambda_3 &= -12.6657 \end{aligned} \tag{3.7.11}$$

Los valores propios del sistema son calculados del sistema de ecuaciones 3.7.7

$$eigen\begin{bmatrix} 0 & 1 & 0 \\ -9.8 & 0 & 1 \\ 0 & -10 & -10 \end{bmatrix} \Rightarrow \begin{aligned} \lambda_1 &= -0.4952 + j3.2607 \\ \lambda_2 &= -0.4952 - j3.2607 \\ \lambda_3 &= -9.0096 \end{aligned} \tag{3.7.12}$$

Como se observa, los valores del Jacobiano de la red multicapa se encuentran próximos a los del sistema, por lo tanto la red a identificado el sistema correctamente, nótese además que todos los valores propios tanto de la red como del sistema se encuentran en el semiplano complejo izquierdo haciendo que el sistema y la red neuronal sean estables.

Según lo demostrado en la sección 2.6.2, una red dinámica multicapa puede ser transformada en una red dinámica tipo Hopfield por medio de la siguiente transformación:

$$\frac{d}{dt}\chi = -I_N\chi + TW\sigma(.) \tag{3.7.13}$$

Tomando los valores de los parámetros de la red multicapa y reemplazándolos en la ecuación 3.7.13 se obtiene la matriz de pesos de la red dinámica de Hopfield equivalente a la red dinámica recurrente, la matriz de pesos de la red de Hopfield se muestra en la tabla 3.7.6

Matriz de Pesos Red de Hopfield = TW

-0.0003	7.4337	5.7721	-0.8742	-93.4671
-0.2803	-3.7494	2.9285	3.6020	0.0568
0.2810	3.7589	-2.9370	-3.6117	-0.0488
0.3813	5.0998	-3.9839	-4.8997	-0.0721
0.0979	-0.2832	0.2229	0.2730	0.8793

Tabla 3.7.6 Matriz de pesos Red de Hopfield equivalente

El Jacobiano de la red de Hopfield según lo demostrado en la sección 2.6.2 se calculan como:

$$J_H = -I_N + TW \tag{3.7.14}$$

y tiene los siguientes valores propios:

$$\begin{aligned}
\lambda_1 &= -0.5207 + j3.2816 \\
\lambda_2 &= -0.5207 - j3.2816 \\
\lambda_3 &= -12.6657 \\
\lambda_4 &= -1 \\
\lambda_5 &= -1
\end{aligned} \tag{3.7.15}$$

Nótese que los tres primeros valores propios de la ecuación 3.7.15 coinciden con los valores propios de la ecuación 3.7.12, y éstos a su vez se encuentran muy próximos a los valores propios del sistema, los valores propios adicionales de la ecuación 3.7.15 corresponden a los estados adicionales que se agregaron para una adecuada identificación del sistema.

Para comprobar la efectividad de la aproximación, en la figura 3.7.3 se muestra la comparación de la respuesta de las variables de estado del sistema con la respuesta de la red neuronal multicapa; como la identificación que se realizó fue del sistema autónomo $(u=0)$, para la simulación se requieren condiciones iniciales en las variables de estado del sistema y en las variables de estado estimadas de la res multicapa, los valores iniciales para la simulación fueron: $X1^0=0.2$, $X2^0=-0.3$, $X3^0=0$.

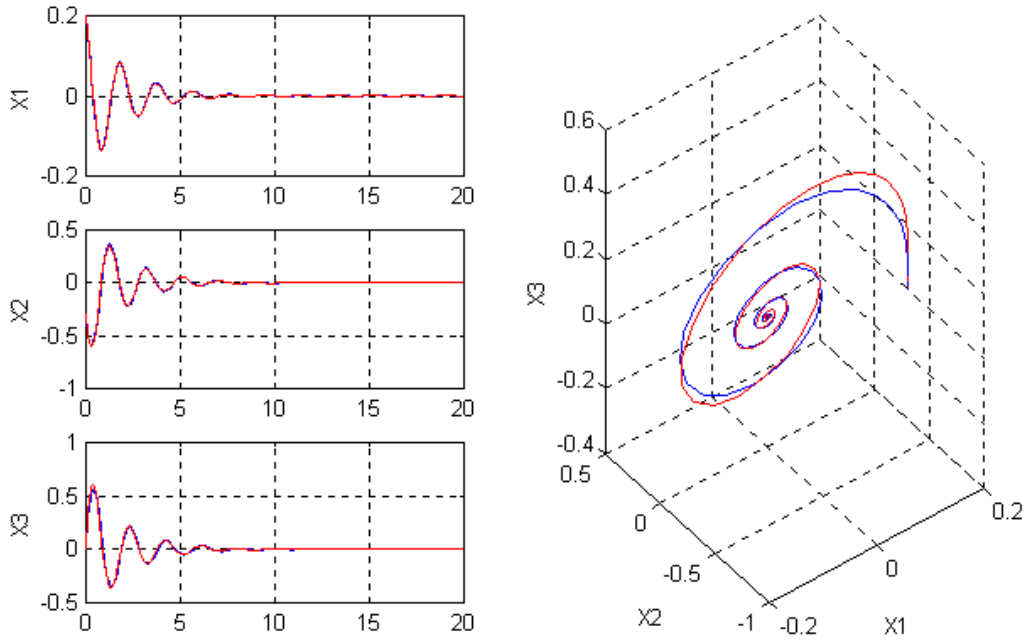

Figura 3.7.3 Respuesta del sistema Vs Red Multicapa

La línea azul representa la respuesta del sistema y la línea roja muestra la respuesta de la red dinámica multicapa, en la gráfica de la izquierda se grafica cada variable de estado en función del tiempo y en la gráfica de la derecha se tienen los planos de fase descritos por las tres variables de estado, de la gráfica 3.7.3 se observa, que se obtuvo una muy buena aproximación del sistema autónomo y de la ecuación 3.7.11 se tiene una buena aproximación de los valores propios del sistema

3.7.3.2 Identificación de la dinámica del Sistema. Se entrenará una red recurrente según el procedimiento descrito en el tutorial del Matlab versión 5.3 [34], esta red tiene como entradas el ángulo, la velocidad, la corriente de armadura y el voltaje aplicado, para identificar

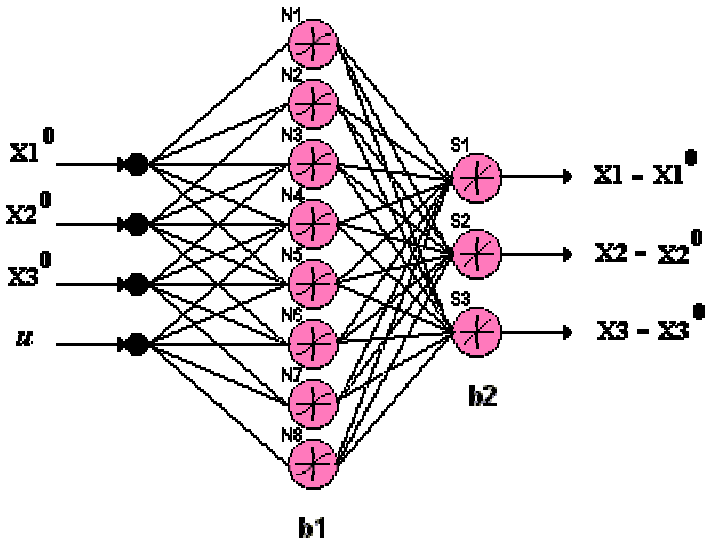

Figura 3.7.4 Capa Estática Red Recurrente

De la figura 3.7.4, se puede observar que la red recurrente posee una capa backpropagation, para la cual el set de entrenamiento fue generado con variaciones en el ángulo, la velocidad, la corriente y todas sus posibles combinaciones

	Valor Inicial	Valor Final	Incremento	Conversión
Ang	-30º	30º	15º	pi/180º
Vel	-115º	55º	115º	pi/180º
ia	0	8	2	-
u	-6	3	6	-

Tabla 3.7.7 Datos de entrenamiento

Adicionalmente se incluyen en el set de entrenamiento datos de posiciones iniciales con velocidad y corriente igual a cero

	Valor Inicial	Valor Final	Incremento	Conversión
Ang	-30º	30º	6º	pi/180º
Vel	0	0	0	-
ia	0	0	0	-
u	0	0	0	-

Tabla 3.7.8 Datos adicionales de entrenamiento

Todos los patrones de entrenamiento se evaluaron en el conjunto de ecuaciones que describen el comportamiento del sistema, las ecuaciones diferenciales no lineales fueron resueltas en cada caso para un tiempo tstep=0.05s, la salida esperada para cada patrón de entrada es el vector solución X menos el valor de Xo utilizado como condición inicial.

```
tstep=0.05;
t=zeros(3,length(P));
for i=1:length(P)
    [time,X]=ode45('modelo',[0 tstep],P(:,i));
   t(:,i)=X(length(X),1:3)'-P(1:3,i);
end
```

P	P1	P2		P72		P144	P145		P340		P636
$X1^0$	-0.5236	-0.2618	....	-0.2618	....	0.2618	0.5236	....	0.5236	....	0.5236
$X2^0$	-2.0071	-2.0071	....	1.8326	....	0.8727	0.8727	....	-0.0873	....	0.0000
$X3^0$	0.0000	0.0000	....	4.0000	....	0.0000	0.0000	....	6.0000	....	0.0000
u	-6.0000	-6.0000	....	-6.0000	....	-3.0000	-3.0000	....	0.0000	....	0.0000

t	t1	t2		t72		t144	t145		t340		t636
$X1-X1^0$	-0.0946	-0.0975	...	0.0972	...	0.0396	0.0367	...	-0.0041	...	-0.0061
$X2-X2^0$	0.2217	0.1069	...	0.1770	...	-0.1772	-0.2935	...	-0.0072	...	-0.2432
$X3-X3^0$	-1.6197	-1.5950	...	-4.7026	...	-1.4890	-1.4641	...	-2.3287	...	0.0520

Tabla 3.7.9 Patrones de entrenamiento P y patrones esperados t

La red fue entrenada con el algoritmo backpropagation *trainlm* con los siguientes pesos iniciales:

W1i=net.IW{1,1}

	I1	I2	I3	I4
N1	4.3432	0.1214	-0.1408	-0.0044
N2	1.8875	0.1985	-0.2226	0.3175
N3	1.7496	0.7324	-0.3268	0.1682
N4	3.6430	-0.1749	-0.3306	0.0347
N5	2.6789	0.6421	0.0647	0.2351
N6	-2.4939	-0.5881	0.3700	0.1018
N7	-1.4134	-0.1684	-0.4461	0.2178
N8	-1.7871	0.1220	0.0541	-0.3562

b1i=net.b{1}

N1	-1.7807
N2	-0.7742
N3	0.3622
N4	0.9709
N5	0.1337
N6	-2.5404
N7	0.0877
N8	-2.5602

W2i=net.LW{2,1}

	N1	N2	N3	N4	N5	N6	N7	N8
S1	-0.5745	-0.8180	-0.1714	0.8758	-0.3649	-0.6993	-0.2245	0.1744
S2	0.4294	-0.4508	-0.9462	-0.5202	0.7740	0.3627	-0.0005	0.6912
S3	-0.7391	-0.9940	0.4196	-0.6382	0.3041	-0.2284	-0.7049	0.1802

b2f=net.b{2}

S1	0.9108
S2	0.1123
S3	-0.7037

Tabla 3.7.10 Pesos iniciales red recurrente

El siguiente código entrena la red backpropagation de 4 entradas, 8 neuronas en la capa oculta y 3 salidas con el algoritmo de entrenamiento *trainlm.*

```
net=newff(minmax(P),[8 3],{'tansig' 'purelin'},'trainlm');
net.trainParam.epochs=700;
net.trainParam.goal=1e-8;
net.trainParam.lr=0.03
net.trainParam.mem_reduc=1
net.trainParam.min_grad=1e-10
[net,tr]=train(net,P,t);
```

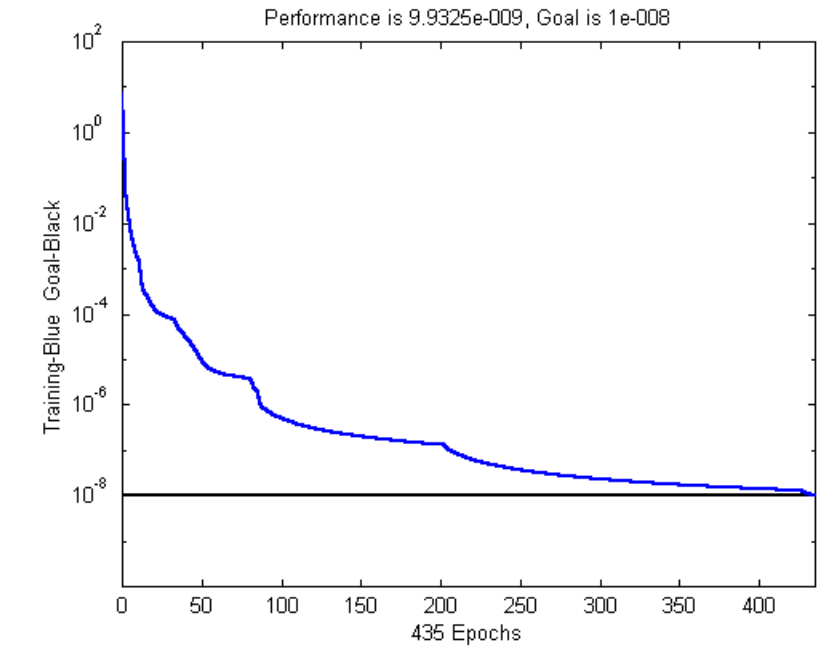

Figura 3.7.5 Error medio cuadrático utilizando *trainlm*

Después de 435 iteraciones de entrenamiento, el error medio cuadrático cayó por debajo de $1x10^{-8}$, los pesos y ganancias de la red entrenada son:

W1f=net.IW{1,1}

	I1	I2	I3	I4
N1	-0.0043	0.0241	0.0298	-0.0246
N2	0.7562	0.0193	0.0003	0.0001
N3	0.0139	0.0040	-0.0005	-0.0173
N4	5.3898	-0.9620	0.1449	0.0234
N5	0.0007	0.0333	-0.0006	-0.0002
N6	-0.2605	-0.0725	0.0168	0.2906
N7	-3.5076	-0.4256	-1.7086	0.0763
N8	-0.0047	0.0266	0.0329	-0.0272

b1f=net.b{1}

N1	0.5084
N2	0.0001
N3	-0.0546
N4	2.5040
N5	0.0052
N6	-1.8745
N7	0.2389
N8	-0.8470

W2f=net.LW{2,1}

	N1	N2	N3	N4	N5	N6	N7	N8
S1	0.0516	-0.0148	-0.1203	0.00001	1.4688	0.0002	0.00000	0.0425
S2	0.9975	-0.5914	-2.3910	0.00000	-0.9662	0.0033	-0.00001	0.8211
S3	-10.8348	0.1225	-3.2830	-0.00005	-1.5610	0.0045	0.00003	-8.8873

b2f=net.b{2}

S1	-0.0089
S2	-0.0235
S3	-1.2166

Tabla 3.7.11 Pesos finales de la red recurrente

Para comprobar si la red neuronal recurrente identifica correctamente la dinámica de la planta, esta fue simulada con los pesos finales anteriormente encontrados utilizando el siguiente algoritmo, con condiciones iniciales mostradas de las variables de estado X1 (ángulo) = -8º, X2 (velocidad) = 0 º/s, X3 (corriente de armadura) = 0 A, y u (voltaje)= 1 V, estos valores están contenidos en *Xo*

```
Xo=[-8*pi/180; 0; 0];
u=1;
tstep=0.05;
times=0:tstep:10;
estado=Xo;
estados=zeros(3,length(times));
estados(:,1)=estado;
for i=2:length(times)
  estado=estado+sim(net,[estado;u]);
  estados(:,i)=estado;
end
```

Para obtener la respuesta de la red en el tiempo deseado, esta debe ser simulada en intervalos de tiempo iguales al tiempo para el cual fueron resueltas las ecuaciones del set de entrenamiento, en este caso las ecuaciones fueron resueltas para un tiempo tstep=0.05s, por lo tanto para obtener la respuesta de la red en un tiempo total de 10s, debe simularse 201 veces en intervalos de 0.05s.

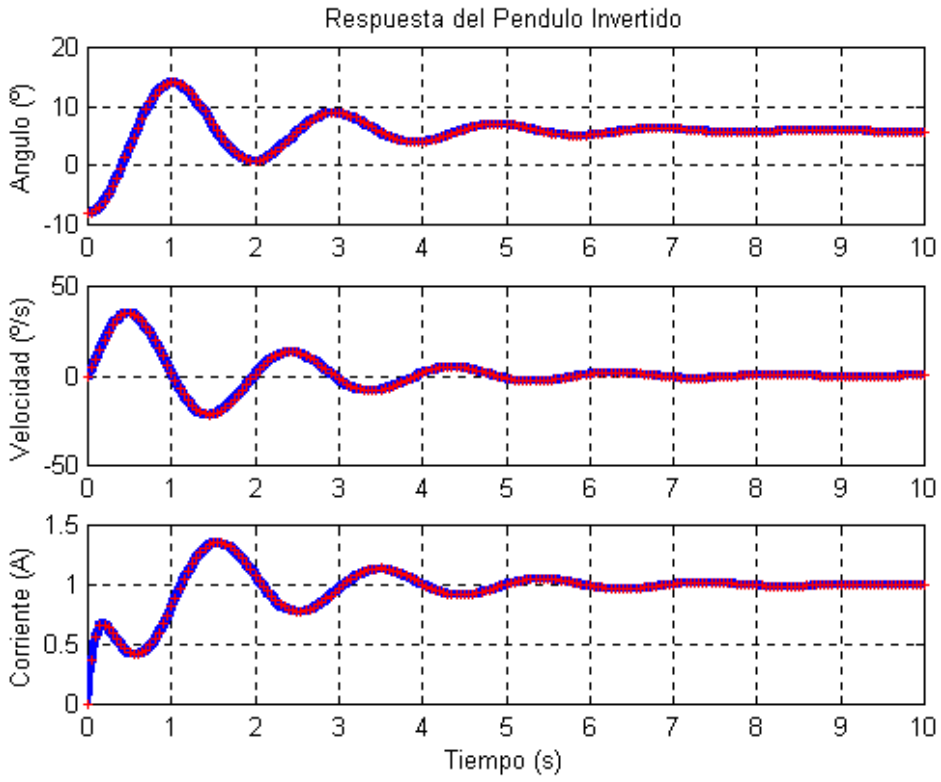

Figura 3.7.6 Comparación de la respuesta del sistema y la RN

La línea continua azul representa la respuesta del sistema simulado y las cruces rojas representan las simulaciones de la red neuronal entrenada, como se observa la red neuronal ha identificado la dinámica de la red perfectamente, en la siguiente gráfica puede observarse que los planos de fase descritos por la planta y por la red neuronal coinciden exactamente.

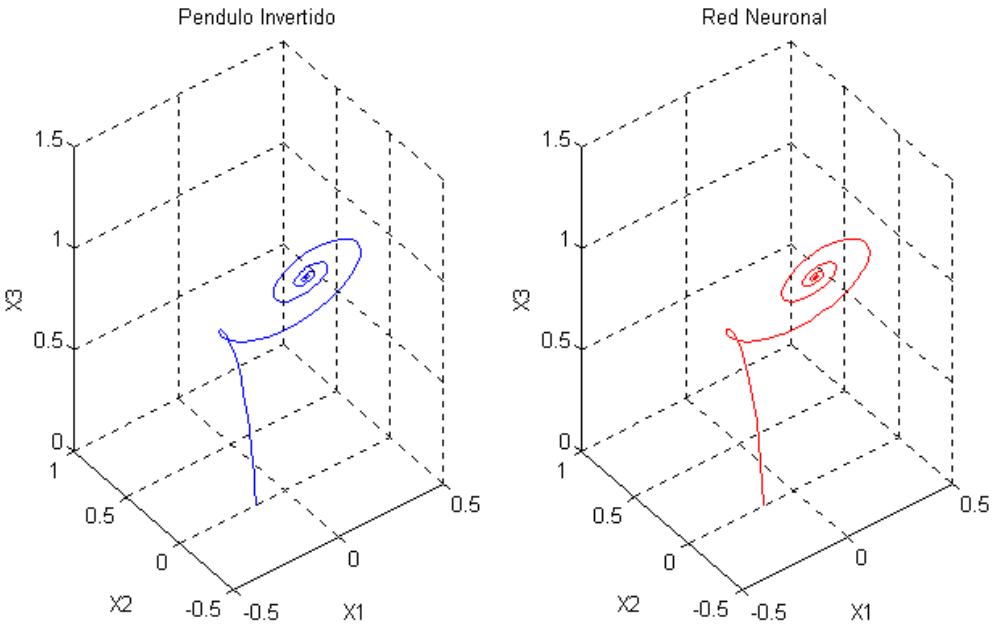

Figura 3.7.7 Comparación de los planos de fase

Como se observa en la figura 3.7.7, cuando la misma red neuronal es simulada en intervalos de tiempo distintos al tiempo para el cual fueron solucionadas las ecuaciones que generaron el set de entrenamiento, la red produce las mismas respuestas que la planta en estado estacionario, pero en el estado transitorio la

forma de la respuesta es similar pero en el caso de un intervalo de simulación menor tstep=0.3 la red neuronal responde mas rápido que la planta y en el caso de un intervalo de simulación mayor tstep=0.10 la red neuronal responde mas lento que la planta, para una identificación óptima de la planta tanto en estado estacionario como en estado el intervalo de simulación debe ser igual a tiempo de solución de las ecuaciones que generaron los patrones de entrenamiento.

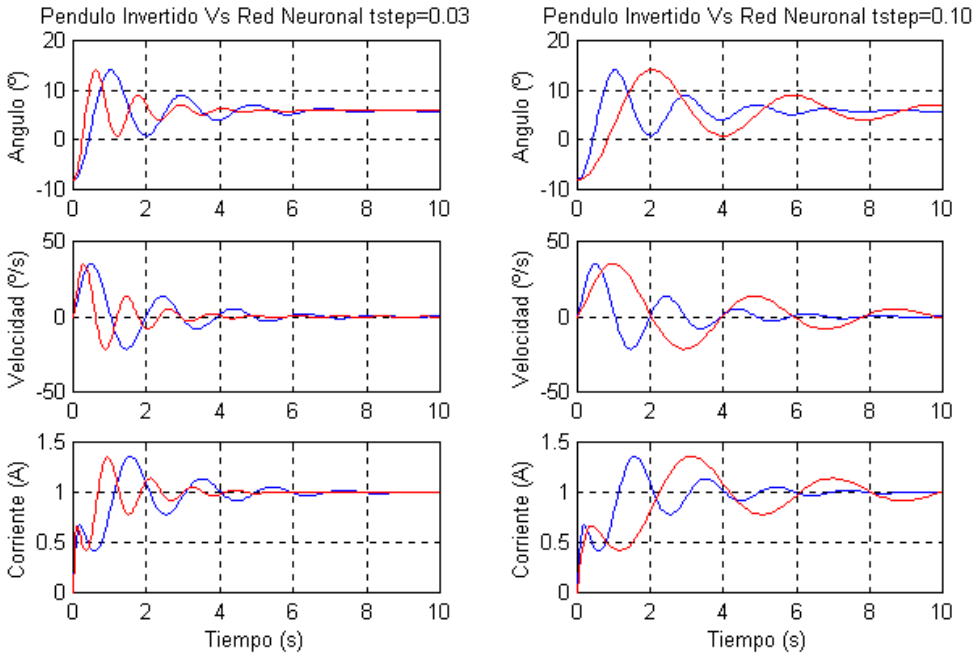

Figura 3.7.8 Red neuronal con diferentes tiempos de simulación