UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

POSGRADO EN CIENCIA E INGENIERÍA DE LA COMPUTACIÓN

"DETECCIÓN DE FALLAS USANDO MAPAS AUTOORGANIZADOS Y LA REPRESENTACIÓN DINÁMICA DEL CASO DE ESTUDIO"

T E S I S

QUE PARA OBTENER EL GRADO DE:

MAESTRA EN CIENCIAS (COMPUTACIÓN)

P R E S E N T A:

LAURA GUTIÉRREZ RUIZ

DIRECTOR DE TESIS: DR. HÉCTOR BENÍTEZ PÉREZ

México, D.F. **2006.**

Agradezco

A la Universidad Nacional Autónoma de México.

Al Instituto de Investigación en Matemáticas Aplicadas y Sistemas (IIMAS)

Al Consejo Nacional de Ciencia y Tecnología (CONACYT)

Al Proyecto PAPIIT IN105-303.

Por el apoyo recibido para lograr ésta tesis.

Dedico ésta tesis:

A mis Padres, por su apoyo, comprensión y cariño.

A David, por su constante ejemplo de esfuerzo y trabajo que me ayudo a ser mejor.

A Rocío, por la motivación que me ha brindado durante mucho tiempo.

A Sebastián, por ser alegría en esos días que más lo necesitaba.

A Jonatan, por su apoyo incondicional.

Y a todos mis compañeros de maestría, pues cada uno me aportó cosas muy importantes.

CONTENIDO

Introducción

Revisión del Campo

La detección de fallas en sistemas dinámicos se ha convertido en un campo maduro de estudio a través de los últimos años, esto derivado de la necesidad de ofrecer técnicas más eficientes, confiables y seguras con relación a una alta capacidad de mantenimiento y seguridad del propio sistema dinámico.

La detección y clasificación de fallas es una tarea difícil de realizar dada la naturaleza incierta en el comportamiento del sistema monitoreado durante un escenario con fallas. Por lo que la implantación de herramientas concernientes al campo de la Inteligencia Artificial (IA) para desarrollar dicho objetivo ha sido aplicada con relativo éxito.

Distintas técnicas de IA han sido propuestas, tales como sistemas expertos, árboles de decisión, sistemas de inferencia lógica, sistemas basados en el conocimiento así como las redes neuronales artificiales, que han sido favorablemente aplicadas como aproximadores universales de sistemas altamente no lineales. Es por ello que su uso en clasificación, control, y procesamiento de señales resulta común aun cuando no son técnicas fáciles de sintonizar para cada caso de estudio.

Por ejemplo, las redes neuronales artificiales pueden ser entrenadas para la detección de fallas, aprendiendo el conocimiento de los expertos mediante un conjunto de datos representativo. Esto es posible debido a su habilidad de reconocimiento de patrones. Otra característica es no requerir modelos básicos en la representación del sistema monitoreado a fin de tomar una decisión. El modelo del proceso puede ser solamente aproximado o parcial ya que las redes neuronales son capaces de mapear internamente las relaciones funcionales que representen al proceso.

Sin embargo, en algunos casos, las redes neuronales se encuentran limitadas cuando la detección de fallas se quiere realizar en un proceso en línea, ya que es necesario tener una gran precisión en la medida de los valores necesarios para llevar a cabo la detección. Debido a que las mediciones se efectúan a través de instrumentos que pueden no ser suficientemente sensibles, la presencia de ruido llega a ser muy probable. En este caso la red neuronal pierde sensibilidad y puede no ser eficiente en la detección de la falla. Para esto se hace necesario el pre-procesamiento de los datos a fin de que se le presente a la red los parámetros significativos.

Por lo que en ésta tesis se propone como método de pre-procesamiento la estimación de parámetros del sistema dinámico empleando el método de mínimos cuadrados. Dicho método obtendrá el modelo en diferencias parcial del sistema, el cual se presentará a la red como entradas a ser aprendidas que contengan la representación más aproximada del sistema en presencia de posibles perturbaciones. Así se logrará que la red obtenga sensibilidad sin perder generalización.

Ya que la detección y clasificación de fallas de un sistema dinámico se realiza sobre un conjunto de datos los cuales no se sabe a que clase pertenece, ni su proximidad a distintos escenarios ya clasificados, el uso de un método de aprendizaje no supervisado que contemple una cercanía a través de un manejo multidimensional adecuado se convierte en lo más apropiado. Por esto, se emplean los mapas auto-organizados (SOM, por sus siglas en inglés Self Organizing Maps), ya que la topología del espacio de entradas es preservada [Kohonen, 1995] [Jämsa-Jounela et. al., 1988], lo que permite una definición más conveniente en la compatibilidad de la medida.

Metas

Se pretende construir un esquema de detección y clasificación de fallas en línea con la capacidad de distinguir las transiciones inherentes a cambios de puntos de operación así como un monitoreo certero en la clasificación de fallas.

Objetivo

El objetivo del presente trabajo es la detección y clasificación de fallas en sistemas dinámicos utilizando estimación de parámetros como un constructor local del sistema observado y mapas auto-organizados como un clasificador de su conducta.

Para alcanzar dicho objetivo se ejemplifica el algoritmo que llevará a cabo la detección y clasificación de fallas con un ejemplo. La señal a analizar es obtenida a través de simulación por medio del uso del paquete Simulink que Matlab ofrece, para posteriormente llevar a cabo la estimación de parámetros. Una vez obtenido el modelo matemático del sistema, este se presenta a la red por medio de la ley de aprendizaje correspondiente. Este proceso se ejecuta en una etapa fuera de línea. Seguido a esto, se procede a una etapa en línea para la detección de posibles fallas, presentando a la red el vector de parámetros actual en la ventana de tiempo considerada (lo cual es la representación del modelo matemático del sistema); se calcula una distancia euclidiana entre el vector y la matriz de pesos correspondiente a la red, para encontrar así la mínima distancia que nos dará la neurona activada para dicho vector, a fin de conocer si el vector de parámetros actual corresponde o no a una falla.

Mapa de la Tesis

El desarrollo de esta tesis se encuentra dividido en 4 capítulos. El capítulo 1 da una descripción general de las redes neuronales profundizando en los mapas auto-organizados. El capítulo 2 hace una descripción del método empleado para la estimación de parámetros como modelado matemático del sistema. Cabe señalar que este capítulo no comprende una teoría profunda de la estimación de parámetros, pues únicamente se emplea como herramienta para el procesamiento de la señal. En el capítulo 3 se introduce al diseño experimental del modelo de detección de fallas. Para la estimación de parámetros se empleó las funciones existentes en el paquete de Matlab 7.0. El mapa auto-organizado fue programada en Matlab 7.0. Los listados de los programas empleados se pueden consultar en el apéndice A. El capítulo 4 muestra los resultados obtenidos a partir de la aplicación del método propuesto en el capítulo 3 así como la descripción de los diferentes casos de estudio.

Finalmente se presentan conclusiones generales, referencias y el apéndice A.

CAPÍTULO 1. REDES NEURONALES

1.1 Introducción

Las redes neuronales artificiales representan un método para implementar procesos de pensamiento parecidos al de los humanos, es decir, simular el funcionamiento del cerebro humano con la finalidad de realizar una tarea en concreto.

La estructura del cerebro humano es altamente compleja, no lineal, cooperativa y paralela; tiene la capacidad de organizar sus componentes estructurales (conocidos como neuronas), así como realizar cálculos precisos (reconocimiento de patrones, percepción y control motor, etc.). Además, el cerebro tiene la estructura y habilidad para acumular sus propias reglas de aprendizaje que usualmente son referidas como "experiencia". De hecho, la experiencia es acumulada a través del tiempo. Es así como las redes neuronales artificiales se basan en estas características para crear una semejanza para aplicaciones computacionales.

Las redes neuronales artificiales emplean una interconexión masiva de células computacionales simples referidas como neuronas o unidades de procesamiento, las cuales almacenan conocimiento experimental y lo hacen disponible para su uso. Esto asemeja al cerebro en dos aspectos [Haykin, 1994]:

- El conocimiento es adquirido por la red de su entorno a partir de un proceso de aprendizaje. Las fuerzas de conexión entre las neuronas, conocidas como pesos sinápticos, son usadas para almacenar el conocimiento adquirido.

- El mecanismo usado para desarrollar el proceso de aprendizaje es llamado algoritmo de aprendizaje, cuya función es modificar los pesos sinápticos de la red para lograr un objetivo deseado.

A lo largo de este capítulo se introducen los aspectos generales que comprenden las redes neuronales artificiales, compuestos por los elementos indispensables para la definición de una red neuronal artificial, como son estructura, topología, características de los nodos (neuronas), así como reglas de aprendizaje o entrenamiento. Esto será útil para hacer énfasis en las redes propuestas para esta tesis, los mapas auto-organizados, modelo de red neuronal no supervisada introducida por Kohonen en 1982 [Kohonen, 1995].

1.2 Generalidades

La unidad de procesamiento de información que es fundamental para la operación de una red neuronal es conocida como neurona. La figura 1.1 muestra el modelo de una neurona, la cual es la base para el diseño de redes neuronales. Se identifican tres elementos básicos del modelo neuronal [Haykin, 1994]:

- ✓ Sinapsis, caracterizadas por un peso. Específicamente, y como lo señala Haykin [Haykin, 1994], una señal x_j a la entrada de la sinapsis j conectada a la neurona k es multiplicada por su peso sináptico w_{jk}. El subíndice k se refiere a la neurona en cuestión y el j se refiere a la entrada

de la sinapsis. Contrario a las sinapsis en el cerebro, el peso sináptico de una neurona artificial puede caer en un rango que incluya valores tanto positivos como negativos.

- ✓ Sumador, para las señales de entrada.
- ✓ Función de activación, para limitar la amplitud de la salida de una neurona.

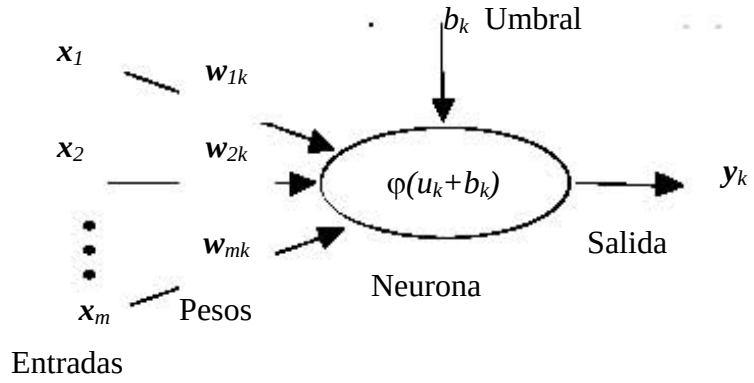

Figura 1.1. Modelo de neurona

Se puede describir una neurona k como:

$$u_k = \sum_{j=1}^{m} w_{jk} x_j \tag{1.1}$$

y

$$y_k = \varphi(u_k + b_k) \tag{1.2}$$

donde $x_1, x_2, \ldots, x_m$ son las señales de entrada; $w_{1k}, w_{2k}, \ldots, w_{mk}$ son los pesos sinápticos de la neurona k; u_k es la salida combinatoria lineal debido a las señales de entrada; b_k es el umbral; $\varphi(.)$ es la función de activación y y_k es la señal de salida de la neurona. El uso del umbral b_k tiene el efecto de afinar la transformación a la salida u_k de la combinación lineal.

La estructura general de una red neuronal consiste de una o más entradas. Existe un primer nivel de grupo de neuronas que generan una salida igual a las entradas presentadas. En este nivel (también conocido como capa de entrada) no existe ningún tipo de procesamiento [Hagan M. et al, 1997].

Además se cuenta con una o más neuronas de salida. En este nivel (también conocido como capa de salida) cada neurona de salida acepta entradas, las procesa y finalmente genera una salida.

La figura 1.2 muestra las neuronas de entrada conectadas a las neuronas de salida a través de una "caja negra", la cual es la encargada de llevar a cabo el proceso de aprendizaje. El modelo exacto de red es determinado por la naturaleza de esta caja negra, la cual puede estar formada por más capas de neuronas, que se conocen como capas ocultas o intermedias.

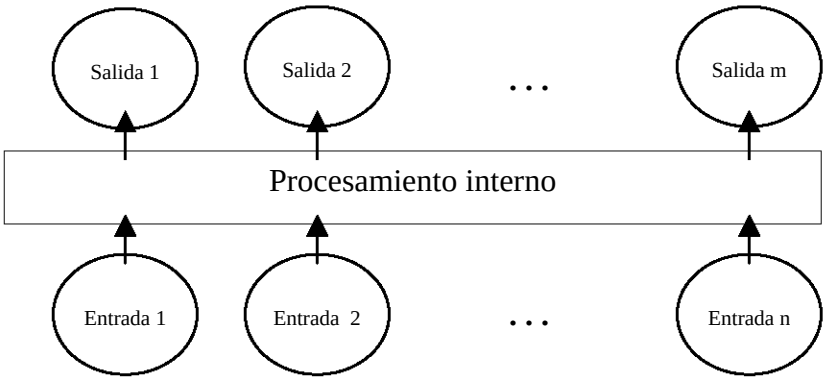

Figura 1. 2. Estructura básica de una red neuronal

El proceso mediante el cual la red neuronal será capaz de almacenar conocimiento a partir de las entradas presentadas se conoce como entrenamiento o regla de aprendizaje.

Una red neuronal puede ser entrenada de dos formas. Una de ellas es a través del entrenamiento supervisado. En este método, cada muestra (donde se hace referencia a muestra como el conjunto de entradas presentadas a la red en un tiempo dado) dentro del grupo de entrenamiento tendrá información tanto de las entradas como de las correspondientes salidas, las cuales se conocen como "salidas deseadas". Para cada muestra, se comparan las salidas obtenidas por la red al habérsele presentado las muestras de entrenamiento en un orden aleatorio, con las salidas deseadas. Posteriormente se actualizan los pesos sinápticos de forma que se reduzca una medida de error en los resultados de la red. Se llama "época" a cada vez que se presenta a la red un conjunto de muestras de entrenamiento aunado a la actualización de los pesos sinápticos. Estas épocas son repetidas hasta que la red alcanza un desempeño satisfactorio.

La segunda forma de entrenamiento es el llamado no supervisado, en el cual se cuenta con muestras de entrenamiento que contienen información solamente sobre las entradas. Aquí se asume que cada entrada representa una de varias clases, y las salidas de la red es una identificación de la clase a la cual esa entrada pertenece. El proceso de entrenamiento de la red consiste en permitir que ésta clasifique características sobresalientes del grupo de entrenamiento, y usar estas características en las diferentes clases que encontró [Haykin, 1994].

Dado que las redes propuestas para esta tesis pertenecen a la segunda forma de entrenamiento se explicará más adelante y con más detalle este proceso.

Entre los beneficios que ofrecen las redes neuronales artificiales tenemos [Haykin, 1994]:

- Estructura distribuida paralela masiva.

- Habilidad para aprender, por lo que se tiene la capacidad de generalizar. (La generalización se refiere a las salidas razonables que produce la red neuronal artificial a partir de entradas no contempladas durante el entrenamiento).

El uso de una red neuronal ofrece las siguientes propiedades y capacidades:

- No-linealidad, la cual es una propiedad importante si lo que se desea en la generación de la señal de salida es no lineal.

- Mapeo entrada-salida: Como se menciona al hablar de tipos de entrenamiento, el aprendizaje supervisado involucra la modificación de los pesos sinápticos de la red neuronal por la presentación de "muestras de entrenamiento" etiquetadas. Así la red aprende de las muestras por construcción de un mapeo entrada-salida para el problema planteado.

- Adaptatividad. Las redes neuronales tienen la capacidad de adaptar sus pesos sinápticos cuando se presentan cambios en el ambiente que las rodea. Es decir, una red neuronal puede ser fácilmente entrenada nuevamente a fin de realizar cambios menores en el funcionamiento y así adaptarse al nuevo ambiente.

- Respuesta evidente. En el contexto de la clasificación de patrones, una red neuronal puede ser diseñada para proporcionar información no solamente acerca de cual patrón seleccionar, sino además acerca de la confianza en la decisión tomada. Esta última información puede ser usada para rechazar patrones ambiguos que pudieran presentarse para mejorar la forma de clasificación de la red.

1.3 Redes no supervisadas

Las redes no supervisadas son llamadas de esta forma por referirse a su regla de aprendizaje de tipo no supervisado. Este tipo de regla de aprendizaje o entrenamiento se basa en que únicamente se debe suministrar a la red los datos de entrada para que extraiga los rasgos característicos esenciales. Esto equivale a los modelos en los que sólo hay vectores de variables independientes y buscan el agrupamiento de los patrones de entrada: análisis de conglomerados o cluster, escalas multidimensionales, etc.

Estas redes deben encontrar las características, regularidades, correlaciones o categorías que se pueden establecer entre los datos de entrada. Puesto que no hay un supervisor que indique a la red la respuesta que debe generar una entrada concreta, se puede decir que existen varias posibilidades en cuanto a la interpretación de las salidas, que dependen de su estructura y de la regla de aprendizaje empleada.

En algunos casos, la salida representa el grado de familiaridad o similitud entre las entradas y las informaciones que se le han mostrado hasta entonces (en el pasado). En otro caso podría realizar una técnica de *clustering* o establecimiento de categorías, indicando a la salida a qué categoría pertenece la información presentada en la entrada. La propia red debe encontrar las categorías apropiadas a partir de correlaciones entre la información presentada.

Finalmente, algunas redes con aprendizaje no supervisado realizan un mapeo de características (*feature mapping)*, obteniendo en las neuronas de salida una disposición geométrica que representa un mapa topográfico de las características de los datos de entrada, de tal forma que si se presentan a la red informaciones similares, siempre sean afectadas neuronas de salida próximas entre sí, en la misma zona del mapa [Haykin, 1994]. En esta categoría es en la que se clasifican los mapas auto-organizados, que se describen con mayor detalle en la siguiente sección.

1.4 Mapas Auto-organizados

Los mapas auto-organizados [Kohonen,1995] (self organizing maps, SOM), fueron introducidos en el año de 1982 por Teuvo Kohonen. Esta red emplea un tipo de aprendizaje no supervisado de tipo competitivo, de mucha utilidad si se piensa en el campo del análisis exploratorio de datos. Debido a su capacidad de análisis puede representar densidades de probabilidad y proyectar un espacio de alta dimensión sobre otro de dimensión menor.

En el aprendizaje competitivo las neuronas compiten unas con otras con el fin de llevar a cabo una tarea dada. Aquí se pretende que cuando se presente a la red un patrón de entrada de la muestra de entrenamiento, sólo una de las neuronas de salida (o un grupo de vecinas) se active. Esto es, las neuronas compiten por activarse, quedando una como neurona ganadora, mientras el resto son forzadas a sus valores de respuesta menores que dependen de su distancia a la neurona ganadora.

El modelo SOM [Jämsa-Jounela et. al, 1988] está compuesto por dos capas de neuronas. La capa de entrada (formada por N neuronas, una por cada señal de entrada) es la encargada de recibir y propagar a la capa de salida las señales de entrada procedentes del exterior. La capa de salida (formada por M neuronas) es la encargada de procesar la información y formar un mapa de rasgos. Las neuronas de la capa de salida se organizan, generalmente, en forma de mapa bidimensional como se muestra en la figura 1.3, aunque pueden usarse capas de diferentes dimensiones.

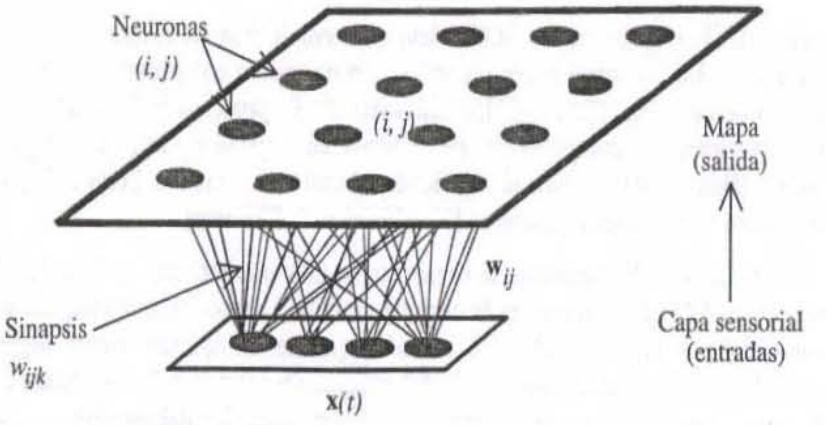

Figura 1.3. Mapa bidimensional

Cada neurona de entrada *i* está conectada a cada una de las neuronas de salida *j* a través de un peso w_{ij}. Las conexiones que se forman entre las dos capas propagan la información desde la capa de entrada hacia la capa de salida.

Entre las neuronas de la capa de salida puede decirse que existen conexiones laterales de excitación o inhibición, pues aunque no estén conectadas, cada una de estas neuronas va a tener cierta influencia sobre sus vecinas. Esto se consigue a través de un proceso de competencia entre las neuronas y de la aplicación de una función llamada de vecindad.

Los pasos para formar el mapa auto-organizado, tal como los describe Kohonen [Kohonen, 1995], son los siguientes:

1. Designar número de neuronas y forma del mapa.

2. Inicializar los pesos sinápticos. Existen tres formas de realizar este proceso:

   - Aleatoria, donde los pesos son inicializados con valores aleatorios pequeños.
   - Por muestras, donde los pesos son inicializados empleando muestras aleatorias del grupo de datos de entrada.
   - Lineal, donde los pesos son inicializados en una manera ordenada a lo largo del subespacio lineal que pasa por los dos vectores propios principales del grupo de datos de entrada. Los vectores propios pueden ser calculados usando el procedimiento Gram.-Schmidt [Hoffman et. al, 1973].

3. Se presenta un patrón *p* de entrada $\boldsymbol{x_p}$: $x_{p1}, \ldots, x_{pi}, \ldots, x_{pN}$, el cual se transmite directamente de la capa de entrada a la capa de salida, para posteriormente llevar a cabo tres procesos esenciales:

   a) Competitivo: Buscar el vector de pesos $\boldsymbol{w_j}$ que mejor empareja al vector $\boldsymbol{x_p}$. Esto se hace calculando

   $$\boldsymbol{w_j^T x_p} \tag{1.3}$$

   y seleccionar el mayor. Así se tendrá la localización donde la vecindad de neuronas excitadas será centrada, es decir, la neurona ganadora.

   El mejor emparejamiento se logra calculando el mínimo de la distancia euclidiana entre $\boldsymbol{x_p}$ y $\boldsymbol{w_j}$.

   b) Cooperativo: Sea $h_{j,i}$ la vecindad centrada sobre la neurona ganadora *i*; *j* determina el grupo de neuronas excitadas.

   Sea $d_{i,j}$ la distancia lateral entre las neuronas *i* y *j*. Entonces, $h_{j,i}$ satisface que:

   - Su valor máximo lo tiene en la neurona i para la cual $d_{j,i} = 0$.
   - La amplitud de $h_{j,i}$ decrece monótonamente con el incremento de la $d_{j,i}$.

   Una $h_{j,i}$ que satisface estos requerimientos es la función gaussiana

   $$h_{j,i}(x) = exp\,(-d_{j,i}^2 / 2\sigma^2) \tag{1.4}$$

   σ mide el grado al cual las neuronas excitadas en los alrededores de la neurona ganadora participan en el aprendizaje.

   $d_{j,i}$ se toma para el caso unidimensional como $|j - i|$ siendo un entero. Para el caso bidimensional se toma la distancia euclidiana

   $$d_{j,i}^2 = ||r_j - r_i||^2 \tag{1.5}$$

donde r_j es la posición de la neurona excitada j (coordenadas dentro del mapa); mientras que r_i es la posición de la neurona ganadora.

Una característica de SOM es que las vecindades se reducen a través del tiempo. Esto se hace haciendo σ decreciente en el tiempo

$$\sigma(n) = \sigma_0 \exp(-n/\tau_1) \qquad n = 0, 1, 2, \ldots \tag{1.6}$$

donde σ_0 es algún σ inicial por definirse y τ_1 es una constante de tiempo.

Por lo tanto, asumiendo variación en el tiempo se tiene

$$h_{j,i}(x)(n) = \exp(-d_{j,i}^2 / 2\sigma^2(n)) \quad n=0, 1, 2, \ldots \tag{1.7}$$

Así, incrementándose el tiempo n, $\sigma(n)$ decrece en un rango exponencial y la vecindad se hace más pequeña.

c) Adaptativo. Para que sea auto-organizado se requiere que el cambio del vector de pesos w_j sea con respecto al vector de entrada.

Para esto se utiliza la hipótesis de Hebbian [Hebb, 1949] adicionando un término que se llama de "olvido", de la forma $g(y_j)\mathbf{w_j}$. $g(y_j)$ es alguna función escalar positiva de la respuesta y_j, la cual debe cumplir la condición $g(y_j)=0$, entonces $y_j=0$; por lo que el cambio en el vector de pesos ($\Delta\mathbf{w_j}$) estará dado por

$$\Delta\mathbf{w_j} = \eta\, y_j\, \mathbf{x_p} - g(y_j)\mathbf{w_j} \tag{1.8}$$

donde η es el parámetro de aprendizaje, es decir, el valor que determina que tan lento ó rápido será el aprendizaje de la red.

Para satisfacer $g(y_j) = 0$ y $y_j=0$, se escoge $g(y_j)$ como una función lineal

$$g(y_j) = \eta y_j \tag{1.9}$$

Se puede simplificar haciendo $y_j = h_{j,i(x)}$.

Entonces

$$\Delta\mathbf{w_j} = \eta \cdot h_{j,i(x)} \cdot (\mathbf{x_p} - \mathbf{w_j}) \tag{1.10}$$

Tomando en cuenta la variación en el tiempo de manera discreta se tendría

$$\mathbf{w_j}(n+1) = \mathbf{w_j}(n) + \eta(n) \cdot h_{j,i(x)}(n) \cdot (\mathbf{x} - \mathbf{w_j}(n)) \tag{1.11}$$

Al paso de la presentación de los datos de entrenamiento, los pesos sinápticos tienden a seguir la distribución del vector de entrada debido a la actualización de la vecindad.

Para que el desempeño de la red sea satisfactorio al aplicar este algoritmo, es necesario tomar en cuenta algunas heurísticas, como:

- El parámetro de aprendizaje $\eta(n)$ deberá comenzar en un valor inicial η_0 para posteriormente decrecer gradualmente conforme pase el tiempo. Esto se logra haciendo:

$$\eta(n) = \eta_0 \exp(-n / \tau_2) \quad n=0, 1, 2, \ldots \tag{1.12}$$

  donde τ_2 es una constante de tiempo.

- En el proceso adaptativo existen dos fases en las que es importante escoger los parámetros empleados:

  - Autoorganización o fase de ordenamiento: Se lleva a cabo el ordenamiento topológico de los vectores de peso. Es importante saber escoger:

    a) $\eta(n)$, el cual debe ser cercano a uno y que decrezca gradualmente hasta 0.01. Esto se satisface con $\eta_0 = 0.1$ y $\tau_2 = 1000$.

    b) Tratar que $h_{j,i}(n)$ incluya casi todas las neuronas centradas sobre la ganadora i, y se reduzca conforme pase el tiempo. Para el caso bidimensional ponemos σ_0 igual al "radio" del mapa. Puede ser más de la mitad del diámetro de la red. τ_1. que es empleada en la ecuación (1.6) para el cálculo de vecindades, puede ser $1000 / \log \sigma_0$. Esta fase tomará aproximadamente 1000 iteraciones.

  - Fase de convergencia: Es necesaria para dar una cuantificación estadística exacta del espacio de entrada. Toma al menos 500 veces el número de neuronas en la red.

1.5 Conclusiones

Este capítulo abarca una introducción a las redes neuronales artificiales, particularizando en las redes SOM, o mapas auto-organizados, empleados en esta tesis; esto con la finalidad de introducir al lector a la terminología usada en el desarrollo posterior.

Aun cuando ya se mencionó el por qué se ve conveniente usar redes neuronales con aprendizaje de tipo no supervisado, el elegir los mapas auto-organizados no es discriminatorio a otro tipo de red con el mismo tipo de aprendizaje tal como las redes de tipo ART (Adaptable Resonance Theory) [Lee S. Et al.,2003] lo cual se deja como un trabajo de investigación a futuro.

CAPITULO 2. ESTIMACIÓN DE PARÁMETROS

2.1 Introducción

El objetivo de la estimación de parámetros consiste en establecer modelos matemáticos de sistemas físicos lineales y observables con base en una discretización finita.

En un sistema lineal, las variables que son de interés son las salidas, mientras las señales que pueden ser manipuladas libremente son sus entradas. El resto de las señales que influyen en las salidas pero que no pueden ser manipuladas son llamadas perturbaciones, denotadas por el símbolo $\varepsilon(t)$. La figura 2.1 muestra un esquema del sistema dinámico.

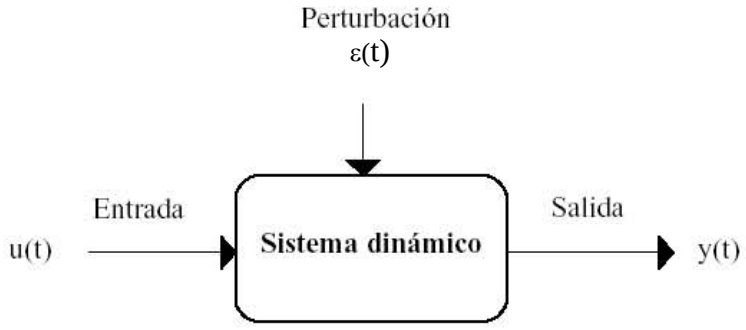

Figura 2.1. Representación de un sistema dinámico

En ocasiones es conveniente trabajar con el modelo del sistema físico a estudiar, ya que esto nos permitirá conocer el comportamiento del sistema en ciertas condiciones y ante determinadas entradas.

Existen dos tipos de modelos [Chi-Tsong, 1999]:

a) No paramétricos, para los cuales es suficiente dar una representación a través de un gráfico o tabla que describa sus propiedades dinámicas mediante un número no finito de parámetros.

b) Paramétricos o matemáticos, en donde se describen las relaciones entre las variables del sistema mediante expresiones matemáticas como pueden ser ecuaciones diferenciales (para sistemas continuos) o en diferencias (para sistemas discretos). En función del tipo de sistema y de la representación matemática utilizada, los sistemas pueden ser:

   - Determinísticos o estocásticos. Se dice que un modelo es determinístico cuando expresa la relación entre entradas y salidas mediante una ecuación exacta. Por el contrario, un modelo es estocástico si posee un cierto grado de incertidumbre. Estos últimos se definen mediante conceptos probabilísticos o estadísticos.

   - Dinámicos o estáticos. Un sistema es estático cuando la salida depende únicamente de la entrada en ese mismo instante. En estos sistemas existe una relación directa entre entrada y salida, independiente del tiempo. Un sistema dinámico es aquel en el que las salidas evolucionan con el tiempo tras la aplicación de una determinada entrada. En

estos últimos, para conocer el valor actual de la salida es necesario conocer el tiempo transcurrido desde la aplicación de la entrada.

- Continuos o discretos. Los sistemas continuos trabajan con señales continuas, y se caracterizan mediante ecuaciones diferenciales. Los sistemas discretos trabajan con señales muestreadas, y quedan descritos mediante ecuaciones en diferencias.

Todo modelo matemático o paramétrico de un sistema consta de una o varias ecuaciones que relacionan las entradas y salidas (en los modelos dinámicos la variable *t* (tiempo) juega también un papel importante).

El modelado matemático de sistemas físicos es posible a través de un conjunto de teorías, métodos, y algoritmos que permiten, a partir de datos experimentales de las entradas y salidas, obtener el modelo matemático del sistema, el cual es equivalente al sistema bajo un cierto criterio.

La estimación de parámetros para sistemas físicos se lleva a cabo excitando al sistema con algún tipo de señal rica en frecuencias y procesando la entrada y salida en un determinado intervalo de tiempo. Un aspecto importante es la selección correcta de la señal de excitación y el número de muestras. Posteriormente se propone un modelo, el cual debe permitir que los cálculos de los parámetros sean simples.

En particular, este trabajo se enfoca en los sistemas dinámicos, describiendo así uno de los métodos más empleado de estimación de parámetros, el método de mínimos cuadrados.

2.2 Mínimos cuadrados para la estimación de parámetros

Se menciona primero la aplicación del método a sistemas estáticos, enfocando por el momento a modelos lineales [Getler,1998].

Considérese un sistema con una salida $y(t)$ y un grupo de entradas $u_1(t)\ u_2(t)\ ...\ u_k(t)$, conectadas de forma lineal

$$y(t) = a_1\, u_1(t) + a_2 u_2(t) + ... + a_k\, u_k(t) + \varepsilon(t) \tag{2.1}$$

donde $\varepsilon(t)$ es un término que indica una señal de perturbación.

Si se introduce la notación vectorial

$$\varphi'(t) = \boldsymbol{u}'(t) = [u_1(t)\ u_2(t)\ ...\ u_k(t)] \qquad \pi = [a_1\ a_2\ \ ...\ \ a_k]' \tag{2.2}$$

donde $\varphi(t) = \boldsymbol{u}(t)$ es llamado el vector de regresión y π es el vector parámetro. Se puede escribir el modelo de la figura 2.1 como

$$y(t) = \varphi'(t)\,\pi + \varepsilon(t) \tag{2.3}$$

Aquí, las entradas $\varphi'(t)$ y la salida $y(t)$ pueden ser medidas; lo que no es posible es medir la señal de perturbación. Además los parámetros π son desconocidos. Es posible predecir las salidas basándonos en las entradas como

$$\hat{y}(t) = \varphi'(t)\hat{\pi} \tag{2.4}$$

donde $\hat{\pi}$ es un estimado del vector de parámetros. Se encontrará este estimado tal que la suma de los cuadrados de los errores de predicción, sobre una serie K de observaciones

$$J = J(t, K) = \sum_{k=0}^{K-1} [\hat{y}(t-k) - y(t-k)]^2 \tag{2.5}$$

sea mínima. A esto es lo que se conoce como el algoritmo de mínimos cuadrados, el cual cumple con dicho propósito. Es importante señalar que esta aproximación depende del tipo de sistema a observar el cual tiene que ser lineal en los parámetros por el hecho de la suma finita de sus diferencias.

Para derivar al algoritmo básico de mínimos cuadrados, se sustituye 2.4 en 2.5, de forma que se obtiene:

$$J = \sum_{k=0}^{K-1} [\varphi'(t-k)\,\hat{\pi} - y(t-k)]^2 \tag{2.6}$$

Diferenciando con respecto a $\hat{\pi}$

$$\frac{dJ}{d\hat{\pi}} = \sum_{k=0}^{K-1} 2\,\varphi(t-k)\,[\varphi'(t-k)\,\hat{\pi} - y(t-k)] \tag{2.7}$$

Con el conocimiento previo de que en el óptimo la derivada es cero se tiene que

$$\sum_{k=0}^{K-1} \varphi(t-k)\,\varphi'(t-k)\hat{\pi} = \sum_{k=0}^{K-1} \varphi(t-k)\,y(t-k) \tag{2.8}$$

La solución de 2.8 es un mínimo porque la segunda derivada del desarrollo de J,

$$\frac{d^2J}{d\hat{\pi}\,d\hat{\pi}'} = \sum_{k=0}^{K-1} 2\,\varphi(t-k)\,\varphi'(t-k) \tag{2.9}$$

es una matriz definida positiva [Getler,1998].

La solución de la ecuación 2.8 da la estimación de los parámetros como

$$\hat{\pi} = [\sum_{k=0}^{K-1} \varphi(t-k)\, \varphi'(t-k)]^{-1} [\sum_{k=0}^{K-1} \varphi(t-k)\, y(t-k)] \qquad (2.10)$$

La ecuación 2.10 es la fundamental en la estimación de parámetros por el algoritmo de mínimos cuadrados. La solución a esta ecuación existe si la matriz resultante del primer corchete tiene rango completo. La aplicación de este método en modelos no lineales no es tan simple como para el caso anterior; sin embargo podemos aplicarlo de una forma sencilla en modelos no lineales cuyos parámetros son lineales.

La forma general de un modelo no lineal el cual es lineal en sus parámetros es

$$y(t) = a_1 f_1[\mathbf{u}(t)] + a_2 f_2[\mathbf{u}(t)] + \ldots + a_\rho f_\rho[\mathbf{u}(t)] + \varepsilon(t) \qquad (2.11)$$

donde $f_1[\mathbf{u}(t)], \ldots, f_\rho[\mathbf{u}(t)]$ son funciones no lineales conocidas del vector de entrada $\mathbf{u}(t)$, y $a_1 \ldots a_\rho$ son parámetros no conocidos. Así, los vectores de regresión y de parámetros son definidos como:

$$\varphi'(t) = [f_1[\mathbf{u}(t)] \ldots f_\rho[\mathbf{u}(t)]] \quad \pi = [a_1 \ldots a_\rho]' \qquad (2.12)$$

De este modo, se puede predecir la salida tal como en el caso lineal como lo muestra la ecuación 2.4. Por lo que se estiman los parámetros usando la ecuación 2.10 del método de mínimos cuadrados.

Como se mencionó, para mostrar la utilidad de este método, este trabajo se enfoca en los sistemas auto regresivos de promedio móvil (ARMA, AutoRegressive Moving Average systems, en inglés), los cuales son una clase de modelos dinámicos lineales discretos.

Existen tres tipos de sistemas ARMA:

1. Una entrada- una salida: Considere el sistema

$$y(t) = \frac{g(\phi)}{h(\phi)} u(t) + \gamma(t) \qquad (2.13)$$

donde $y(t)$ es una salida escalar, $u(t)$ es una entrada escalar y $\gamma(t)$ es un término que representa una señal de perturbación la cual actúa sobre la salida del sistema. $g(\phi)/ h(\phi)$ representa la función de transferencia polinomial, ya que se tiene al sistema de estudio representado por ecuaciones diferenciales ordinarias simultáneas. $g(\phi)$ y $h(\phi)$ están dadas por:

$$g(\phi) = g^0 + g^1 \phi^{-1} + \ldots + g^\nu \phi^{-\nu}$$
$$h(\phi) = 1 + h^1 \phi^{-1} + \ldots + h^\nu \phi^{-\nu} \qquad \nu=[0..t] \qquad (2.14)$$

Se busca convertir la ecuación 2.13 en una ecuación en forma diferencial recursiva. Para esto se multiplica de forma cruzada la ecuación 2.13 por $h(\phi)$, obteniendo:

$$h(\phi)\, y(t) = g(\phi)\, u(t) + h(\phi)\, \gamma(t) \qquad (2.15)$$

Para así expresar $y(t)$ de la forma

$$y(t) = g(\phi)\, u(t) - [h(\phi) - 1]\, y(t) + h(\phi)\, \gamma(t) \tag{2.16}$$

lo que se puede escribir como

$$y(t) = \varphi'(t)\, \pi + \varepsilon(t) \tag{2.17}$$

donde

$$\varphi'(t) = [u_1(t), u_1(t\text{-}1), \ldots, u_1(t\text{-}\nu)\ \text{-}y(t\text{-}1) \ldots \text{-}y(t\text{-}\nu)] \tag{2.18}$$

$$\pi = [g^0\ g^1 \ldots\ g^\nu\ \ h^1 \ldots h^\nu\ ]' \tag{2.19}$$

$$\varepsilon(t) = h(\phi)\gamma(t) \tag{2.20}$$

que presenta la misma estructura que la ecuación 2.3, permitiendo así aplicar la ecuación 2.10 del método de mínimos cuadrados para la obtención de los parámetros, pensando en la predicción como se muestra en la ecuación 2.4.

Cabe señalar que en las ecuaciones 2.18, 2.19 y 2.20 se tienen diferencias importantes con respecto a lo mencionado anteriormente:

- El vector de regresión $\varphi(t)$ contiene valores tanto presentes como pasados.
- El vector de regresión contiene salidas pasadas, además de las entradas.
- La ecuación de error del sistema, $\varepsilon(t)$, no es idéntica a la señal de perturbación de salida $\gamma(t)$ como en los casos anteriores.

2. Múltiples entradas – una salida: Considere el sistema

$$y(t) = \frac{g_1(\phi)}{h(\phi)}\, u_1(t) + \frac{g_2(\phi)}{h(\phi)}\, u_2(t) + \ldots + \frac{g_k(\phi)}{h(\phi)}\, u_k(t) + \gamma(t) \tag{2.21}$$

donde

$$g_j(\phi) = g_j^{\,0} + g_j^{\,1}\, \phi^{-1} + \ldots + g_j^{\,\nu}\, \phi^{-\nu} \quad j = 1 \ldots k \tag{2.22}$$

y $h(\phi)$ es el común denominador de las funciones de transferencia. Esto da la expresión recursiva para $y(t)$

$$y(t) = g_1(\phi)\, u_1(t) + g_2(\phi)\, u_2(t) + \ldots + g_k(\phi)\, u_k(t) - [h(\phi) - 1]\, y(t) + h(\phi)\, \gamma(t) \tag{2.23}$$

que puede ser descrito como

$$y(t) = \varphi'(t)\, \pi + \varepsilon(t) \tag{2.24}$$

con

$$\varphi'(t) = [u_1(t) \ldots u_1(t-\nu) \ldots u_k(t) \ldots u_k(t-\nu)\ -y(t-1) \ldots -y(t-\nu)] \tag{2.25}$$

$$\pi = [g_1^{\,0} \ \ldots \ g_1^{\,\nu} \ \ldots g_k^{\,0} \ \ldots \ g_k^{\,\nu} \ \ h^1 \ldots h^\nu \ ]' \tag{2.26}$$

$$\varepsilon(t) = h(\phi)\gamma(t) \tag{2.27}$$

De ésta forma se puede aplicar el método de mínimos cuadrados.

3. Múltiples entradas – múltiples salidas: En este tipo de sistemas no es tan fácil como en los anteriores aplicar el método de mínimos cuadrados, se modela de la forma

$$y_i(t) = \sum_{j=1}^{k} \frac{g_{ij}(\phi)}{h_i(\phi)} u_j(t) + \gamma_i(t) \qquad i=1 \ldots \mu \tag{2.28}$$

donde el denominador $h_i(\phi)$ se asume que es diferente en cada salida de un subsistema, lo que da por entendido que el sistema puede ser identificado como un conjunto de μ salidas de subsistemas no relacionados entre sí. Sin embargo, en un sistema real de múltiples salidas, los denominadores de los subsistemas son, al menos, parcialmente idénticos, lo que nos lleva a poder representar la ecuación anterior de la forma

$$y_i(t) = \sum_{j=1}^{k} \frac{\tilde{g}_{ij}(\phi)}{\tilde{h}(\phi)} u_j(t) + \nu_i(t) \quad i=1 \ldots \mu \tag{2.29}$$

donde $\tilde{h}(\phi)$ es el mínimo común múltiplo de $h_i(\phi)$. Por medio de una formulación recursiva esto nos da

$$y_i(t) = \sum_{j=1}^{k} \tilde{g}_{ij}(\phi)\, u_j(t) - [\tilde{h}(\phi) - 1]\, y_i(t) + \tilde{h}(\phi)\, \gamma_i(t) \qquad i = 1 \ldots \mu \tag{2.30}$$

Definiendo un vector de parámetros en común

$$\pi = [[\tilde{g}_{11}]' \ \ldots \ [\tilde{g}_{1k}]' \ \ldots [\tilde{g}_{\mu 1}]' \ \ldots \ [\tilde{g}_{\mu k}]' \ \ [\tilde{h}]']' \tag{2.31}$$

donde

$$[\tilde{g}_{ij}]' = [\tilde{g}_{ij}^{\,0}\ , \tilde{g}_{ij}^{\,1} \ldots \tilde{g}_{ij}^{\,\nu}] \qquad i = 1 \ldots \mu\ , j = \ 1 \ldots k \tag{2.32}$$

$$[\tilde{h}]' = [\tilde{h}^1 \ldots \tilde{h}^\nu\ ] \tag{2.33}$$

Entonces, la ecuación 2.30 puede ser escrita como

$$y_i(t) = \psi_i'(t)\, \pi + \varepsilon_i(t) \qquad i = 1 \ldots \mu \tag{2.34}$$

donde

$$\psi_i'(t) = [\underbrace{\mathbf{0}' \dots \mathbf{0}'}_{1} \dots \underbrace{\mathbf{0}' \dots \mathbf{0}'}_{i-1} \underbrace{u_1(t) \dots u_1(t-\nu) \dots u_k(t) \dots u_k(t-\nu)}_{i}$$

$$\underbrace{\mathbf{0}' \dots \mathbf{0}'}_{i+1} \dots \underbrace{\mathbf{0}' \dots \mathbf{0}'}_{\mu} - y_i(t-1) \dots - y_i(t-\nu)] \qquad i = 1 \dots \mu \qquad (2.35)$$

(donde **0**' denotan vectores renglón de ceros) y

$$\varepsilon_i(t) = h(\phi)\gamma_i(t) \quad i = 1 \dots \mu \qquad (2.36)$$

Escribiendo la predicción del vector de salida $\mathbf{y}(t) = [y_1(t) \dots y_\mu(t)]'$ como

$$\hat{y}(t) = \Psi'(t)\hat{\pi} \qquad (2.37)$$

donde

$$\Psi'(t) = \begin{pmatrix} \psi_1'(t) \\ \psi_2'(t) \\ \cdot \\ \cdot \\ \cdot \\ \psi_\mu'(t) \end{pmatrix} \qquad (2.38)$$

Y se define el índice del desarrollo de mínimos cuadrados como

$$J = J(t, K) = \sum_{k=0}^{K-1} \sum_{i=1}^{\mu} [\hat{y}_i(t-k) - y_i(t-k)]^2 = \sum_{k=0}^{K-1} [\hat{y}(t-k) - \mathbf{y}'(t-k)]' [\hat{y}(t-k)\, \mathbf{y}(t-k)] \qquad (2.39)$$

Sustituyendo la ecuación 2.37 en la anterior, se obtiene

$$J = \sum_{k=0}^{K-1} [\hat{\pi}' \psi(t-k) \psi'(t-k) \hat{\pi} - 2\, \mathbf{y}'(t-k) \psi'(t-k) \hat{\pi} + \mathbf{y}'(t-k)\, \mathbf{y}(t-k)] \qquad (2.40)$$

que al derivar da

$$\frac{dJ}{d\hat{\pi}} = \sum_{k=0}^{K-1} 2 [\psi(t-k) \psi'(t-k) \hat{\pi} - \psi(t-k)\, y(t-k)] \qquad (2.41)$$

De aquí, el parámetro de estimación será

$$\hat{\pi} = [\sum_{k=0}^{K-1} \psi(t-k) \psi'(t-k)]^{-1} [\sum_{k=0}^{K-1} \psi(t-k)\, \mathbf{y}(t-k)] = [\Phi'\Phi]^{-1}\Phi'\, \mathbf{Y} \qquad (2.42)$$

donde

$$\Psi'(t) = \begin{pmatrix} \psi'(t) \\ \psi'(t-1) \\ \cdot \\ \cdot \\ \cdot \\ \psi'(t-K+1) \end{pmatrix} \tag{2.43}$$

$$\boldsymbol{Y} = \begin{pmatrix} \boldsymbol{y}(t) \\ \boldsymbol{y}(t-1) \\ \cdot \\ \cdot \\ \cdot \\ \boldsymbol{y}(t-K+1) \end{pmatrix} \tag{2.44}$$

2.3 Conclusiones

En este capítulo se ha dado la teoría fundamental del algoritmo de mínimos cuadrados que lleva a cabo la estimación de parámetros del sistema. Ésta nos dice: “Los parámetros desconocidos de un modelo se deben elegir de modo que la suma de los cuadrados de las diferencias entre los valores observados realmente y los estimados, multiplicada por números que midan el grado de precisión, sea un mínimo” [Ljung, 1987].

En los capítulos posteriores se mostrará su aplicación para presentar el modelado matemático del sistema dinámico a emplearse en el entrenamiento de la red neuronal propuesta.

Cabe señalar que aún cuando se han presentado la aplicación del método de mínimos cuadrados para los tres tipos principales de modelos ARMA, en esta tesis se emplea un sistema dinámico ARMA de una entrada y una salida, por lo que se aplican las ecuaciones correspondientes a dicho apartado.

Es importante resaltar dentro de este capítulo que el empleo del método de mínimos cuadrados dentro de esta tesis para la estimación de parámetros no es limitativo, existiendo otras técnicas para la estimación de parámetros tales como los algoritmos genéticos o alguna otra técnica de optimización.

CAPITULO 3. DISEÑO DEL MODELO DE DETECCIÓN DE FALLAS

3.1 Introducción

En este capítulo se presenta la descripción del método propuesto para la detección de fallas usando mapas auto-organizados y la estimación de parámetros como representación de la dinámica del sistema.

El objetivo de este proyecto es clasificar fallas en línea con base en la estimación de parámetros de manera local. Esto es, el caso de estudio es observado durante una ventana de tiempo, con lo que se produce una representación acotada del sistema. El algoritmo propuesto en esta tesis consta de dos etapas, la primera encargada de producir un modelo local paramétrico del comportamiento del caso de estudio y la segunda se encarga de clasificar el cómputo de parámetros propuestos en la primera. Este algoritmo requiere de dos escenarios, fuera de línea y en línea. Durante el primer escenario se entrena a la red neuronal con diversas respuestas de modelos ARMAX producidos por la primera etapa. Para el segundo escenario se usa al mapa auto-organizado para clasificar la respuesta local de la primer etapa. Durante el segundo escenario si la respuesta no es satisfactoria, el mapa tendrá que ser reentrenado como objetivo posterior al proceso actual.

El algoritmo propuesto busca clasificar la conducta del sistema considerando una respuesta paramétrica con lo cual se pueda tener una transición suave entre condiciones no conocidas por el diseñador.

El método propuesto se expresa gráficamente en la figura 3.1.

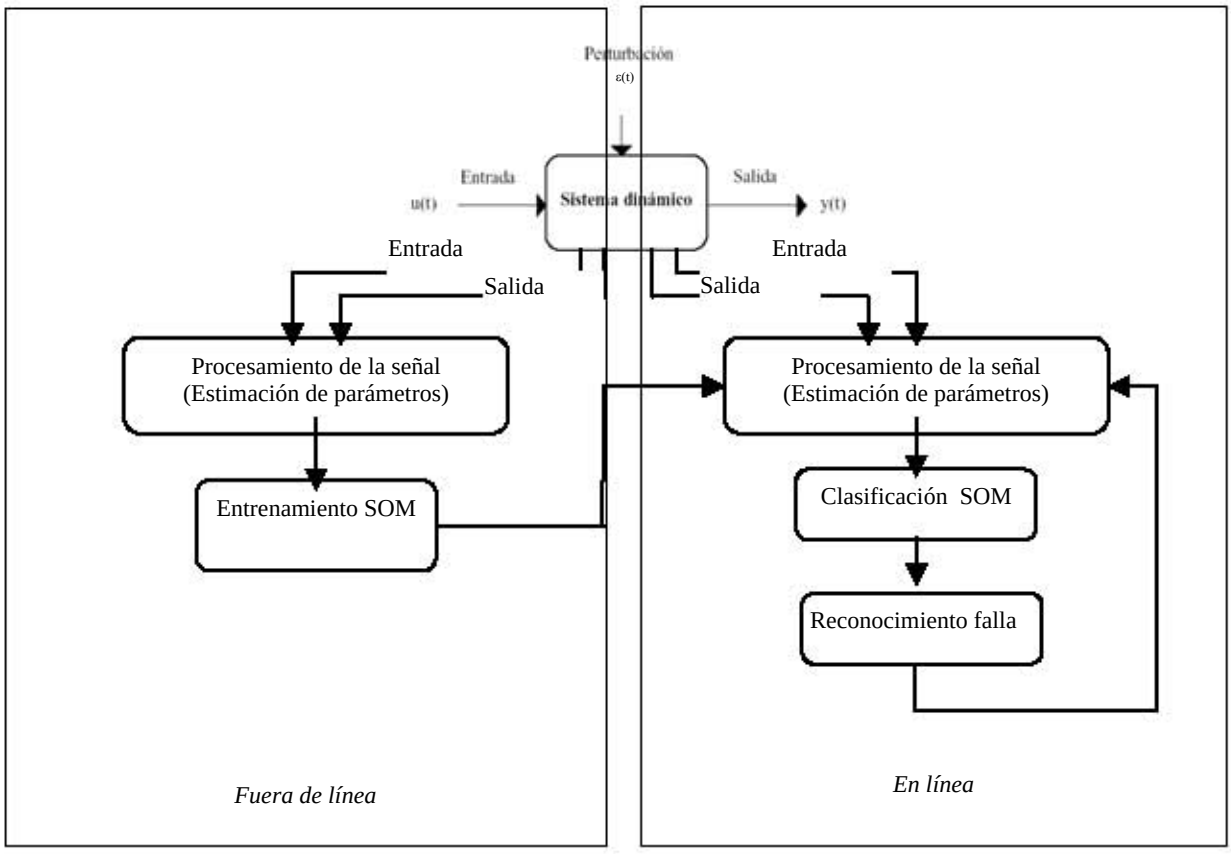

Figura 3.1. Modelo de detección de fallas

La red neuronal es entrenada fuera de línea con las condiciones de operación normales del sistema dinámico durante una ventana de tiempo determinada. Para esto, se lleva previamente el procesamiento de la señal por medio de la estimación de parámetros a través del algoritmo de mínimos cuadrados. Una vez obtenidos los parámetros necesarios para la descripción del sistema, éstos son presentados a la red como los patrones para el aprendizaje, previa normalización de los mismos, lo cual es requerido por este tipo de redes neuronales para un mejor desempeño [Haykin, 1994]. Como parte de la propuesta de tesis, la ley de aprendizaje original de la red [Kohonen, 1995] fue modificada a fin de que el mapa auto-organizado reflejara en su topología de forma adecuada el modelo matemático del sistema.

En la etapa en línea, se presenta a la red, previo procesamiento de la señal (estimación de parámetros) y normalización de los datos, el vector de parámetros correspondiente a las entradas y salidas del sistema en diferentes condiciones, ya sea libre o en presencia de fallas, para una determinada ventana de tiempo. La clasificación se lleva a cabo calculando la distancia euclidiana entre los vectores de parámetros y los pesos de la red, los cuales se sabe por el entrenamiento que contienen la estimación de parámetros del modelo en condiciones normales de operación del sistema dinámico. Es aquí donde se compara el modelo aprendido por la red con el correspondiente a la ventana actual para realizar la detección de posibles fallas en el funcionamiento del sistema.

La figura 3.2 muestra en forma sintética lo expresado anteriormente.

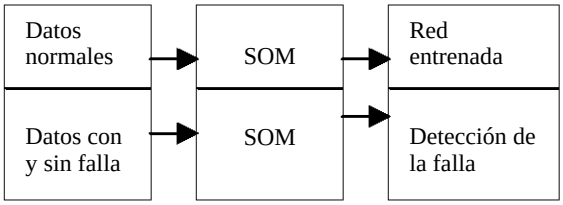

Figura 3.2.Proceso de monitoreo

En la siguiente sección se da una explicación más detallada del proceso.

3.2 Descripción del método propuesto

En esta sección se describe la obtención de datos del sistema dinámico, el procesamiento de los mismos y el desarrollo de la red neuronal para la detección de fallas.

Los datos del sistema dinámico empleado para explicar los resultados de esta tesis provienen directamente del ambiente que Simulink de Matlab 7.0 ofrece [Hunt et al, 2001].

Una vez obtenidos los datos, estos son empleados para llevar a cabo el modelado paramétrico del sistema. Para esto se aplica el algoritmo de mínimos cuadrados, para obtener dichos parámetros. Para la estimación, se empleó la función llamada **rarmax**, la cual estima los parámetros de forma recursiva para los modelos de tipo ARMA. La sintaxis de la función empleada es

[thm,yhat] = rarmax(z,nn,adm,adg)

Los datos de entrada y salida están contenidos en la variable *z*, la cual es una matriz de la forma *z = [y u]* donde *y*, que es la salida, y *u*, que es la entrada son vectores columna. El modelo esta dado por

$$A(q)y(t) = B(q)u(t-nk) + C(q)e(t) \tag{3.1}$$

Donde *t* es el tiempo, *q* es el operador de retraso y *nk* es la variable de retraso.

Los ordenes del modelo están contenidos en *nn*, por lo que *nn* = [*na nb nc nk*]. Específicamente:

$$na : A(q) = 1 + a_1 q^{-1} + ... + a_{na} q^{-na} \tag{3.2}$$

$$nb : B(q) = b_1 + b_2 q^{-1} + \ldots + b_{nb} q^{-nb+1} \tag{3.3}$$

$$nc : C(q) = 1 + c_1 q^{-1} + ... + c_{nc} q^{-nc} \tag{3.4}$$

En el caso de estudio que se describirá mas adelante, se emplea nc=0, ya que no se cuenta con una medida de error en especifico; nk=1; y los valores de na y nb son probados por medio de diferentes simulaciones (que se mostraran en el capitulo siguiente) a fin de encontrar aquel valor que ayude a expresar de forma más precisa el modelo matemático el sistema.

Para completar los argumentos necesarios en la función **rarmax** que calcula la estimación de parámetros, hace falta explicar los argumentos *adm* y *adg*, los cuales seleccionan el algoritmo actual a emplearse para dicho proceso. Estos pueden ser elegidos dentro de las siguientes opciones:

- Con *adm = 'ff'* y *adg = lam* se emplea el algoritmo con un factor de olvido, siendo éste la variable *lam*. Esto es referido usualmente como el algoritmo recursivo de mínimos cuadrados con factor de olvido [Getler, 1998] (RLS).
- Con *adm ='ug'* y *adg = gam*, se emplea el algoritmo del gradiente no normalizado.
- De forma similar, *adm ='ng'* y *adg = gam* da el algoritmo del gradiente normalizado o mínimos cuadrados normalizados (NLMS) .
- Con *adm ='kf'* y *adg = R1*, se emplea el algoritmo basado en el Filtro Kalman.

Para el desarrollo de esta tesis se seleccionó *adm = 'ff' y adg = 0.98*.

La estimación de parámetros obtenida por la función [thm,yhat] = rarmax(z,nn,adm,adg) es dada en la matriz *thm*. El k-ésimo renglón contiene los parámetros asociados con el tiempo *k*. Cada renglón de *thm* contiene los parámetros estimados en el siguiente orden:

$$thm(k,:) = [a_1, a_2, ..., a_{na}, b_1, ..., b_{nb}, c_1, ..., c_{nc}] \tag{3.5}$$

yhat es el valor predicho para la salida, de acuerdo al modelo actual, es decir, el renglón *k* de *yhat* contiene el valor predicho de *y(k)* basado en los datos pasados.

Una vez obtenida la matriz *thm* para una ventana (intervalo) de tiempo dada, se toma la columna que dio una mejor aproximación al modelo de dicha matriz como patrón de entrenamiento. Este proceso se repite para las ventanas de tiempo restantes a fin de obtener una matriz con vectores de parámetros representativos que nos ayuden en el mejor aprendizaje de la red, sabiendo de antemano que el tomar todos los vectores de parámetros satura a la red disminuyendo notablemente su desempeño. La matriz

de vectores de parámetros obtenida es normalizada empleando la fórmula para cada columna de la matriz:

$$| \textit{vector de parámetros} | / (\sum_{i=1}^{n} \textit{(elementos en el vector parámetros)}^2)^{1/2} \qquad (3.6)$$

donde *n* es el número de parámetros en el vector dado. A esta nueva matriz se le llama *Vg*.

Posteriormente se inicia el entrenamiento de la red neuronal. Los patrones de entrenamiento empleados son los parámetros estimados para una ventana de tiempo que contiene los datos del sistema dinámico en condiciones normales de operación, es decir, sin fallas. Esto se lleva a cabo por medio del programa escrito para Matlab 7.0 que se muestra en el apéndice A.

Se determina un mapa auto-organizado el cual será rectangular (bidimensional). A fin de encontrar la red neuronal que proporcione el mejor desempeño, se realizó una serie de pruebas tanto en la parte de estimación de parámetros como en el entrenamiento de la red originando diferentes redes neuronales con diferentes variables contempladas, tales como: valor de na y nb (que se consideran iguales por facilidad de diseño) para el cálculo de los vectores de parámetros, tamaño de la ventana de tiempo , tamaño de la red neuronal, número de épocas de entrenamiento, valor del factor de aprendizaje η y valor del factor de vecindad σ. La siguiente tabla muestra el rango de valores entre los cuales se considero probar estas variables. Cabe resaltar que debido a que el espacio de valores contemplados es muy grande, no se realizaron todas las pruebas posibles.

na y nb	***Tamaño de la Ventana***	***Tamaño de la red***	***Épocas***	η	σ
De 3 a 10	De 5000 a 20000	De 3*3 hasta 8*8	De 100 a 200	De 0.01 a 0.2	De 1.0 a 3.0

Tabla 3.1. Rangos para las variables necesarias en la estimación de parametros y entrenamiento de la red

Una vez determinados los valores apropiados (lo cual se mostrará en el siguiente capítulo), se continua con el entrenamiento de la red. Se asignan pesos a la red de forma aleatoria. Se establecen magnitudes para los pesos dentro del intervalo [0, 0.2], esto basado en que al asignar pesos pequeños la red tendrá un mejor desempeño en el proceso de aprendizaje [Kohonen, 1995].

Posteriormente se inicia el entrenamiento de la red. Para cada época se realizarán los siguientes pasos:

a. Se escoge un patrón de entrenamiento de forma aleatoria.

b. Se calcula la distancia euclidiana entre el patrón de entrada de la matriz *Vg* y cada neurona de la forma

$$Distancia = ||\boldsymbol{Vg}(i) - \boldsymbol{w}_{ji}|| \qquad (3.7)$$

Donde w_{ji} representa los pesos de la neurona *j* a cada uno de los elementos del vector ***Vg,*** (i=1.. número de vectores parámetro en la matriz *Vg*).

c. Se escoge la neurona con la distancia mínima y se declara ganadora.

d. Una vez seleccionada la neurona ganadora, se buscan las coordenadas de ésta dentro del mapa, para así calcular la distancia lateral entre ésta y las neuronas restantes. Esto se hace empleando

$$d_{j,i}^{\ 2} = ||r_j - r_i||^2 \tag{3.8}$$

e. Se calculan las vecindades alrededor de la neurona ganadora por medio de

$$h_{j,i}(x) = exp\ (-\ d_{j,i}^{\ 2} / 2\sigma^2) \tag{3.9}$$

f. Se actualizan los pesos sinápticos de la red empleando como ley de aprendizaje

$$w_{ij}^{\ N} = (w_{ij}^{\ O} * (1- \eta)) + (\eta * (h_{i,j} * \mathbf{Vg})) \tag{3.10}$$

donde $w_{ij}^{\ N}$ es la matriz de pesos actualizada, $w_{ij}^{\ O}$ es la matriz de pesos anterior y η es el factor de aprendizaje.

g. Se repite los incisos del (a) al (g) para los diferentes patrones de entrenamiento que forman parte de la matriz *Vg*.

La ejecución del algoritmo presentado se repite para un número de épocas determinado.

Una vez efectuado el entrenamiento de la red, se procede a la etapa en línea, en la que se efectuará la detección de posibles fallas en la operación del sistema dinámico.

Para esto se emplean los vectores de parámetros correspondientes a la nueva ventana de tiempo a analizar.

Previo a la presentación de los vectores de parámetros, se normalizan los mismos siguiendo la ecuación 3.6.

Para realizar el proceso de clasificación de la fallas (es decir, la detección de éstas), se calcula la distancia euclidiana entre la matriz de pesos obtenida de la red (que se sabe contiene el conocimiento de los vectores de parámetros) y los vectores de parámetros para una ventana de tiempo determinada. Esto lleva a obtener la distancia mínima entre lo aprendido por la red y lo presentado actualmente; esto es:

$$Distancia = ||\mathbf{Vg}_{\mathbf{actual}}(i) - \mathbf{w}_{ji}|| \tag{3.11}$$

Se escoge la neurona con la distancia mínima y se declara ganadora. Se detecta una falla cuando se activa una neurona diferente a la que se activo en un ambiente fuera de fallas. Esto crea las regiones necesarias para identificar diferentes escenarios [Alhoniemi, 2002].

3.3 Conclusiones

El presente capítulo muestra la definición del algoritmo a partir de la integración del mapa auto-organizado y la estimación de parámetros para el modelado matemático del sistema, lo cual lleva a cabo la detección de fallas.

El uso de redes neuronales no supervisadas aunado a la dinámica del sistema se presenta como alternativa eficaz para la detección de fallas en el funcionamiento del sistema.

CAPITULO 4. RESULTADOS

4.1 Introducción

El objetivo de este capítulo es mostrar los resultados obtenidos en la aplicación del método presentado para un caso de estudio en específico y así evaluar el modelo de detección de fallas propuesto.

La sección 4.2 muestra los resultados para el sistema dinámico evaluado, el cual se obtuvo por medio de una simulación en la herramienta de Matlab 7.0 Simulink, la cual nos permite la construcción de una representación por medio de diagramas de bloques de un proceso. Así mismo se explicará el sistema y las condiciones del mismo.

4.2 Caso de estudio: Simulación de un Sistema de Primer Orden.

4.2.1 Descripción general del modelo

En esta sección se presenta el caso de estudio simulado por medio de la herramienta Simulink, el cual forma parte de Matlab 7.0. La figura 4.1 muestra el sistema desarrollado en dicha herramienta

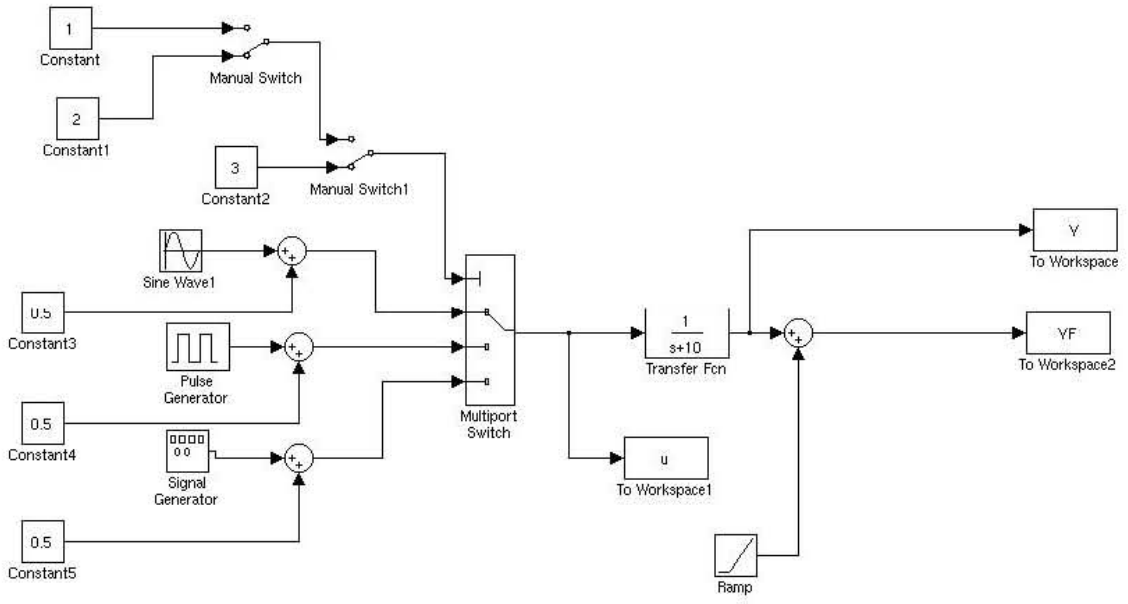

Figura 4.1 Sistema Dinámico simulado

El modelo representado es un sistema ARMAX de primer orden de una entrada y una salida, donde se representa en primer lugar una señal seno, posteriormente una señal de tipo escalón para finalizar con una señal de tipo aleatorio. Esto se hace de forma manual a fin de tener las tres señales en un periodo de tiempo determinado. Cada señal pasa por una función de transferencia la cual nos dará como salida la misma función a una escala de 1/10. Como se ve, el sistema es simple buscando tener una herramienta útil para el análisis de los resultados al aplicar el modelo de detección de fallas. En este sistema se asegura no existe ruido , lo cual afectaría de forma significativa la evaluación de los mismos. En este experimento no se hace un análisis en corrimiento de las frecuencias dado que solo se intenta tener una respuesta aproximada de la herramienta durante fallas observables.

4.2.2 Aplicación del modelo de detección de falla

El conjunto de datos requeridos para el escenario fuera de línea requieren una respuesta rica en frecuencia con el objeto de entrenar de manera completa al algoritmo propuesto. Este tipo de datos estándar son comunes en la prueba de redes neuronales tal como se presenta en [Narendra et. al., 1990]. Como se ha mencionado anteriormente, el algoritmo requiere tomar una ventana mínima de datos, que para cada prueba se muestra en su respectiva figura. Dicha ventana no tiene corrimiento contiguo con sus predecesores.

La figura 4.2 muestra las gráficas de los datos obtenidos del sistema correspondiente a la entrada y salida del sistema después de la ejecución de la simulación en condiciones normales de operación.

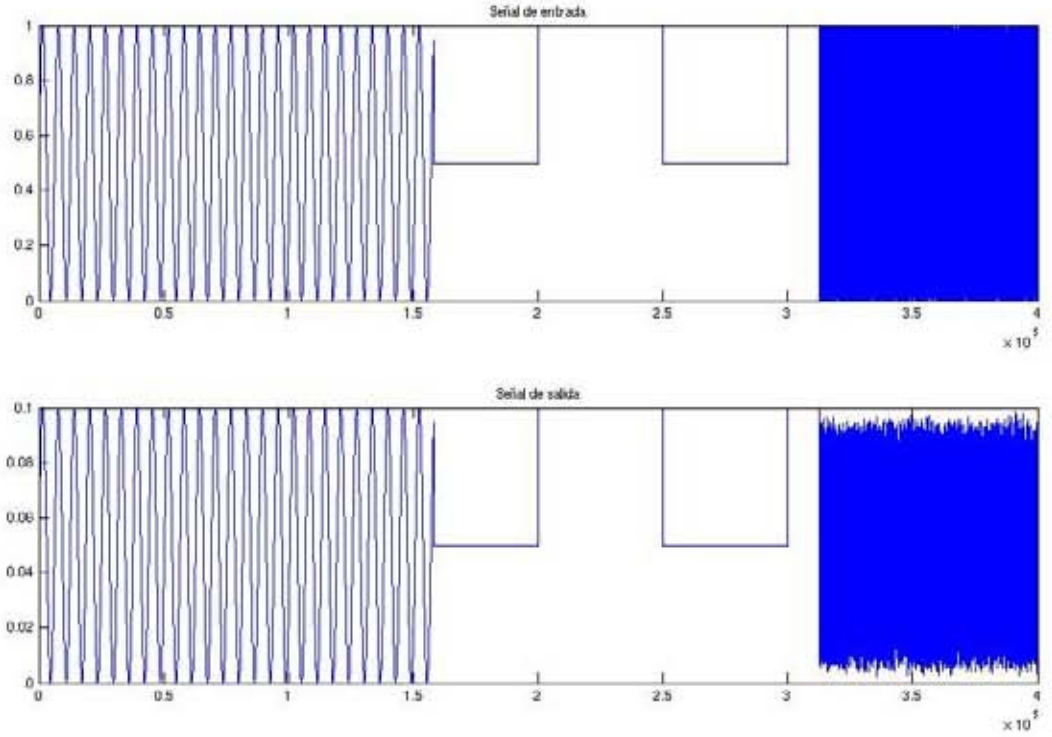

Figura 4.2 Señal de entrada y salida correspondiente al caso de estudio

Una vez obtenidos los datos del sistema se lleva a cabo la prueba para diferentes escenarios de entrenamiento de la red para determinar los valores de las variables para la estimación de parámetros y el entrenamiento de la red mencionadas en el capítulo anterior, que llevan a un mejor desempeño de la misma. Se presentan en las siguientes figuras los distintos desempeños obtenidos por dichas redes con algunos valores de las variables, empleando un escenario libre de fallas. La primera gráfica de cada figura representa la distancia euclidiana entre la matriz de pesos y el vector de parámetros actual, la segunda gráfica representa la neurona activada para cada vector de parámetros. Como vemos en algunos casos el desempeño es muy bueno, pues es capaz de activar 3 neuronas diferentes para cada una de las señales presentadas, además de que la distancia mejora entre las que pudieron seleccionar tres neuronas diferentes; en otros casos esto no se da.

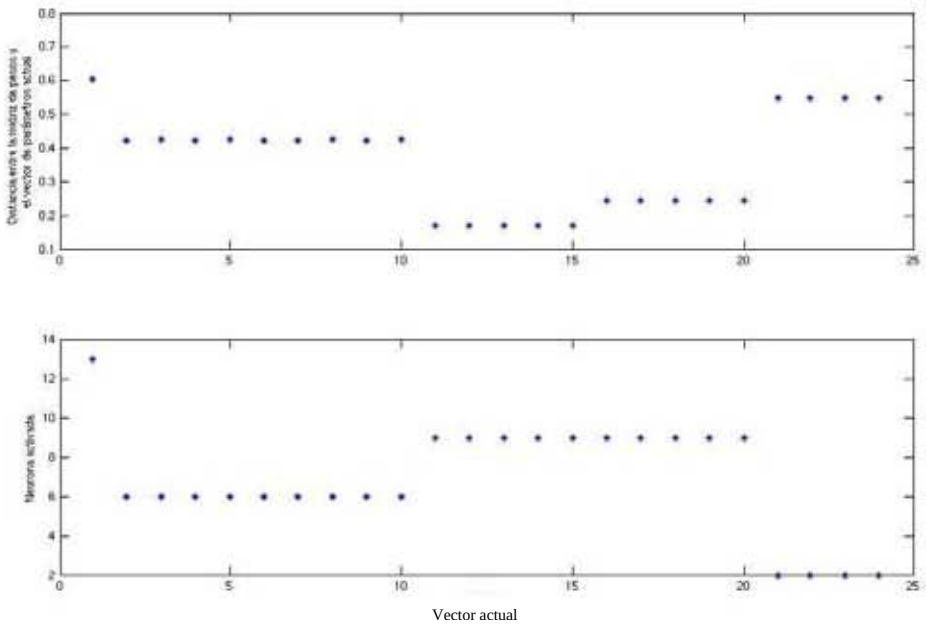

*Figura 4.3 Desempeño de la red con Tamaño de red =4*4; na=nb=5; Tamaño de ventana =20000; Épocas = 100;* η*= 0.2;* σ*= 3.0.*

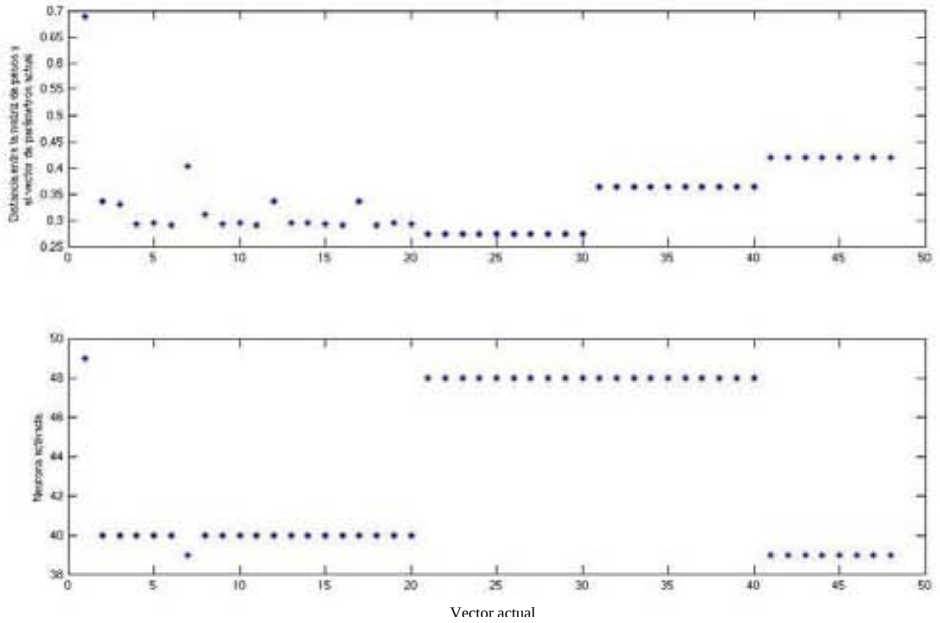

*Figura 4.4 Desempeño de la red con Tamaño de red =7*7; na=nb=5; Tamaño de ventana =10000; Épocas = 100;* η*= 0.1;* σ= 3.0.

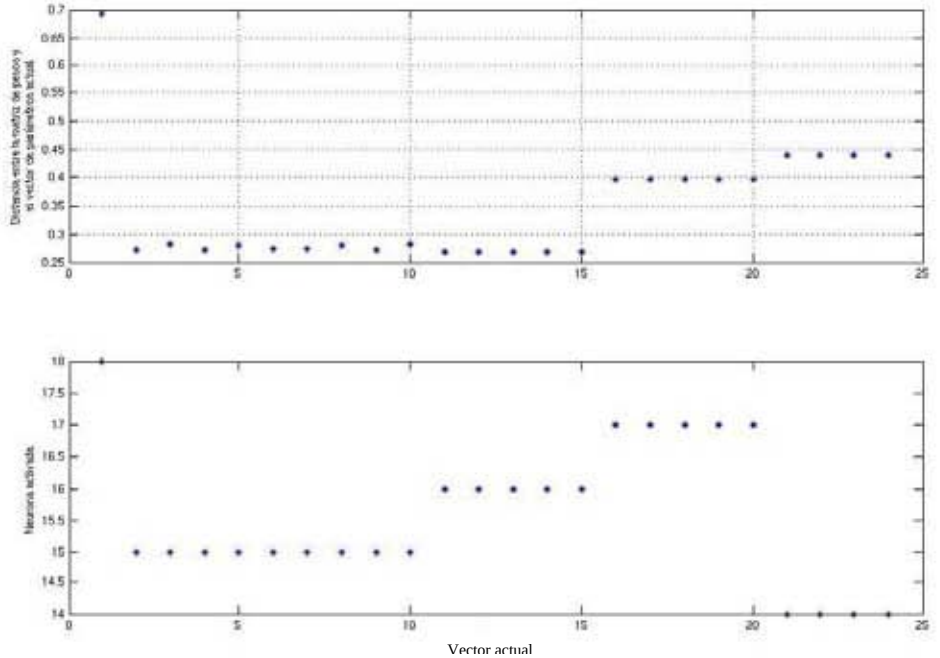

*Figura 4.5 Desempeño de la red con Tamaño de red =6*6; na=nb=7; Tamaño de ventana =20000; Épocas = 100;* η*= 0.1;* σ= 3.0.

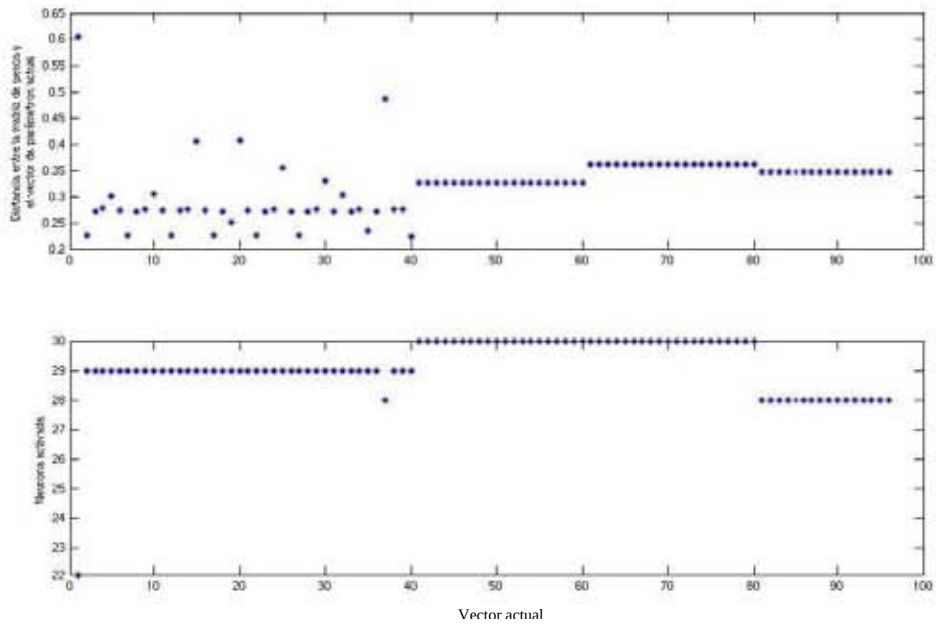

*Figura 4.6 Desempeño de la red con Tamaño de red =8*8; na=nb=3; Tamaño de ventana =5000; Épocas = 200; η= 0.015; σ= 3.0.*

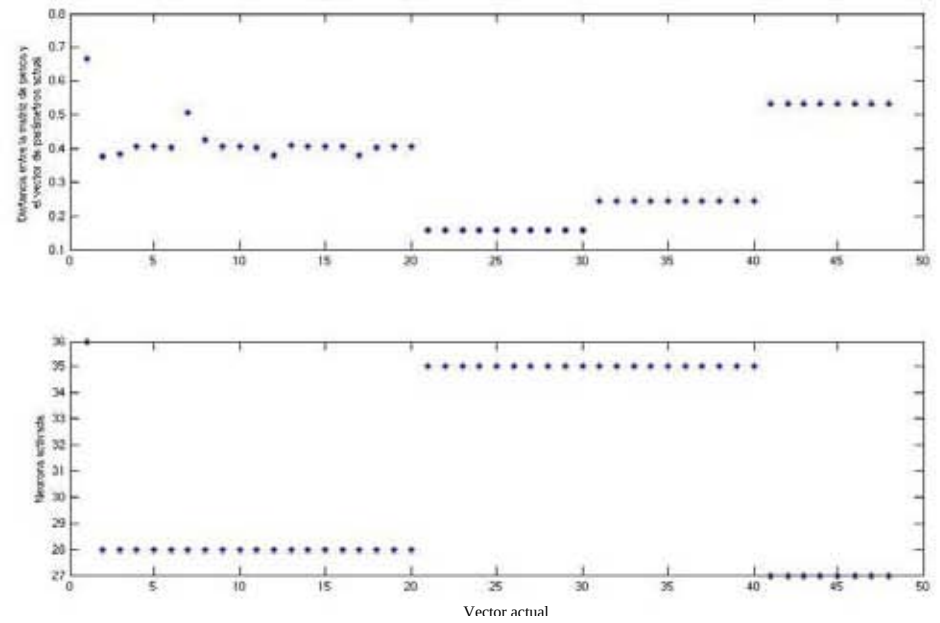

*Figura 4.7 Desempeño de la red con Tamaño de red =6*6; na=nb=5; Tamaño de ventana =10000; Épocas = 150; η= 0.2; σ= 3.0.*

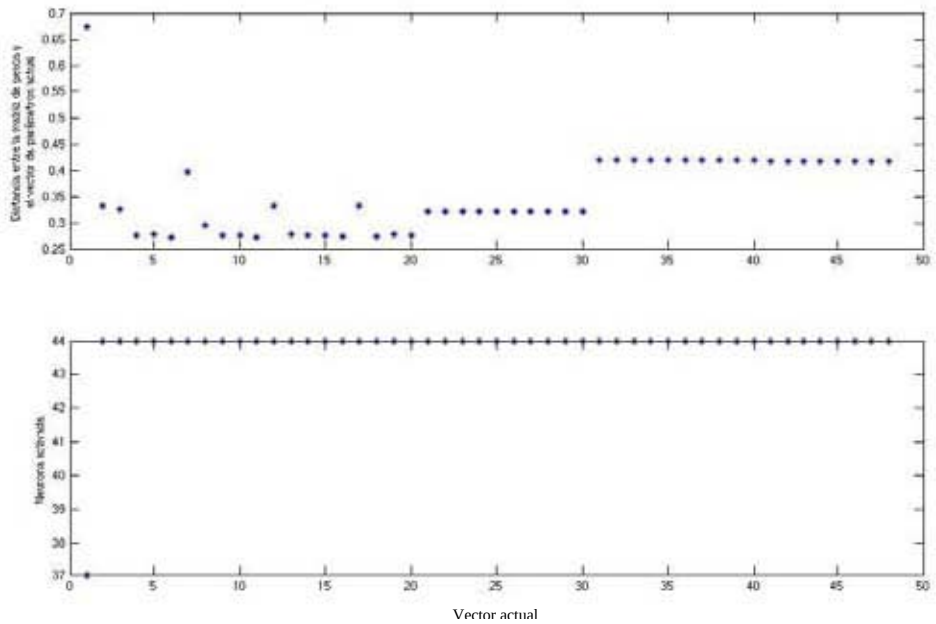

*Figura 4.8 Desempeño de la red con Tamaño de red =7*7; na=nb=5; Tamaño de ventana =10000; Épocas = 100;* η*= 0.1;* σ*=1.0.*

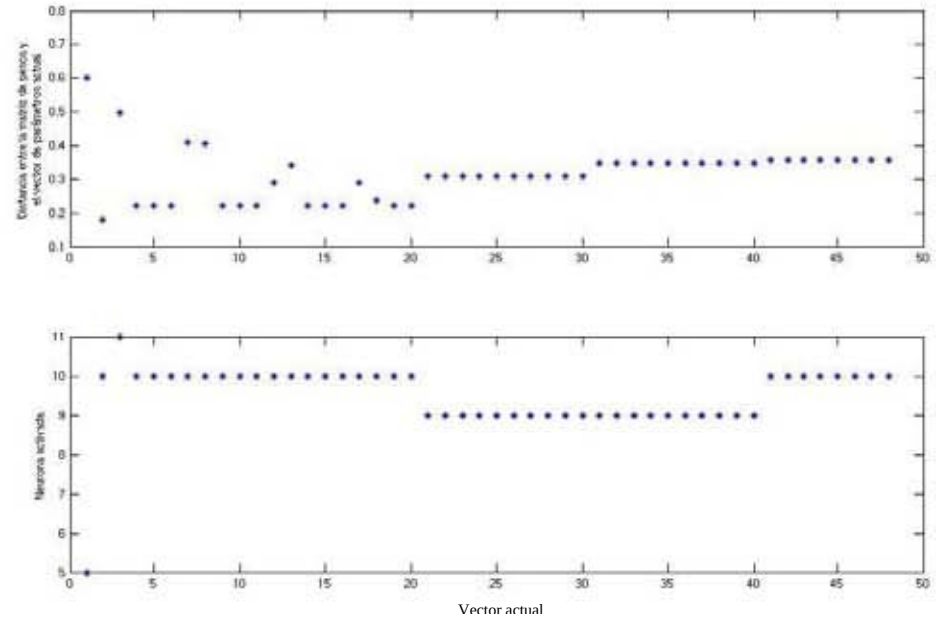

*Figura 4.9 Desempeño de la red con Tamaño de red =4*4; na=nb=3; Tamaño de ventana =10000; Épocas = 200;* η*= 0.1;* σ=3.0.

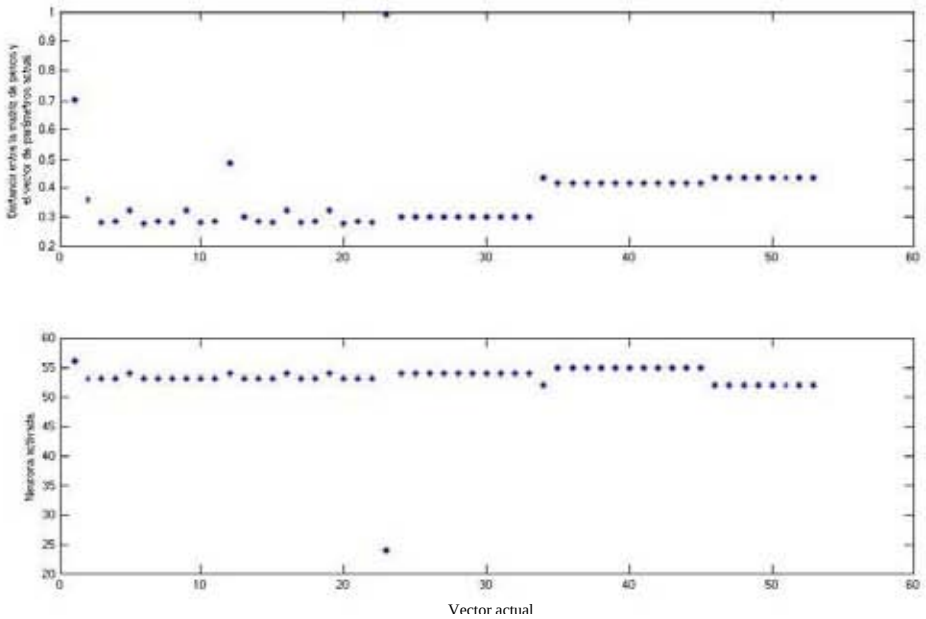

*Figura 4.10 Desempeño de la red con Tamaño de red =8*8; na=nb=7; Tamaño de ventana =9000; Épocas = 200; η= 0.015; σ=3.0*

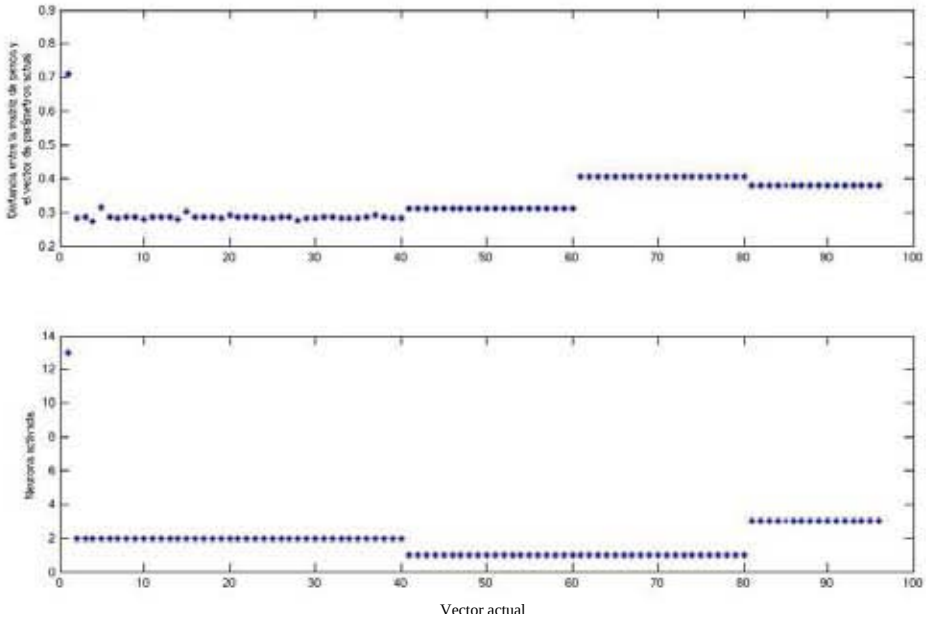

*Figura 4.11 Desempeño de la red con Tamaño de red =6*6; na=nb=5; Tamaño de ventana =5000; Épocas = 200; η= 0.015; σ=3.0*

La siguiente tabla muestra en resumen lo expuesto anteriormente. Esta tabla fue obtenida por medio de una ejecución exhaustiva de cada variable bajo un rango local, manteniendo el resto constante. Se encontró que los casos reportados son aquellos que dan una respuesta aceptable con la combinación mostrada. Otros de estos son ejemplos de casos que no mostraron ser tan eficientes. El resto de los casos no es mostrado por razones de calidad.

Señal	*Tamaño de la red*	*na; nb*	*Tamaño de la ventana*	*Épocas*	η	σ	*Neurona activada*	*Error máximo*
Seno	4*4	5	20000	100	0.2	3.0	6	0.42
Pulso							9	0.25
Aleatorio							2	0.55
Seno	7*7	5	10000	100	0.1	3.0	40	0.4
Pulso							48	0.37
Aleatorio							39	0.42
Seno	6*6	7	20000	100	0.1	3.0	15	0.28
Pulso							16 y 17	0.27 y 0.38
Aleatorio							14	0.42
Seno	8*8	3	5000	200	0.015	3.0	29	0.4
Pulso							30	0.36
Aleatorio							28	0.34
Seno	6*6	5	10000	150	0.2	3.0	28	0.5
Pulso							35	0.25
Aleatorio							27	0.54
Seno	7*7	5	10000	100	0.1	1.0	44	0.4
Pulso							44	0.43
Aleatorio							44	0.42
Seno	4*4	3	10000	200	0.1	1.0	10	0.5
Pulso							9	0.32
Aleatorio							10	0.34
Seno	8*8	7	9000	200	0.015	3.0	53 y 54	0.36
Pulso							54 y 55	0.4
Aleatorio							52	0.43
Seno	6*6	5	5000	200	0.015	3.0	2	0.3
Pulso							1	0.42
Aleatorio							3	0.38

Tabla 4.1. Resumen de resultados

Tomando en consideración como mejor resultado el que muestra la figura 4.11 (resaltando nuevamente que se considero solamente las pruebas realizadas, lo que hace que sea el mejor de una muestra de posibles escenarios), se procede a hacer la prueba de la red ya entrenada con escenario para el sistema dinámico en los que existe una falla.

La figura 4.12 muestra la señal empleada en cuya salida existen tres diferentes fallas. En este caso la falla elegida es un incremento constante del tipo rampa con lo cual se busca una respuesta aceptable ante un cambio constante sin importar el tipo de entrada. En este caso existen dos premisas: la saturación no existe y la frecuencia no es un factor de cambio.

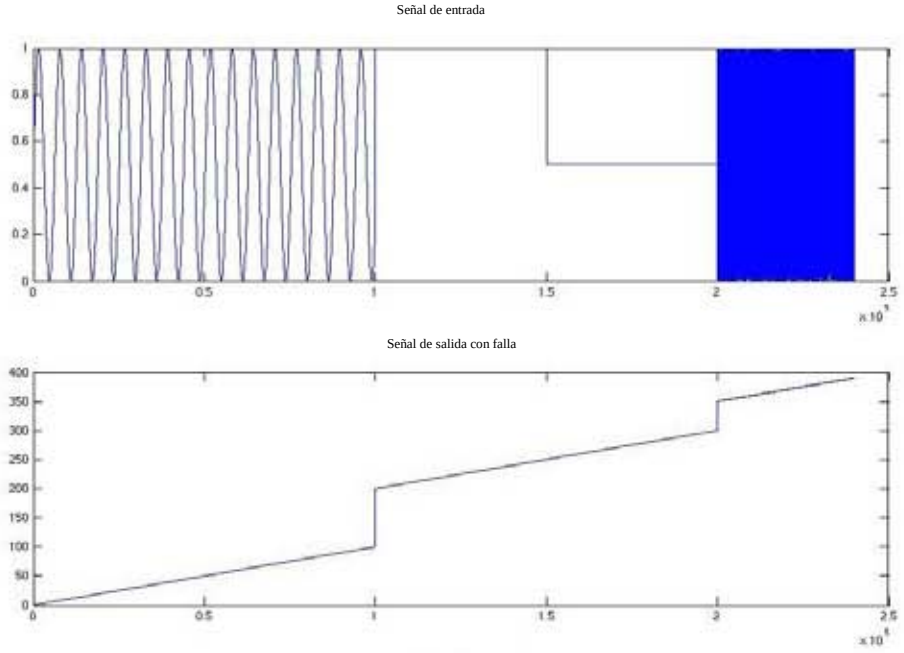

Figura 4.12 Señal de entrada y salida correspondiente a un escenario con fallas

La siguiente figura muestra el resultado de la prueba de la red con los 3 diferentes escenarios con fallas mostrados en la figura anterior para las condiciones de la figura 4.11.

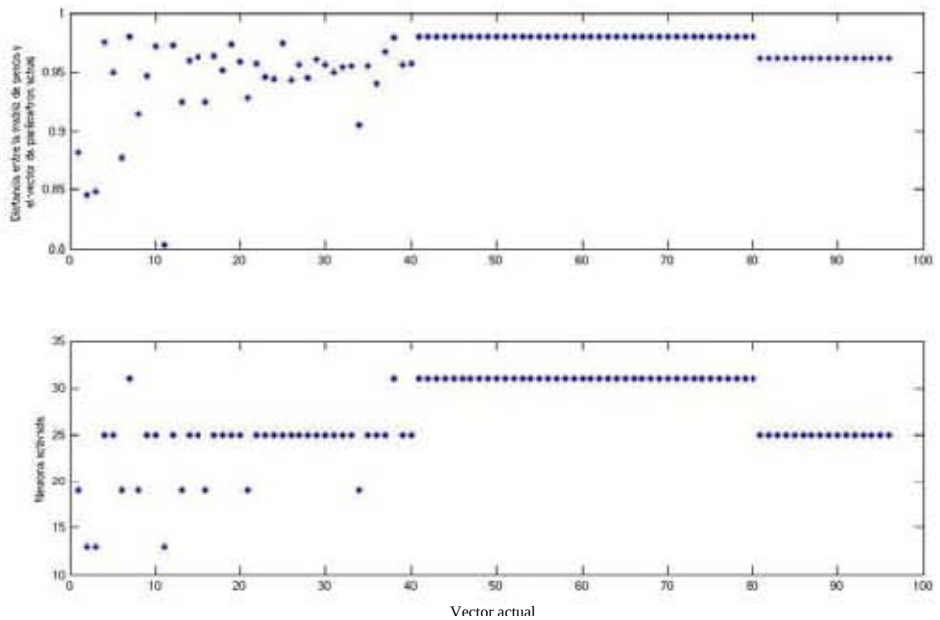

*Figura 4.13 Desempeño de la red al presentarse fallas con Tamaño de red=6*6; na=nb=5; Tamaño de ventana=5000; Épocas = 200;* η*= 0.015;* σ=3.0

Vemos que la red es capaz de clasificar las fallas en neuronas diferentes a las activadas con escenarios sin fallas, aunque en este caso repite en la primera y en la tercer falla neuronas de activación. Sin embargo, se puede tomar como satisfactorio el resultado, pues la red reconoce fallas gracias al entrenamiento previo sin fallas por medio de las activación de diferentes neuronas, que es lo que se buscaba.

En el entendido de que en algunos casos no siempre la red demuestra un mejor desempeño en la etapa de entrenamiento de aquel en la etapa de prueba, se presentan otros resultados probando con otras redes entrenadas para mostrar como hacen estas la clasificación de las fallas.

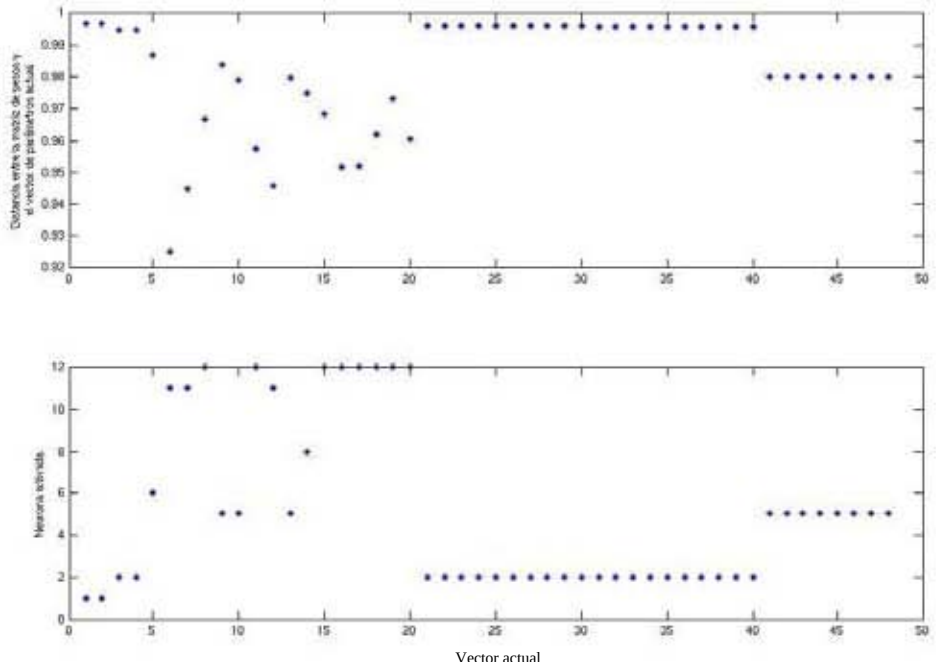

*Figura 4.14 Desempeño de la red al presentarse fallas con Tamaño de red=6*6; na=nb=5; Tamaño de ventana=10000; Épocas = 150;* η*= 0.2;* σ*=3.0*

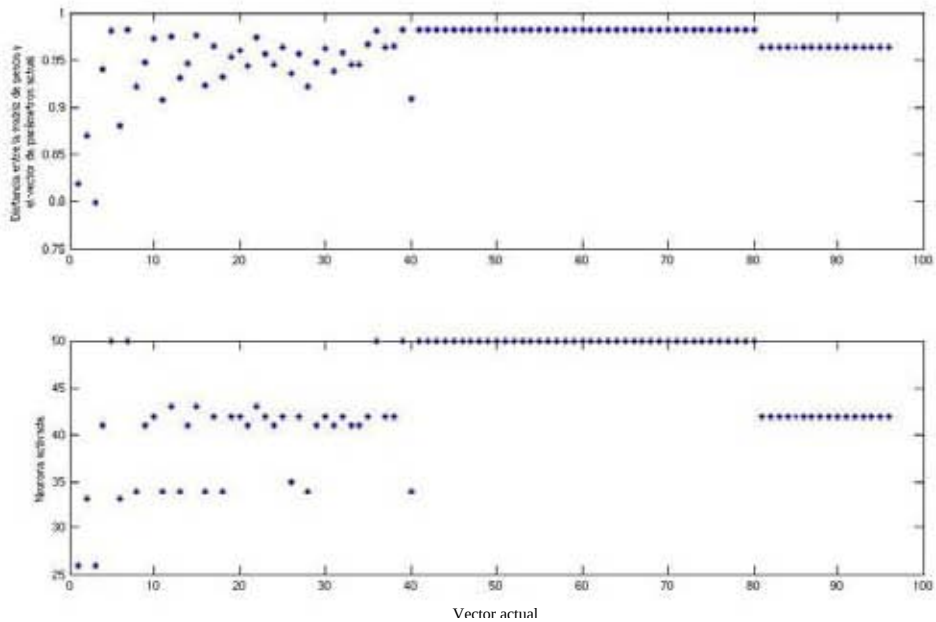

*Figura 4.15 Desempeño de la red al presentarse fallas con Tamaño de red=8*8; na=nb=5; Tamaño de ventana=5000; Épocas = 200;* η*= 0.015;* σ=3.0

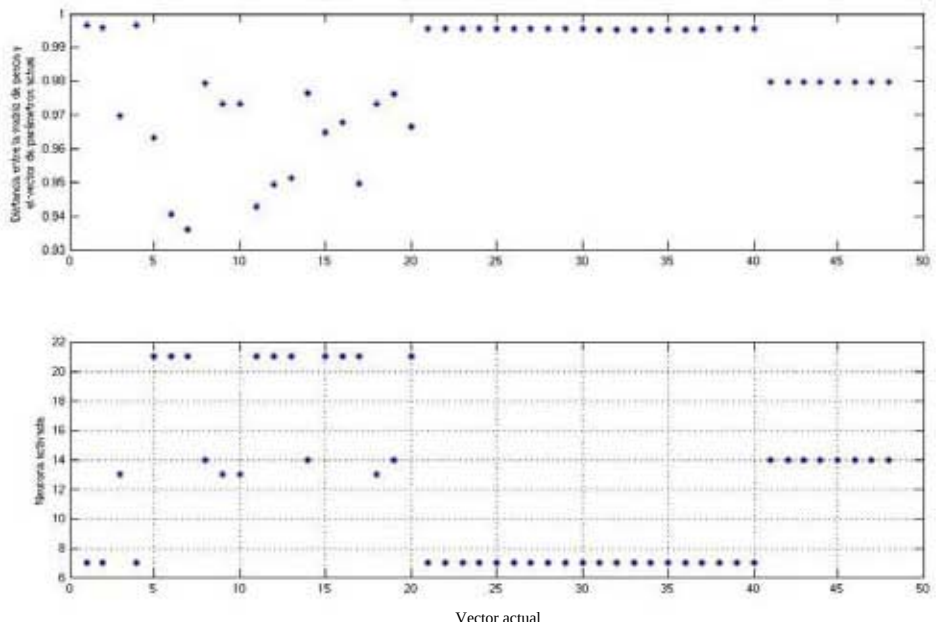

*Figura 4.16 Desempeño de la red al presentarse fallas con Tamaño de red=7*7; na=nb=5; Tamaño de ventana=10000; Épocas = 100; η= 0.1; σ=3.0*

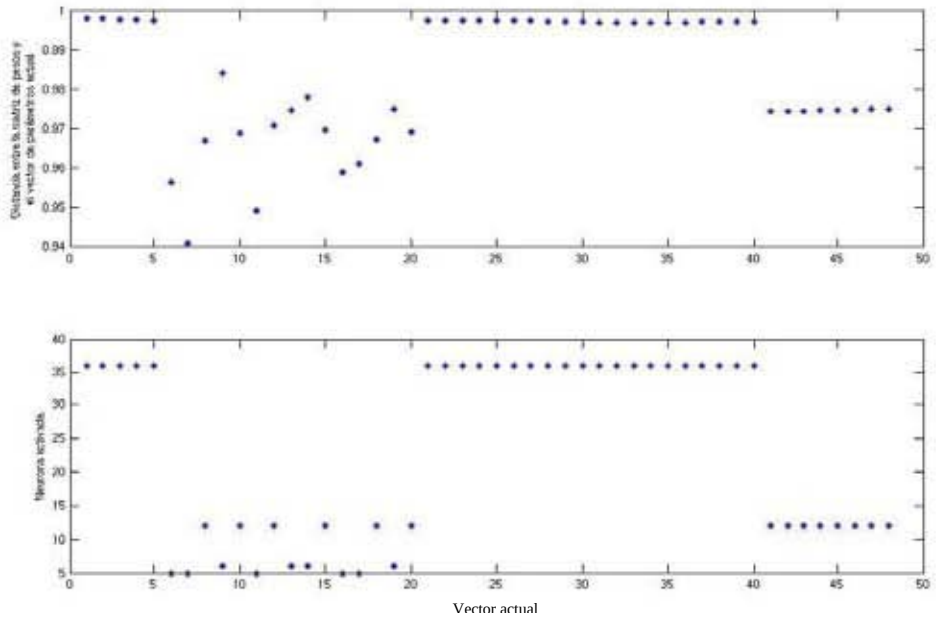

*Figura 4.17 Desempeño de la red al presentarse fallas con Tamaño de red=6*6; na=nb=7; Tamaño de ventana=10000; Épocas = 100; η= 0.2; σ=3.0*

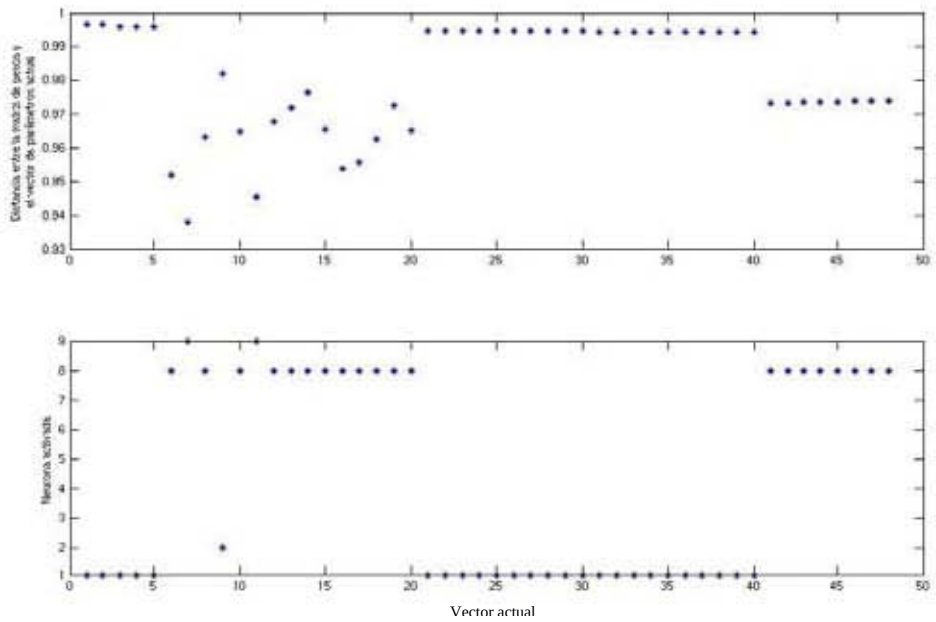

*Figura 4.18 Desempeño de la red al presentarse fallas con Tamaño de red=7*7; na=nb=7; Tamaño de ventana=10000; Épocas = 200;* η*= 0.2;* σ=3.0

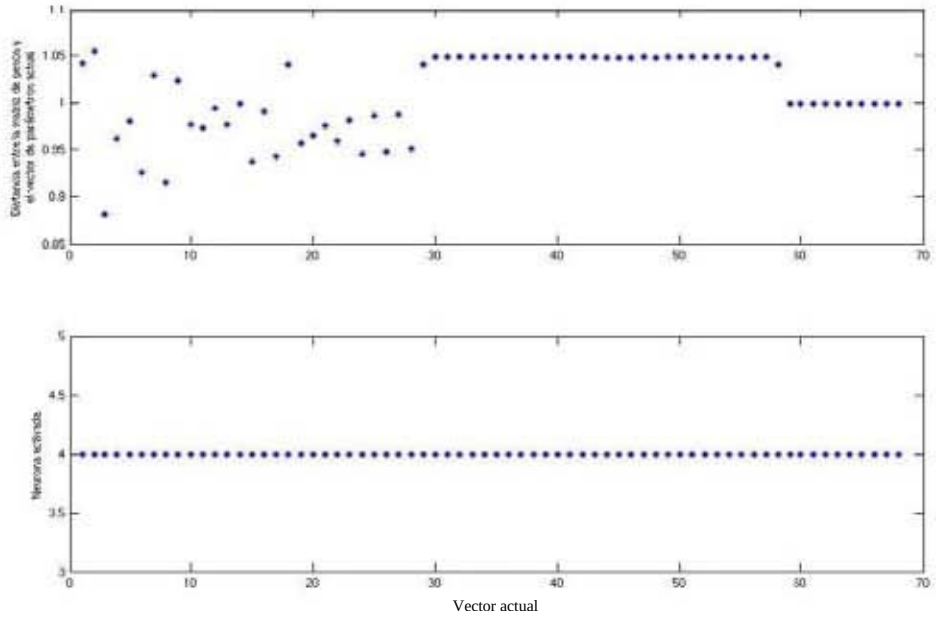

*Figura 4.19 Desempeño de la red al presentarse fallas con Tamaño de red=4*4; na=nb=5; Tamaño de ventana=7000; Épocas = 200;* η*= 0.015;* σ*=3.0*

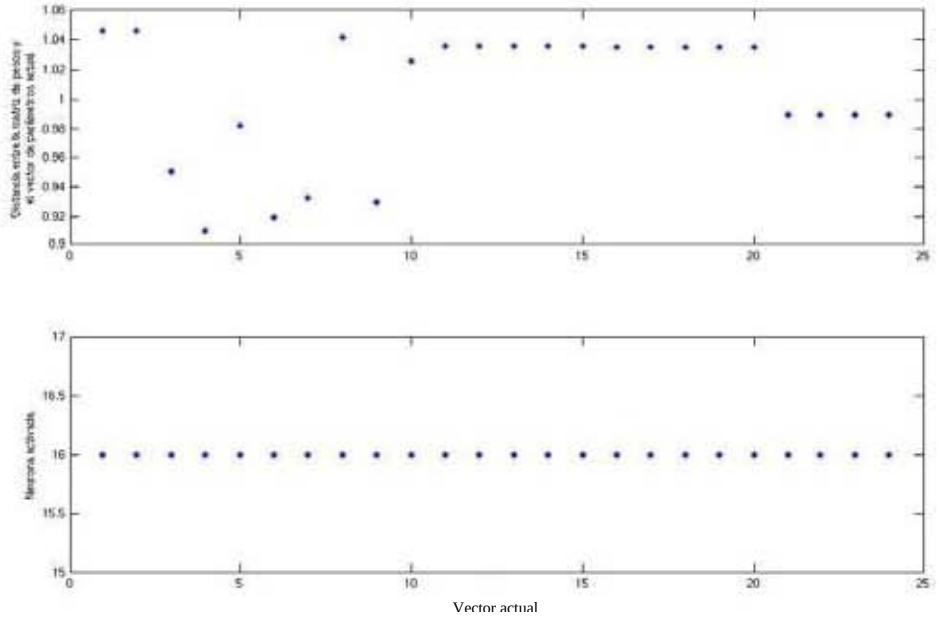

*Figura 4.20 Desempeño de la red al presentarse fallas con Tamaño de red=4*4; na=nb=5; Tamaño de ventana=20000; Épocas = 100; η= 0.2; σ=3.0*

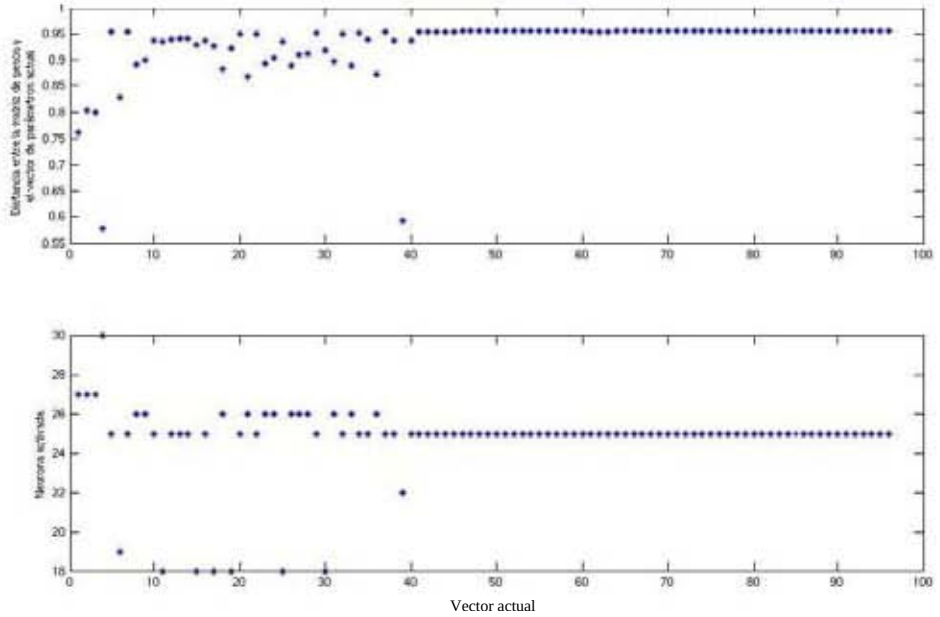

*Figura 4.21 Desempeño de la red al presentarse fallas con Tamaño de red=8*8; na=nb=3; Tamaño de ventana=5000; Épocas = 200; η= 0.015; σ=3.0*

Los casos anteriores fueron pruebas de redes que tenían un buen desempeño, sin embargo no todos los casos mostraron un buen resultado en la prueba. En el mejor de los casos conviene aun mas seleccionar los valores para las variables mostrados en la figura 4.14, los cuales detectaron mejor las tres diferentes fallas.

La tabla 4.2 muestra en resumen los resultado presentados en las ultimas figuras.

Señal	*Tamaño de la red*	*na; nb*	*Tamaño de la ventana*	*Épocas*	η	σ	*Neurona activada*	*Error máximo*
Falla 1	6*6	5	5000	200	0.015	3.0	13,19 y 25	0.98
Falla 2							31	0.98
Falla 3							25	0.96
Falla 1	6*6	5	10000	150	0.2	3.0	1,2,5,6,8 11,12	0.99
Falla 2							2	0.99
Falla 3							5	0.98
Falla 1	8*8	5	5000	200	0.015	3.0	26,33,34,41,42,43,50	0.98
Falla 2							50	0.98
Falla 3							42	0.97
Falla 1	7*7	5	10000	100	0.1	3.0	7,13,14,21	0.99
Falla 2							7	0.99
Falla 3							14	0.98
Falla 1	6*6	7	10000	100	0.2	3.0	5,6,12,36	0.99
Falla 2							36	0.99
Falla 3							12	0.97
Falla 1	7*7	7	10000	200	0.2	3.0	1,2,8,9	0.99
Falla 2							1	0.99
Falla 3							8	0.97
Falla 1	6*6	7	20000	100	0.1	3.0	36	1.01
Falla 2							36	1.01
Falla 3							36	0.97
Falla 1	4*4	5	20000	100	0.2	3.0	16	1.04
Falla 2							16	1.03
Falla 3							16	0.99
Falla 1	8*8	3	5000	200	0.015	3.0	18,19,25,26	0.95
Falla 2							22,25	0.95
Falla 3							25	0.95

Tabla 4.2. Resumen de resultados

CAPITULO 5. CONCLUSIONES

A partir del análisis de resultados expuesto en el capítulo anterior se considera que se ha cumplido con la meta de construir un esquema de detección y clasificación de fallas en línea, que nos permite distinguir los cambios de puntos de operación así como un monitoreo certero en la clasificación de fallas.

Si bien en la propuesta mostrada queda al final por parte del operador dar una interpretación a lo que la red neuronal esta mostrando como resultado, tiene sus ventajas cuando la señal es muy larga, o es requerida una lectura continua de mediciones manuales, pues las herramientas basadas en computación como ésta pueden ser usadas para dar soporte a los operadores para entender mejor el estado actual del proceso.

El algoritmo para la creación del mapa auto-organizado, siendo básicamente igual al presentado por Kohonen, se ve enriquecido al usar como procesamiento de los datos la estimación de parámetros del sistema, lo que ayuda a lograr que este tipo de redes neuronales sea capaz de aprender series de tiempo, y así reconocer los diferentes estados en los que opera el sistema, tanto libre de fallas como en presencia de éstas.

Otra de las ventajas que nos da el uso de mapas auto-organizados es que pueden ser usados tanto para revisar si el mapa conoce un nuevo patrón, o si muestra una nueva neurona dentro del mapa o un conjunto de estas la cual dé la mejor respuesta del nuevo patrón, dando así la opción de generalización, necesario en un proceso como éste en el cual la estimación de parámetros no garantiza ser exacta.

Con lo que respecta a la estimación de parámetros, se vio que mucho depende de los datos el mejor desempeño de la aplicación del algoritmo de mínimos cuadrados, y que aún contando con las herramientas que Matlab 7.0 y Simulink ofrecen, esto no implica que la estimación de parámetros este garantizada ser la mejor, pero si nos da un buen parámetro de confianza aunado a otras herramientas, tales como los parámetros variables en la red neuronal, para asegurar así un eficaz desempeño del mapa auto-organizado.

Para el caso de la red neuronal, el uso de los mapas auto-organizados como un clasificador de la conducta del sistema de la forma en que ha sido planteada en el capítulo 3 nos muestra que en conjunción con estimación de parámetros refleja muy buenos resultados. Es importante que se observe que las condiciones iniciales de los valores necesarios en el aprendizaje de la red, como son el número de neuronas en la red, el número de épocas de entrenamiento, el factor de aprendizaje y el factor que ayuda a modificar las vecindades pueden diferir en cada caso de estudio, obligando así a desarrollar una selección adecuada de los mismos para llevar al mejor desempeño principalmente al proceso en línea. En el desarrollo de esta tesis se vio que las variables dependían de la adecuada estimación de parámetros, y que cada caso estudiado daba una pauta en particular para una selección idónea de estos valores. Cabe aclarar que hace falta una búsqueda exhaustiva en todas las posibles combinaciones para los valores de estas variables, que por falta de tiempo no fue posible de realizar, y que queda como trabajo futuro.

Además, como parte de un trabajo futuro, se deja la idea de emplear otros métodos para el procesamiento de los datos, como podría ser la aplicación de onduletas en lugar de la estimación de

parámetros por mínimos cuadrados, lo que lleva a una revisión de la clasificación de fallas. Además sería importante probar con distintos casos de estudio.

REFERENCIAS:

Alhoniemi, E. *Unsupervised pattern recognition methods for exploratory analysis of industrial process data*. Doctoral thesis, Helsinki University of Technology, Espoo 2002, 65 pp. ISBN 951-22-6242-8. UDC 004.032.26:004.93:66.102:519.237.8

Chi-Tsong, Chen. *Linear system. Theory and design*. Oxford University Press. Tercera edición. 1999

Getler, Janos J. *Fault detection and Diagnosis in Engineering Systems*. Marcel Dekker Inc. 1998.

Hagan, Martin T., Demuth Howard B., Beale Mark H. *Neural Network Design*. ISBN 0-9717321-0-8. University of Colorado.

Narendra, K. S. and Parthasarathy K. *Identification and control dynamical systems using neural networks.*, IEEE Trans. Neural Networks, 1:4-27, 1990.

Haykin, S. *Neural Networks: A Comprehensive Foundations*. Macmillan College Publishing Co., New York 1994.

Hebb, DO. *The Organization of Behavior: A Neuropsychológical Theory*. John Wiley, 1949.

Hoffman, K. y Kunze, R. *Álgebra lineal*. Prentice Hall. 1973.

Hunt / Lipsman / Rosenberg. *A Guide to MATLAB: for Beginners and Experienced Users*. Cambridge University Press, 2001

Jämsa-Jounela et. al. *A process monitoring system based on the Kohonen self-organizing maps*. 1988

Kohonen, T. *Self-organizing Maps*. Springer Series in Information Sciences. Springer. 1995.

Lee, S., Kim, J.T., Lee, J.W., Lee, D.Y., Kim K.Y. *Model Based Fault Detection and Isolation Method Using ART2 Neural Network*. International Journal of Intelligent Systems, Vol. 18, 2003

Ljung, L. *System Identification: Theory for the User*. Prentice-Hall. EnglewoodCliffs, N. J., 1987.

APÉNDICE A. CÓDIGO DE PROGRAMAS

Estimación de parámetros:

```
load train12

%Normalizamos las entradas y salidas
nn=[u;Y];
entrada=abs(u)/sqrt(sum(nn.^2));
salida=abs(Y)/sqrt(sum(nn.^2));

param=5;
maxtam=12000;
[val1s,val2s]=size(salida);

%Efectuamos la estimación de parámetros
for i=1:floor(val1s/maxtam)
   [jiji  jojo]=rarmax([salida(1+maxtam*(i-1):maxtam*i)…
                        entrada(1+maxtam*(i-1):maxtam*i)],[param param 0 1],'ff',0.98);
   [val1j,val2j]=size(jiji);
   erryhat(i,:)=abs(jojo-salida(1+maxtam*(i-1):maxtam*i));
  %Se calcula el error en la estimación para elegir el mejor  vector de parámetros
   [val,indval]=min(erryhat(i,:));
   errvect(i)=val;
   mtx_vg(i,:)=jiji(indval,:);
end

%Declaramos los vectores necesarios para el entrenamiento como los patrones
Vg=mtx_vg';
[x,y]=size(Vg);

%Llamamos a la función encargada del entrenamiento
entrena
```

Entrenamiento:

```
%Normalizamos Vg
for colum=1:y
  Vgnorm(:,colum)=abs(Vg(:,colum))/sqrt(sum(Vg(:,colum).^2));
end

%INICIA LA DEFINICION DE LA RED NEURONAL
D=m*n; %numero de neuronas de salida
%inicializamos valores necesarios para el aprendizaje
sigma_cero=3.0;
eta_cero=0.05;
eta=1.0;
```

```
tao2=1000;
epoca=150;
tao1=1000/log(sigma_cero);

topologia=creaTopologia(m,n); %Diseña la forma y tamaño de la red
coord_neuronas=creaCoord_neuronas(m,n); %Da las coordenadas de las neuronas dentro del lattice

%Inicializamos la matriz de pesos.
aa = 0; bb = 0.1;
w = aa + (bb-aa) * rand(D,x); %matriz de pesos
wini=w;


   C=x; %numero de neuronas de entrada o caracteristicas en el vector
       %de entrada
   E=y;  %numero de vectores de entrada en cada ventana

   %Inicia el entrenamiento
   t=1;
   while t<epoca & eta>=0.01 %Epocas en las que se ejecutara el
                        %entrenamiento o mientras que eta sea
                        %mayor a 0.01
      eta=eta_cero*(exp(-1*(t/tao2))); %factor de aprendizaje
      choice = randperm(E); %Arreglo que contiene el orden en el cual
                        %se presentaran los vectores de entrada
      sigma=sigma_cero*(exp(-1*(t/tao1)));%Calculamos el valor de
                              %sigma que se usara para
                              %las vecindades, el cual
                              %mide el grado al cual las
                              %neuronas excitadas en la
                              %vecindad de la neurona
                              %ganadora participan en el
                              %aprendizaje

      for a=1:E
         %Se realiza el proceso competitivo para seleccionar la neurona
         %ganadora con cada uno de los vectores de entrada
         distancia=dist(w,Vgnorm(:,choice(1,a))); %Calculamos la
                                    %distancia euclidiana
                                    %entre el vector de
                                    %entrada y cada
                                    %neurona de salida
         [mindist, indice]=min(distancia); %Se elige el vector de
                              %salida con la distancia
                              %minima

        if (t==epoca-1 | eta<=0.01)
           C = strvcat('Para el vector de entrada',...
```

```
                int2str(choice(1,a)),...
                'la neurona de salida activada es la',...
                int2str(indice))
        end

        %Una vez determinada la neurona ganadora se inicia el proceso
        %cooperativo obteniendo la vecindad para la actualizacion de
        %pesos.Calculamos la distancia lateral con respecto a ganadora
        [gi,gj]=find(topologia==indice); %Buscamos las coordenadas
                                %de la neurona ganadora
                                %dentro del lattice
        ind_ganadora=[gi;gj];
        %Calculamos distancia lateral entre la neurona ganadora y
        %las restantes pertenecientes al lattice
        dist_lateral=dist(coord_neuronas',ind_ganadora);
        %calculamos vecindades
        h=exp(-1*((dist_lateral.^2)/(2*(sigma^2))));

        %Ahora se ejecuta el proceso adaptativo donde se actualiza la
        %matriz de pesos de la red neuronal
        for k=1:D
           w(k,:)=(w(k,:)*(1-eta))+(eta*(h(k,1)*...
                                Vgnorm(:,choice(a))))';
        end
      end
    t=t+1
   end

save w w param %Guardamos la matriz de pesos
```

Función para crear la red:

```
function topologia=creaTopologia(m,n)
%Topologia:matriz que nos sera util solo para determinar las vecindades, la cual
%contendra las coordenadas que le corresponden a cada neurona de salida
   contador=1;
   for j=1:n
      for i=1:m
         topologia(i,j)=contador;
         contador=contador+1;
      end
   end
```

Función para determinar las coordenadas de las neuronas dentro del lattice:

```
function coord_neuronas=creaCoord_neuronas(m,n)
%coord_neuronas:matriz que nos ayudara para calcular las diferencias
%laterales con respecto a la neurona ganadora
   contador=1;
   for i=1:m
      for j=1:n
         coord_neuronas(1,contador)=j;
         coord_neuronas(2,contador)=i;
         contador=contador+1;
      end
   end
```

Prueba en línea:

```
load train12
load w

tam=7;
param=5;
maxtam=5000;
epoca=100;
eta_cero=0.1;
sigma_cero=3.0;
m=tam;
n=tam;

%Estimación de parámetros
nn=[u;Y];
entrada=abs([u(1:100000);u(200000:300000);u(350000:390000)])/sqrt(sum(nn.^2));
salida=abs([Y(1:100000);Y(200000:300000);Y(350000:390000)])/sqrt(sum(nn.^2));

maxtam2=maxtam;
[val1s,val2s]=size(salida);
for i=1:floor(val1s/maxtam2)
        [jiji jojo]=rarmax([salida(1+maxtam2*(i-1):maxtam2*i)...
                                entrada(1+maxtam2*(i-1):maxtam2*i)],[param param 0 1],'ff',0.98);
         [val1j,val2j]=size(jiji);
         erryhat(i,:)=abs(jojo-salida(1+maxtam2*(i-1):maxtam2*i));
         [val,indval]=min(erryhat(i,:));
         mtx_vg(i,:)=jiji(indval,:);
end

Vg=mtx_vg';
 [x,y]=size(Vg);
```

```
 %Normalizamos Vg
for colum=1:y
        Vgnorm(:,colum)=abs(Vg(:,colum))/sqrt(sum(Vg(:,colum).^2));
end

%Probamos los vectores actuales en la red
for colum=1:y
         resultado=dist(w,Vgnorm(:,colum)); %Distancia euclidiana
        [mindist, indice]=min(resultado);
        [maxdist, indicem]=max(resultado);
        distanciamin(colum)=mindist;
         indicemin(colum)=indice;
        distanciamax(colum)=maxdist;
         indicemax(colum)=indicem;
 end

 figure;subplot(2,1,1);plot(distanciamin,'*');title(archivo);xlabel('Distanciaminima falla');
 subplot(2,1,2);plot(indicemin,'*');xlabel('Indice minimo falla');
```