



**INSTITUTO POLITÉCNICO
NACIONAL**



ESCUELA SUPERIOR DE INGENIERÍA MECÁNICA Y ELÉCTRICA

UNIDAD CULHUACAN

SECCIÓN DE ESTUDIOS DE POSGRADO E INVESTIGACIÓN

**COMPARACIÓN DE LAS REDES
NEURONALES ARTIFICIALES
APLICADAS AL DIAGNÓSTICO DE
TURBINAS DE GAS**

T E S I S

QUE PARA OBTENER EL GRADO DE:

**MAESTRO EN CIENCIAS DE INGENIERÍA EN
SISTEMAS ENERGÉTICOS**

P R E S E N T A

ING. EULALIO TORRES GARCÍA

ASESOR: DR. IGOR LOBODA

SEPTIEMBRE 2010

AGRADECIMIENTOS

Al Instituto Politécnico Nacional, por soportar mi conocimiento y por el apoyo económico.

A mi asesor, por guiarme en el camino.

A mis padres, por ser el pilar de mi formación, cada esfuerzo, cada gota de sudor esta reflejada aquí...

A la familia Cruz Trejo, por aceptarme como parte de la familia y por brindarme un segundo hogar, en especial a la Señora Jovita Cruz.

A Dulce Ibarra, por ser una compañera de trabajo desafiante.

ÍNDICE

I. RELACION DE TABLAS Y FIGURAS	i
II. RESUMEN	iv
III. ABSTRACT	v
IV. JUSTIFICACIÓN	vi
V. INTRODUCCIÓN	vii

CAPÍTULO I ESTADO DEL ARTE

Introducción.....	2
1.1 Turbinas de gas.....	2
1.1.1 Historia.....	2
1.1.2 ¿Qué es una turbina de gas?.....	2
1.1.3 Principales componentes.....	3
1.1.4 Funcionamiento	4
1.2 Diagnóstico de turbinas de gas	5
1.2.1 ¿Qué es el diagnóstico?.....	5
1.2.2 Etapas del diagnóstico	5
1.2.3 Diagnóstico en la actualidad	5
1.2.4 Situación del diagnóstico y las turbinas de gas en México.....	6
1.3 Reconocimiento de patrones.....	7
1.3.1 Antecedentes Históricos	8
1.3.2 Tipos de problemas de clasificación.....	8
1.3.3 Enfoques dentro del reconocimiento de patrones	9
1.3.4 Fundamentos Matemáticos	10
1.3.4.1 Teoría de decisión de Bayes	10
1.3.4.2 Clasificadores Gaussianos	11
1.3.4.3 Árboles de decisión	12
1.3.4.4 K vecinos más cercanos.....	13
1.3.4.5 Support Vector Machine (SVM)	14
1.3.4.6 Redes Neuronales Artificiales (bases).....	15
1.3.5 Campos de aplicación	16
1.3.6 Algunos Desarrollos	17
Conclusiones.....	18

CAPÍTULO II REDES NEURONALES APLICADAS Y ALGORITMOS DE ENTRENAMIENTO

Introducción.....	20
2.1 Funcionamiento de una red biológica.....	21
2.2 Características de una red neuronal artificial	22
2.3 Funciones de transferencia	23
2.3.1 Limitador fuerte (hardlim)	24
2.3.2 Función de transferencia lineal (purelin)	25
2.3.3 Función de transferencia tipo logarítmica sigmoidea (logsig).....	26
2.3.4 Principales funciones de transferencia.....	26
2.4 Topología de una red	27

2.5 Perceptrón.....	30
2.6 Retropropagación (backpropagation)	31
2.7 Redes neuronales de base radial (RBR)	33
2.7.1 Diseño más eficiente	35
2.7.2 Redes neuronales probabilísticas	36
2.7.3 Regresión generalizada	37
Conclusiones.....	37

CAPÍTULO III DESCRIPCIÓN DE LOS ALGORITMOS

Introducción.....	39
3.1 Enfoque de diagnóstico	39
3.2 Estructura del algoritmo	41
3.3 Ensamblados de datos de entrenamiento	41
3.3.1 Declaración de constantes	42
3.3.2 Lectura de datos	43
3.3.3 Formación del arreglo multidimensional	43
3.3.4 Generación de vectores de entrada	44
3.3.5 Generación de vectores de prueba	45
3.4 Formación de entradas a la red	46
3.5 Entrenamiento de la red (redes de retropropagación).....	47
3.5.1 Algoritmo de entrenamiento Levenberg Marquart	47
3.5.2 Gradiente descendente	48
3.5.3 Gradiente descendente con momento	49
3.5.4 Tasa de aprendizaje adaptativa	49
3.5.5 Gradiente descendente con momento y tasa de aprendizaje adaptativa	50
3.5.6 Retropropagación resiliente	50
3.5.7 Gradiente conjugado	50
3.5.8 Gradiente conjugado propuesto por Polak y Ribière	51
3.5.9 Gradiente conjugado propuesto por Beal.....	52
3.5.10 Gradiente conjugado escalado	52
3.5.11 Método de Newton.....	53
3.5.12 Método de la secante.....	53
3.6 Redes de base radial	54
3.6.1 Diseño más eficiente	54
3.6.2 Redes radiales probabilísticas	54
3.6.3 Regresión generalizada	56
3.7 Generalidades	56
3.8 Simulación	57
3.9 Cálculo de probabilidades	57
3.10 Gráfica de clases.....	58
3.11 Interface usuario	59
Conclusiones.....	59

CAPÍTULO IV EXPERIMENTACIÓN Y RESULTADOS

Introducción.....	61
-------------------	----

4.1 Procedimiento de experimentación	61
4.2 Resultados y comparación, retropropagación.....	62
4.2.1 Optimización.....	62
4.2.2 Incremento de número puntos de simulación	64
4.2.3 Cambio de semilla (número llave).....	66
4.3 Resultados y comparación, cálculos con las redes de base radial	67
4.3.1 Red de diseño mejorado.....	67
4.3.2 Red probabilística	68
4.3.3 Red de regresión generalizada	71
4.4 Comparación y resultados de las mejores redes	74
Conclusiones.....	77
CONCLUSIONES GENERALES	78
RECOMENDACIONES	80
BIBLIOGRAFÍA	81
APENDICE A: ALGORITMOS	83
APÉNDICE B: PUBLICACIONES	88

I. RELACION DE TABLAS Y FIGURAS

Figura 1.1 Esquema de una turbina de gas.....	4
Figura 1.2 Ciclo de operación Joule Brayton.....	4
Figura 1.3 Ejemplo para la teoría de Bayes.....	11
Figura 1.4 La densidad de probabilidad condicional para dos tipos en un solo parámetro. Punto 'a' sería asignado a tipo A_1 , mientras que el punto 'b' se asignarían a A_2	12
Figura 1.5 Ejemplo de árbol de decisión simple.....	13
Figura 1.6 Clasificación por medio de SVM.....	14
Figura 1.7. Sumador hecho con un amplificador operacional.....	15
Figura 1.8 Clasificación de RNA.....	16
Figura 2.1 Interconexión de dos neuronas biológicas.....	21
Figura 2.2 Comparación entre una neurona biológica y una artificial.....	22
Figura 2.3 Proceso de "sinapsis" en una neurona artificial.....	23
Figura 2.4 Modelo de una neurona artificial.....	23
Figura 2.5 Representación gráfica de la función de transferencia con limitador fuerte.....	24
Figura 2.6 Representación gráfica de la función de transferencia Hardlims.....	25
Figura 2.7 Salida de una función de transferencia lineal.....	25
Figura 2.8 Salida de una función de transferencia logarítmica sigmoidea.....	26
Figura 2.9 Funciones de transferencia utilizadas en redes neuronales.....	27
Figura 2.10 Topología de una red neuronal.....	27
Figura 2.11 Notación compacta de una red neuronal.....	28
Figura 2.12 Red neuronal de una capa oculta.....	29
Figura 2.13 Notación compacta de una RNA de una capa oculta.....	29
Figura 2.14 Red neuronal de tres capas.....	30
Figura 2.15 Representación abreviada de una RNA de tres capas.....	30
Figura 2.16 Diagrama de Venn, para un perceptrón simple.....	31
Figura 2.17 Estructura general de una red neuronal multicapa.....	33
Figura 2.18 Arquitectura de una red de base radial.....	35
Figura 2.19 Estructura de una red probabilística.....	36
Figura 2.20 Arquitectura de las redes de regresión generalizada.....	37
Figura 3.1 Representación de las clases del conducto de flujo.....	41
Figura 3.2 Estructura del algoritmo.....	42
Figura 3.3 Etapas en el ensamble de datos de entrenamiento.....	43
Figura 3.4 Declaración de constantes.....	43
Figura 3.5 Lectura de datos.....	44
Figura 3.6 Esquema de organización de los datos.....	45
Figura 3.7 Código de organización de datos.....	45
Figura 3.8 Código de generación de vectores de entrada.....	46
Figura 3.9 Generación de vectores de prueba.....	47
Figura 3.10 Formación de entradas a la red.....	48
Figura 3.11 Código para entrenamiento por medio de <code>trainlm</code>	49
Figura 3.12 Sintaxis para el entrenamiento por medio de la función <code>traingd</code>	50
Figura 3.13 Sintaxis para el entrenamiento por medio de la función <code>traingdm</code>	50

<i>Figura 3.14 Sintaxis para el entrenamiento por medio de la función traingda.</i>	51
<i>Figura 3.15 Sintaxis para el entrenamiento por medio de la función traingdx.</i>	51
<i>Figura 3.16 Sintaxis para el entrenamiento por medio de la función trainrp.</i>	52
<i>Figura 3.17 Sintaxis para el entrenamiento por medio de la función traincgf.</i>	53
<i>Figura 3.18 Sintaxis para el entrenamiento por medio de la función traincgp.</i>	53
<i>Figura 3.19 Sintaxis para el entrenamiento por medio de la función traincgb.</i>	54
<i>Figura 3.20 Sintaxis para el entrenamiento por medio de la función trainscg.</i>	54
<i>Figura 3.21 Sintaxis para el entrenamiento por medio de la función trainbfg.</i>	55
<i>Figura 3.22 Sintaxis para el entrenamiento por medio de la función trainoss.</i>	55
<i>Figura 3.23 Sintaxis para el entrenamiento por medio de la función Newrb.</i>	56
<i>Figura 3.24 Sintaxis para el entrenamiento por medio de la función newpnn.</i>	57
<i>Figura 3.25 Sintaxis para el entrenamiento por medio de la función newgrnn.</i>	58
<i>Figura 3.26 Entrenamiento de la red</i>	59
<i>Figura 3.27 Simulación de la red neuronal.</i>	59
<i>Figura 3.28 Código para el cálculo de probabilidades.</i>	60
<i>Figura 3.29 Gráfica de clases, diferentes ángulos para tener diferentes vistas de las clases.</i>	60
<i>Figura 3.30 Código para graficar las clases.</i>	61
<i>Figura 3.31 Interface de control para experimentación.</i>	61
<i>Figura 4.1 Comparación en el desempeño en funcion de error cuadratico medio (ECM) inicial y el optimizado (final).</i>	66
<i>Figura 4.2 Comparación en la probabilidad antes y despues de la optimización</i>	67
<i>Figura 4.3 Comparación de tiempo de entrenamiento.</i>	67
<i>Figura 4.4 Lado izquierdo comparación de 5 métodos con mejor menor ECM alcanzado, del lado derecho el tiempo para los mismos 5 métodos.</i>	67
<i>Figura 4.5 Comparación de cambio de régimen e incremento del número de puntos de simulación.</i>	69
<i>Figura 4.6 Error cuadrático medio (performance) en diferentes regímenes de operación con cambio de semilla.</i>	71
<i>Figura 4.7 Probabilidad en distintas regímenes y con cambio de números semilla, para clases simples.</i>	79
<i>Figura 4.8 Tiempo de cálculo para una sola combinación de semilla en diferentes regímenes de operación. Clase simples.</i>	79
<i>Figura 4.9 Probabilidad en distintas regímenes y sin cambio de números semilla para clases múltiples.</i>	81
<i>Figura 4.10 Tiempo de cálculo para una sola combinación de semilla en diferentes regímenes de operación. Clase múltiple.</i>	81
<i>Tabla 4.1 Tabla comparativa las redes de retropropagación con y sin optimización.</i>	65
<i>Tabla 4.2 Newrb con cambio en el numero de neuronas y aumento en el número de puntos de simulación. Régimen 3, números llave 1 y 2, constantes.</i>	72
<i>Tabla 4.3 Comparaciones con cambio de semilla y cambio de régimen de operación. 500 puntos de simulación por clase.</i>	73
<i>Tabla 4.4 Newpnn con cambio de semilla y cambio de régimen de operación. Valor de dispersión constante 0.026.</i>	73
<i>Tabla 4.5 Newpnn con cambio de semilla y cambio de régimen de operación. Valor de dispersión constante 1.</i>	74
<i>Tabla 4.6 Newpnn con cambio de semilla y cambio de régimen de operación. Valor de dispersión constante 5</i>	74

<i>Tabla 4.7 Newgrnn, con variación en el número de puntos de simulación y régimen de operación, numero llave 0,1, dispersión 0.026</i>	<i>75</i>
<i>Tabla 4.8 Newgrnn con cambio de semilla y cambio de régimen de operación. Valor de dispersión constante 0.026 y 350 puntos de simulación.</i>	<i>76</i>
<i>Tabla 4.9 Newgrnn con cambio de semilla y cambio de régimen de operación. Valor de dispersión constante 0.026 y 450 puntos de simulación</i>	<i>76</i>
<i>Tabla 4.10 Newgrnn con cambio de semilla y cambio de régimen de operación. Valor de dispersión constante 1 y 450 puntos de simulación</i>	<i>77</i>
<i>Tabla 4.11 Newgrnn con cambio de semilla y cambio de régimen de operación. Valor de dispersión constante 5 y 450 puntos de simulación</i>	<i>78</i>
<i>Tabla 4.12 Comparación de redes en clasificación con clases simples, con 500 puntos de operación.</i>	<i>79</i>
<i>Tabla 4.13 Comparación de redes en clasificación con clases múltiples, con 500 puntos de operación.</i>	<i>80</i>

II. RESUMEN

En el presente trabajo de tesis se hace la comparación de distintos tipos de redes neuronales artificiales (RNA) aplicadas al diagnóstico de turbinas de gas. Se utilizan dos redes básicas: redes de retropropagación y redes radiales. Dentro de las redes de retropropagación, 12 funciones de entrenamiento fueron utilizadas; por el lado de redes de base radial (RBR) se utilizaron 3 redes particulares.

Los algoritmos, para realizar y comparar estas redes, están codificados en un ambiente de programación de Matlab que tiene una caja de herramientas (toolbox) de redes neuronales artificiales, la cual contiene las funciones de entrenamiento y las mencionadas redes radiales. Para hacer el ambiente de experimentación más rápido y efectivo, fue creada una interface usuario. La interface utilizada durante la ejecución de toda la experimentación.

Los algoritmos fueron diseñados para calcular la influencia de distintos factores en el entrenamiento y la exactitud de las redes. Entre ellos se analiza el incremento del número de puntos de simulación, el cambio de régimen de operación, el cambio en los números de inicialización de generadores aleatorios, etc.

Se hace una comparación en el desempeño de las funciones perceptrón multicapa (retropropagación), con los criterios de la probabilidad de la diagnosis correcta de falla del motor. El entrenamiento con función de propagación resiliente (trainrp) ha sido considerado como la mejor.

También fueron probadas 3 RBR. Se investigó la influencia de los factores mencionados y se determinaron las mejores variaciones de cada red.

Finalmente, el perceptrón y las 3 RBR fueron comparadas. El principio de esta comparación consiste en que, en cada cálculo comparativo, se entrenan las 4 redes y se validan en las mismas condiciones de operación del motor y de su diagnóstico. La probabilidad de diagnóstico correcto se calcula para cada red y se comparan utilizando la probabilidad como criterio. Tal cálculo comparativo se repite con distintas variantes, para llegar a las conclusiones generales acerca de la eficacia de las redes. Al final se dan las recomendaciones para seleccionar la mejor red en los sistemas de diagnóstico reales.

III. ABSTRACT

This thesis is focus on different types of artificial neural networks (ANN) comparison applied to the diagnosis of gas turbines. Two networks: backpropagation network and radial basis networks were used. In backpropagation networks, 12 performances of training were used; on the side of radial basis networks (RBR) three networks were used.

The algorithms to place and compare these networks are encoded in a programming environment Matlab that has a toolbox of artificial neural networks, which contains the functions of the above networks. To make the experimental environment faster and more effective was created a user interface. The interface was used during the execution of any experimentation.

The algorithms were designed to calculate the influence of different factors in the training and the accuracy of the networks. Among them are increasing the number of points of simulation, the operating regime change, the change in the numbers of random generator initialization, etc.

A comparison is made in the performance of the functions multilayer Perceptron (backpropagation), the criteria of probability of correct diagnosis of engine failure. Training with resilient propagation function (trainrp) has been considered the best.

RBR also were tested in March. The influence of these factors and determined the best variations of each network.

Eventually, the perceptron and the three RBR were compared. The principle of this comparison is that, in each calculation of comparison, the four networks are trained and validated on the same engine operating conditions and their diagnosis. The probability of correct diagnosis for each network is calculated and compared using the likelihood criterion. Such comparative calculation is repeated with variations, to reach general conclusions about the effectiveness of networks. At the end there are recommendations for selecting the best network in the real diagnostic systems.

IV. JUSTIFICACIÓN

Los motores de turbina de gas han tomado gran importancia en la actualidad, ésta industria está en constante crecimiento e investigación y requiere ser confiable. Sin embargo, son máquinas con varios miles de componentes, ello disminuye potencialmente su fiabilidad. Si se tienen sistemas de monitoreo y pronóstico estos incrementan su fiabilidad.

Existen tres etapas de todo proceso de diagnóstico: monitoreo, diagnóstico detallado (reconocimiento de las fallas del motor) y pronóstico. Este trabajo de tesis está enfocado a la segunda etapa, diagnóstico detallado. En la actualidad, los métodos de diagnóstico se han convertido en populares, ya que permiten saber con antelación el estado del motor. Ello conduce la reducción de los costos de mantenimiento y previene los efectos negativos de falla imprevista de los motores.

Los métodos de diagnóstico se basan en distintos algoritmos de reconocimiento de patrones, en particular, probabilísticos, basados en el teorema de Bayes, árboles de decisión y redes neuronales. Estos métodos, aplicados al diagnóstico de turbinas demuestran ser capaces de clasificar y diagnosticar las fallas de turbina de gas en forma adecuada.

El objetivo de este trabajo de tesis, dedicado a la investigación de redes neuronales, es saber cuál es la red que tiene mejor desempeño, que es capaz de clasificar y diagnosticar con altos índices de confianza. Si se conoce cuál es la red con mayor rango de confianza será posible implementar ésta a un sistema real de diagnóstico automatizado.

V. INTRODUCCIÓN

Una turbina de gas es una máquina térmica que opera bajo el principio del ciclo Joule-Brayton, convierte la energía, producto de la combustión de una mezcla de aire comprimido y combustible, en energía útil (trabajo). Éstas constituyen la forma más barata en la producción de grandes cantidades de energía y el objetivo principal de cualquier empresa es tener amplia disponibilidad de energía en forma confiable, segura y barata.

Por estas razones, las turbinas de gas son sometidas a estrictos planes de mantenimiento con el objetivo de reducir fallas en medida de lo posible. Al encontrarse en operación por largos periodos de tiempo, los componentes de turbina de gas están sometidos a periodos de fatiga, desgaste, corrosión y a las inclemencias climáticas, por lo que tarde o temprano una falla puede ocurrir.

Las reparaciones ocasionadas por fallos inesperados constituyen un impacto económico para cualquier empresa, por lo que gran parte del capital es destinado al mantenimiento y a la búsqueda de nuevas tecnologías que ayuden a mantener el motor en buen estado. Una parte integral del mantenimiento de las turbinas de gas es el monitoreo y el diagnóstico. A través de éstos se determina el estado de trabajo de la turbina de gas y se identifican fallas incipientes que alertan sobre la aparición de un fallo serio.

El poder alertar con antelación implica un método de diagnóstico complejo; además entre mayor sea el número de componentes, hay mayor probabilidad de falla. Esto nos lleva a requerir una técnica que sea capaz de detectar las fallas con suficiente antelación para evitar pérdidas mayores. La aplicación de sistemas automatizados de diagnóstico y monitoreo, los cuales han llegado a ser una práctica estándar, permiten menguar los efectos negativos causados por los procesos de deterioración.

El diagnóstico es una ciencia aplicada que implica conocer el motor, su comportamiento en estado normal, de este modo es posible identificar cuando tiene algún síntoma de falla. Para ello se aplican algoritmos basados en la teoría de reconocimiento de patrones, así como modelos avanzados de turbina de gas.

Las técnicas aplicadas son cada vez más complejas, por ende, más exactas. Parten desde el modelo termodinámico como base y una serie de datos obtenidos del motor, cuando éste fue nuevo, utilizando estos valores como base para la comparación con las lecturas de los datos actuales, hasta las redes neuronales. Las cuales después de una fase de entrenamiento, son capaces de clasificar las decisiones entre datos reales y el desempeño del motor nuevo, es decir, hacer un diagnóstico.

CAPÍTULO I
ESTADO DEL ARTE

Introducción

Los motores de turbinas de gas han evolucionando rápidamente en las últimas décadas. Aunque el ciclo básico de trabajo sigue siendo el mismo, en actualidad son muy sofisticados y tienen un gran número de componentes y subsistemas. Hoy en día estas son máquinas a las que se les exige fiabilidad elevada, más sin embargo, con el incremento del número de componentes, también se incrementa la probabilidad de alguna falla. Los sistemas automatizados de diagnóstico son capaces de prevenir fallas serias, pero sólo cuando sus algoritmos, en particular los de reconocimiento de fallas son bastante exactos.

1.1 Turbinas de gas

1.1.1 Historia

Aunque los principios de reacción fueron conocidos desde épocas A.C., fue hasta 1872, cuando Stolze diseñó la primera verdadera turbina de gas. Incorporaba una turbina de varias etapas y compresión en varias etapas con flujo axial probando modelos funcionales hasta los años 1900. En 1914 Charles Curtis aplicó para la primera patente en los Estados Unidos para una turbina de gas. Ésta fue otorgada, pero generó mucha controversia. La Compañía General Electric comenzó su división de turbinas de gas en 1903. Un ingeniero llamado Stanford Moss dirigió la mayoría de los proyectos. Su desarrollo más notable fue el turbo super-cargador, que utilizaba los gases de escape de un motor alternativo para mover una rueda de turbina que, a su vez, movía un compresor centrífugo utilizado para supercargar. Este dispositivo hizo posible construir las primeras turbinas de gas confiables.

En los años 30, tanto británicos como alemanes diseñaron turbinas de gas para la propulsión de aviones. Los alemanes alcanzaron a diseñar aviones de propulsión a chorro y lograron utilizarlos en la 2° guerra mundial (II GM)[1].

El primer motor fue ensayado en vuelo en 1938, suministrando menos empuje que el previsto, lo que obligó a incorporarle modificaciones. Más tarde, un motor de 360 kg, Heinkel He 5-3b, desarrolló 500 kg de empuje y constituyó la planta de poder del H-178, con la que el capitán Warsitz voló el 24 de agosto de 1939. Este fue el primer vuelo realizado por un avión propulsado por turborreactor que, luego de diversas pruebas, alcanzó a desarrollar una velocidad máxima de 700 km/h.

En las postrimerías de la II GM los alemanes asombraron nuevamente al mundo cuando en el otoño de 1944 apareció el primer caza de reacción construido en serie, el Me 262 provisto de dos turborreactores Jumo 004 de flujo axial y 900 kg de empuje c/u, que le permitían alcanzar una velocidad horizontal cercana a los 900 km/h. A partir de entonces comenzaron a manifestarse diversas tendencias en la notable evolución que experimentaron estas plantas de poder [2].

1.1.2 ¿Qué es una turbina de gas?

Existen muchas definiciones que se basan en el ámbito de aplicación del concepto, pero en general se puede decir que una turbina de gas es un motor térmico rotativo de flujo continuo con una baja relación peso-potencia y velocidad de giro muy elevada [3]. La

turbina es impulsada por los gases de combustión de una mezcla comprimida de algún combustible y aire, en general es utilizada para generación de energía [4].

Así se puede uno dar cuenta de que el término no solo refiere al componente de turbina, sino al motor en general. Un motor es una máquina capaz de transformar la energía almacenada en combustibles, baterías u otras fuentes, en energía mecánica capaz de realizar un trabajo. El término “térmico” se refiere a la obtención del trabajo a partir de energía térmica. “Rotativo” porque su principio de funcionamiento es mover los discos alrededor de un eje y “de flujo continuo” porque a diferencia de los motores que trabajan en ciclo Otto el flujo tanto de entrada como de salida al motor es constante; es decir, no hay interrupciones en la entrada de aire ni en la salida de potencia.

La turbina de gas se caracteriza por una elevada velocidad de giro que en función del tamaño puede llegar a alcanzar valores de 40000 revoluciones por minuto, con velocidades en la punta de los discos cercanas a la velocidad del sonido ($Mach = 1$). Su utilización se orienta a una unidad de generación de gases con elevada entalpía que puede utilizarse para propulsión a reacción o ser la encargada de accionar una turbina de potencia acoplada a un eje. A este eje puede articularse cualquier tipo de carga [5], que permita la producción de trabajo mecánico útil, al utilizar la energía proveniente de los gases de combustión. En cuanto a la relación peso potencia, se obtiene por pesos muy pequeños de motor una cantidad inmensa de potencia.

1.1.3 Principales componentes

La turbina de gas está formada por dos elementos principales:

- el generador de gases;
- la unidad generadora de potencia.

El generador de gases está formado a su vez por uno o varios compresores, la cámara de combustión y finalmente la o las turbinas de expansión de gases, que obtendrán la potencia necesaria para mover los compresores. La unidad generadora de potencia es en donde se obtendrá la potencia útil de la maquina, dependiendo de la aplicación, será otra turbina de expansión de gases, o bien, una tobera de propulsión [5].

Dicho de otra manera, los elementos que componen el ciclo de una turbina de gas, la turbina está formada por un compresor, una cámara de combustión y la turbina propiamente dicha (figura 1.1).

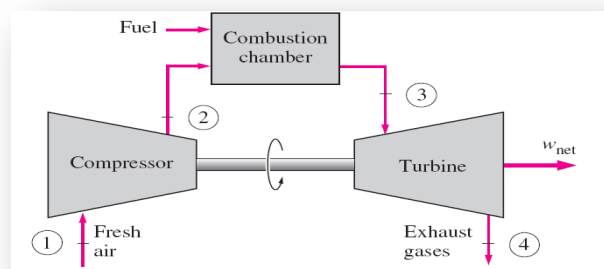


Figura 1.1 Esquema de una turbina de gas.

1.1.4 Funcionamiento

El modelo termodinámico de las turbinas de gas corresponde al ciclo Joule Brayton (figura 1.2), aunque se clasifica como un ciclo termodinámico, en realidad el fluido de trabajo no completa el ciclo ya que acaba con una composición o estado diferente al que empezó. Pero para efectos de análisis se considera un ciclo ideal y por ende corresponde al ciclo antes mencionado [3].

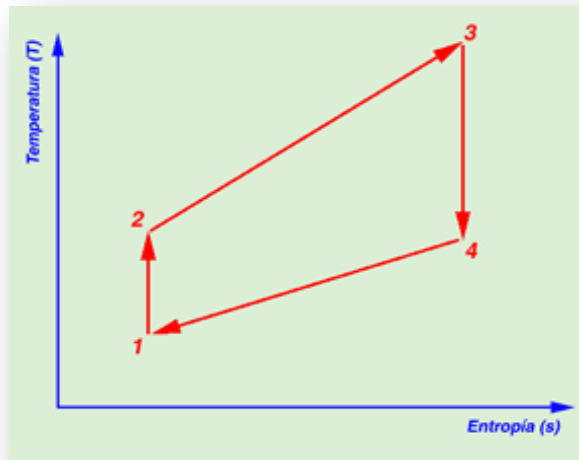


Figura 1.2 Ciclo de operación Joule Brayton..

El ciclo básico de Brayton en condiciones ideales está compuesto por cuatro procesos:
1-2, compresión isentrópica (proceso adiabático reversible) en un compresor;
2-3, adición de calor al fluido de trabajo a presión constante en un intercambiador de calor o una cámara de combustión;
3-4, expansión isentrópica en una turbina;
4-1, sustracción del calor del fluido de trabajo a presión constante en un intercambiador de calor o en la atmósfera.

Recordando la figura 1.1, podemos integrar el ciclo Joule Brayton a la secuencia de procesos físicos en la turbina, la misma figura ayuda en la explicación física y termodinámica de lo que ocurre dentro del ciclo térmico y de la máquina. Entonces se tiene de 1-2 será una compresión isentrópica del fluido de trabajo dentro de un compresor, pues al ir disminuyendo el volumen físico conservando la cantidad de masa, aumentamos la presión, al mismo tiempo se aumenta la temperatura y se entrega a la cámara de combustión. En la cámara de combustión se agrega el combustible propiciand así la combustión y por ende aumentando la energía contenida, la cual es proyectada hacia la turbina. La turbina extrae la energía cinética y la transforma en trabajo que se entrega a un eje, ésta energía no es absorbida totalmente en una sola fase, sino que requiere de una expansión isentrópica paulatina hasta que se le extrae la mayor cantidad de energía posible. En el último proceso, la energía que no se puede aprovechar se desecha a la atmósfera o a un sumidero en forma de calor.

1.2 Diagnóstico de turbinas de gas

1.2.1 ¿Qué es el diagnóstico?

El término diagnosis proviene del griego y significa reconocimiento o determinación. El diagnóstico en nuestro caso de estudio es el procedimiento por el cual se identifica o reconoce una condición de estado de un motor (sin falla o con ella). El objetivo que tiene es el aumento de la confiabilidad en el tiempo de trabajo de la turbina [7].

Este proceso se puede llevar a cabo en diferentes etapas de ciclo de vida del motor como son diseño, producción y mantenimiento; en base a ello se dan tareas concretas al diagnóstico. En el caso del diseño, el diagnóstico paramétrico se utiliza para encontrar defectos primarios, secundarios y evitar la destrucción del motor piloto, además es necesario formar las recomendaciones para el mejoramiento óptimo del motor. Durante la producción se confirma el diseño; en esta etapa el diagnóstico paramétrico ayuda a las pruebas del motor y verifica estado sin falla; también se forman las recomendaciones de ajuste y cambio de piezas estropeadas; durante tal proceso se analiza una serie de motores en producción [7]. En mantenimiento, el cual es la principal aplicación del diagnóstico, se monitorea el estado del motor localizando las fallas tempranamente, determinando las causas y se forman recomendaciones para eliminar tales fallas; se hace una acumulación de material estadístico para el mejoramiento futuro del motor y del sistema.

1.2.2 Etapas del diagnóstico

Una de las formas de diagnosticar motores es utilizando los parámetros medidos en el motor en funcionamiento, estos métodos son llamados métodos paramétricos, los cuales pueden ser clasificados según el sistema a ser diagnosticado, por ejemplo diagnosis en el conducto de flujo, diagnosis de las vibraciones, diagnosis del sistema de aceite, diagnosis de combustible, etc.

Aunque se hace la separación de la clasificación, todas y cada una de ellas pasan por las mismas etapas de proceso que son: monitoreo, diagnóstico y pronóstico. El monitoreo es la etapa del proceso de diagnóstico en la cual se hace la recolección de datos, se describen las clases que en general pueden ser dos, con falla y sin falla, y se explica cuál fue el criterio de clasificación. Durante la segunda etapa se toma la decisión diagnóstica que implica la búsqueda de la clase que está más cerca del estado actual del motor. Éste reconocimiento se hace en el espacio multidimensional con uso de las mediciones, las cuales contienen la información del estado actual del motor. Por último se aplica una característica de autenticidad de la decisión acorde al enfoque elegido, que puede permitir o no la intersección “entre clases”, es decir, que una decisión actual puede pertenecer, o no, a más de un grupo.

1.2.3 Diagnóstico en la actualidad

Las turbinas de gas son un sistema muy complejo, entre mayor sea el número de componentes del sistema, hay mayor probabilidad de falla. Ello nos lleva a requerir una técnica que sea capaz de poder detectar las fallas con antelación en cada componente y así evitar pérdidas mayores.

Las fallas y los procesos de deterioración afectan no sólo la fiabilidad del motor, si no su disponibilidad y los costos de operación. La aplicación de sistemas de diagnóstico y monitoreo, los cuales han llegado a ser una práctica estándar, permiten menguar estos efectos negativos.

El diagnóstico implica conocer el comportamiento del motor en estado normal. De este modo es posible identificar cuando tiene algún síntoma de falla. Para ello se aplican cálculos del estado normal del motor (estado inicial) y desviaciones (patrones) inducidas por las fallas, así como algunas técnicas de reconocimiento de patrones y modelos avanzados de probabilidad que permitan identificar las desviaciones.

Es claro que se puede implementar una fusión entre un sistema de procesamiento de información de datos de vibración, parámetros de desempeño y mediciones relativas, detectando así las fallas con muy alta precisión. Los sensores son de gran importancia, permiten desde la toma de lecturas hasta poder llegar a una validación, para poder dar un pronóstico [8].

Las técnicas aplicadas son cada vez más exactas, por ende más complejas. Parten desde tener el modelo termodinámico como base y una serie de datos obtenidos del motor, cuando este fue nuevo, utilizando estos valores como base para comparación con las lecturas de los datos actuales; hasta el entrenamiento de redes neuronales. Ellas, después de una fase de entrenamiento son capaces de identificar las fallas y son capaces de dar un pronóstico [9].

1.2.4 Situación del diagnóstico y las turbinas de gas en México

La industria energética ha evolucionado mucho a nivel mundial. En México, aunque con un poco de retraso, también lo ha hecho, de tal manera que las turbinas de gas como parte fundamental en esta industria son cada vez más utilizadas, pero también cada vez más complejas. Esta complejidad deriva en mayor número de componentes que requieren sistemas más avanzados de diagnóstico. Los sistemas de diagnóstico permiten al usuario, operador, o cualquiera que tenga algún qué hacer con estos motores, tener una indicación de en qué momento el motor necesita mantenimiento y/o reparación. Entonces, teniendo un control y programación del desgaste y tiempo aproximado de falla de los componentes, estos se hacen más confiables, al mismo tiempo que se reducen costos de mantenimiento.

Actualmente en México existe una gran cantidad de empresas que ocupan las turbinas de gas como componente fundamental en su función, pues son utilizadas para la producción de energía eléctrica, transporte de gas, o empuje mismo. Algunas de las empresas con mayor impacto en México son: Comisión Federal de electricidad, Luz y Fuerza del Centro, Petróleos Mexicanos y (por su puesto las empresas que brindan servicios aéreos) Mexicana de Aviación, Aeroméxico y Aviacsa por mencionar algunas.

La mayoría de estas empresas no realizan los servicios de mantenimiento de tercer escalón a su propia maquinaria, por lo general son empresas de servicios las que se dedican al mantenimiento, reparaciones mayores, reingeniería, reemplazo de componentes y rehabilitación.

Existen muchas empresas que se dedican, además de prestar estos servicios, al rediseño de componentes, re-potenciación y modernización de los sistemas. Por mencionar algunas de estas empresas esta ITR, Turbomaquinarias S.A. de C.V., especialistas en turbo partes S.A. de C.V., entre otras.

También existen institutos orientados a la investigación y capacitación de personal, como el Instituto de Investigaciones Eléctricas, el Centro de Investigaciones Avanzadas Tecnológicas de Querétaro, entre sus funciones esta el brindar apoyo en la investigación de desarrollo de métodos y herramientas para el diagnóstico de fallas, la rehabilitación y extensión de la vida útil de los componentes de turbomaquinaria.

Independientemente del campo de acción de cada una de las empresas mencionadas anteriormente, se sabe que todas ellas cuentan con sistemas de diagnóstico para hacer más eficiente su labor.

1.3 Reconocimiento de patrones

De las explicaciones anteriores se deduce que es en el área de la identificación de las fallas de turbina de gas, en donde se presenta un área de aplicación de la teoría de reconocimiento de patrones. Algunos conceptos de esta teoría se dan a continuación.

Objeto. Es un concepto con el cual representamos los elementos sujetos a estudio, pueden ser concretos o abstractos; para nuestro caso el objeto son las fallas que se presentan en la turbina, descritas en un espacio multidimensional.

Patrón. Es sinónimo de objeto, en ocasiones se le llama así a los objetos ya clasificados, los tipos de fallas que se presentan en un motor pueden ser determinadas como patrones, o comportamiento.

Rasgo. Propiedad, factor, característica, etc. que se toma en cuenta para estudiar los objetos. Existen dos tipos:

- esenciales, que no pueden ser eliminados de la descripción de los objetos sin confundirlos;
- accidentales, que pueden ser ignorados en una descripción y los objetos no se confunden.

Clase. Es un conjunto de objetos. Al agrupar en clases, se puede hacer de dos formas distintas:

- por pertenencias duras, un objeto pertenece o no a una clase;
- por pertenencias difusas, los objetos pertenecen parcialmente a una clase, existen clases con intersecciones no vacías, es decir, que puede haber más de una falla, o más de un componente en estado defectuoso.

Reconocimiento. Proceso de clasificación de un objeto en una o más clases.

Filtración. Consiste en quitar información o datos indeseados de la entrada dependiendo del uso, el algoritmo o método de filtrado; es decir, la eliminación de ruido o datos

causados por falla de los sensores o que simplemente no aportan ningún dato valioso para ser analizado.

Reconocimiento de patrones. Existen varios intentos para definir el reconocimiento de patrones, algunos son:

- “La disciplina dedicada a la clasificación de objetos y el pronóstico de fenómenos”.
- “Rama del conocimiento, de carácter multidisciplinario, cuyo objeto de estudio son los procesos de identificación, caracterización, clasificación y reconstrucción sobre conjuntos de objetos o fenómenos, así como el desarrollo de teorías, tecnologías y metodologías relacionadas con dichos procesos”.
- “Es la ciencia que se ocupa de los procesos sobre ingeniería, computación y matemáticas relacionados con objetos físicos y/o abstractos, con el propósito de extraer información que permita establecer propiedades de o entre conjuntos de dichos objetos”.

1.3.1 Antecedentes Históricos

Principios de 1960, Eden, Narasimham y Ledley publicaron los primeros trabajos. En 1966, Zhuravliov hace el primer trabajo que habla de la Teoría de Testores y el Reconocimiento de Patrones. A partir de este momento, el reconocimiento de patrones es un objeto de estudio más serio. En 1972 se realiza la primera conferencia sobre el reconocimiento de patrones en Washington. En 1974 se realiza la segunda conferencia sobre el reconocimiento de patrones en Copenhague. En 1917 Vapnik y Chervonenkis escriben el libro Reconocimiento de Patrones. En éste libro mencionan: “En esencia, diferentes puntos de vista en la formulación del problema del reconocimiento de patrones se determinan por la respuesta a la pregunta: “¿Hay principios generales para describir clases de patrones de diversas naturalezas?”. En enero de 1978 se funda la IAPR (International Association for Pattern Recognition). En 1980 llega a México el reconocimiento de patrones a través de maestros cubanos que vienen a enseñar.

1.3.2 Tipos de problemas de clasificación

El punto esencial del reconocimiento de patrones es la clasificación: se quiere clasificar una señal dependiendo de sus características. El objetivo es clasificar patrones con base en un conocimiento a priori o información estadística extraída de los patrones. Los patrones a clasificar suelen ser grupos de medidas u observaciones, definiendo puntos en un espacio multidimensional apropiado.

La clasificación utiliza habitualmente uno de las siguientes procedimientos: clasificación estadística (teoría de la decisión) o clasificación sintáctica (estructural). El reconocimiento estadístico de patrones está basado en las características estadísticas de los patrones, asumiendo que han sido generados por un sistema probabilístico. El reconocimiento estructural de patrones está basado en las relaciones estructurales de las características [10]. Otros tipos de clasificación son:

Clasificación supervisada: También es conocida como clasificación con aprendizaje, en este tipo de problemas ya se encuentran definidas las clases y cuentan con algunos objetos previamente clasificados.

Clasificación parcialmente supervisada: También conocida como de aprendizaje parcial, en éstos problemas existe una muestra de objetos sólo en algunas de las clases definidas.

Clasificación no supervisada: También conocida como clasificación sin aprendizaje, en éstos problemas no existe ninguna clasificación previa de objetos y en algunas ocasiones ni siquiera se han definido las clases.

Un sistema de reconocimiento de patrones completo consiste en un sensor que recoge las observaciones a clasificar, un sistema de extracción de características que transforma la información observada en valores numéricos o simbólicos, y un sistema de clasificación o descripción que, basado en las características extraídas, clasifica la medición.

Para la clasificación se puede usar un conjunto de aprendizaje, del cual ya se conoce la clasificación de la información a priori y se usa para entrenar al sistema, siendo la estrategia resultante conocida como aprendizaje supervisado. El aprendizaje puede ser también no supervisado, en donde el sistema no tiene un conjunto de patrones para aprender a clasificar la información a priori, sino que se basa en cálculos estadísticos para clasificar los patrones [10].

Entre las aplicaciones del reconocimiento de patrones están el reconocimiento de voz, la clasificación de documentos (por ejemplo spam/no spam), el reconocimiento de escritura, reconocimiento de caras humanas y muchas más. Los dos últimos ejemplos son representativos del análisis de imágenes, un subconjunto del reconocimiento de patrones que toma imágenes digitales como entradas del sistema.

1.3.3 Enfoques dentro del reconocimiento de patrones

Geométrico (Clustering): Los patrones deben ser graficables, en éste enfoque se emplea el cálculo de distancias euclidianas, geometría de formas, vectores numéricos, puntos de atracción, etc.

Estadístico: Se basa en la teoría de la probabilidad y la estadística, utiliza análisis de varianzas, covarianzas, dispersión, distribución, etc.

Sintáctico-estructural: Estudia la estructura de los objetos, es decir, usa teoría de lenguajes formales, gramáticas, teoría de autómatas, etc.

Neuro-reticular: Se utilizan redes neuronales que se ‘entrenan’ para dar una cierta respuesta ante determinados valores.

Lógico-Combinatorio: Se basa en la idea de que la modelación del problema debe ser lo más cercana posible a la realidad del mismo, sin hacer suposiciones que no estén fundamentadas. Se utiliza para conjuntos difusos y utiliza lógica simbólica, circuitos combinatoriales y secuenciales, etc.

1.3.4 Fundamentos Matemáticos

El reconocimiento de patrones no es una teoría aislada, sino que se basa en muchos métodos matemáticos. Los métodos principales inmersos dentro de problemas de clasificación se describen a continuación.

1.3.4.1 Teoría de decisión de Bayes

La esencia de la aproximación Bayesiana es proveer una regla matemática de cómo se deben cambiar las creencias existentes en vista de una evidencia. En otras palabras, esta regla permite combinar nuevos datos con el conocimiento previo.

El ejemplo clásico es imaginar que un niño recién nacido observa por primera vez una puesta de sol y se pregunta si el sol saldrá de nuevo o no. Se asigna la igualdad de probabilidades a priori a los dos posibles resultados y se representa éstos colocando una pelota blanca y negra en una bolsa. Al día siguiente, cuando sale el sol, el niño pone otra pelota blanca en la bolsa. La probabilidad de sacar una pelota blanca de la bolsa se incrementa (es decir, el grado del creencia del niño en amaneceres futuros), ha pasado de una mitad a dos tercios. Después de la salida del sol al día siguiente, el niño añade otra pelota blanca, y la probabilidad (y por tanto el grado de creencia) va de dos tercios a tres cuartos y así sucesivamente. Poco a poco, la creencia inicial de que el sol salga es tan probable como levantarse cada mañana; así esa probabilidad es modificada para convertirse en una certidumbre casi total de que el sol siempre saldrá [11].

El teorema de Bayes es esencialmente una expresión de probabilidades condicional. Más o menos, las probabilidades condicionales representan la probabilidad de que ocurra un evento dada cierta evidencia. Para mejor entendimiento el teorema de Bayes (ecuación 1) puede ser derivado de una probabilidad conjunta de A y B ($P(A, B)$), en donde $P(A, B) = P(B, A) = P(A|B)P(B) = P(B|A)P(A)$.

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)} \dots\dots\dots(1)$$

en donde $P(A|B)$ es conocida como la *posterior*; $P(B|A)$ es conocida como vecindario, $P(A)$ es la *priori* y $P(B)$ es generalmente al evidencia y es utilizado como factor de escalamiento. Entonces, es útil recordar la regla de Bayes como:

$$Posterior = \frac{Vecindario \times priori}{evidencia}$$

Si se tiene el caso en donde A es más de un evento, $A=\{A_1, A_2, \dots, A_n\}$ entonces la probabilidad va a estar dada por:

$$P(A_i|B) = \frac{P(B|A_i)P(A_i)}{\sum_{i=1}^n P(B|A_i)P(A_i)} \dots\dots\dots(2)$$

Observe la figura 1.3. Con la regla de Bayes es posible determinar la probabilidad de que X pertenezca a A_1 o A_2 . En este ejemplo es obvio que X pertenece a A_2 , sin embargo, la

respuesta correcta para muchos eventos reales usualmente no es tan clara como este ejemplo.

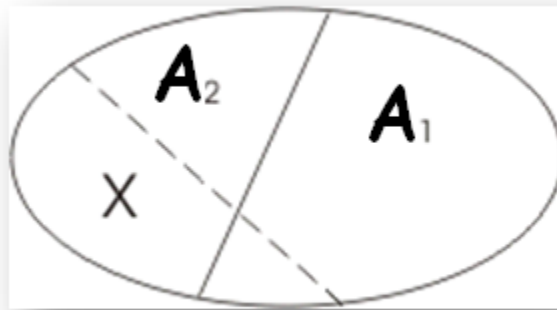


Figura 1.3 Ejemplo para la teoría de Bayes.

La regla de Bayes ayuda a determinar la probabilidad, mientras que la regla de decisión de Bayes ayuda a hacer la decisión más óptima basada en conocimiento. La regla de decisión dice que se clasifica en A_j si:

$$P(A_j|B) = \max_{i=1,n} P(A_i|B) \dots\dots\dots(3)$$

1.3.4.2 Clasificadores Gaussianos

El modelo de Gauss se puede utilizar para caracterizar un grupo de vectores característicos de cualquier número de dimensiones con dos valores: un vector de medias y una matriz de covarianza. La clave es entender el modelo de Gauss como modelo probabilístico. El modelo de Gauss es una manera de calcular P (observación | tipo). Cada tipo se caracteriza por una sola gaussiana y la probabilidad se calcula midiendo la altura de la curva (o más generalmente el valor de la función) en las coordenadas especificadas por el vector de entrada. Ésta medida de probabilidad condicional se convierte en una probabilidad a posteriori a través de la fórmula de Bayes utilizando las probabilidades a priori de la observación y el tipo [12].

La probabilidad posterior se puede utilizar en la misma forma que la distancia Euclidiana como medida que se desarrollaron antes de comparar a un desconocido con una serie de modelos. Sin embargo, ahora los modelos son modelos Gaussianos en lugar de puntos únicos y la probabilidad será mayor si la incógnita está cercana al modelo. La decisión entre los tipos se pueden hacer encontrando el modelo para el cual la medida de probabilidad es más grande, como se demuestra en la ecuación 4 [12].

$$P(A_i|Y_1) = \frac{f(Y|A_i)P(A_i)}{\sum_{i=1}^n f(Y|A_i)P(A_i)} \dots\dots\dots(4)$$

Esto se ilustra en la Figura 1.4 que muestra un ejemplo de una dimensión.

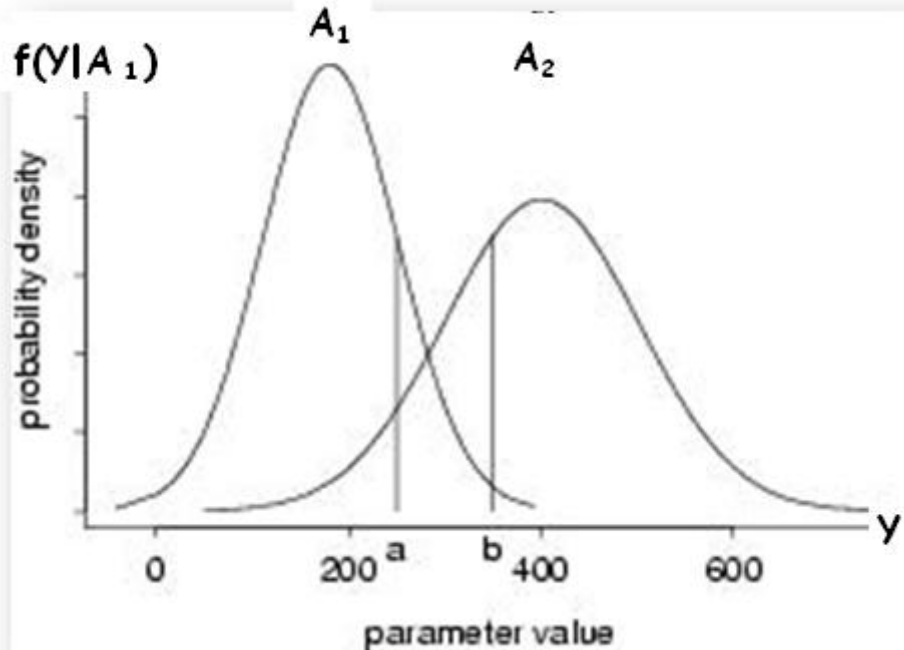


Figura 1.4 La densidad de probabilidad condicional para dos tipos en un solo parámetro. Punto 'a' sería asignado a tipo A_1 , mientras que el punto 'b' se asignarían a A_2 .

El uso de un modelo de Gauss nos da una solución al segundo problema de tomar en cuenta la forma de la distribución de los vectores. Ésta distribución está codificada en la media y la covarianza de la curva de Gauss, por lo que el cálculo de probabilidades tendrá en cuenta de esto.

1.3.4.3 Árboles de decisión

Los árboles de decisión son clasificadores con estructura de árbol, en donde cada nodo es un nodo hoja, el cual indica el valor de atributo de un objetivo (clase), o un nodo de decisión, que especifica algún test para ser llevado a cabo en un simple valor de atributo, con una rama y subárbol para cada posible resultado de la prueba.

Un árbol de decisión puede ser usado para clasificar, empezando a clasificar desde la raíz y moviéndose a través de él, tomando una decisión en cada hoja, lo cual provee una clasificación. Este método es típicamente una aproximación inductiva para aprender los conocimientos sobre la clasificación. La figura 1.5 muestra un ejemplo básico de árbol de decisión [13].

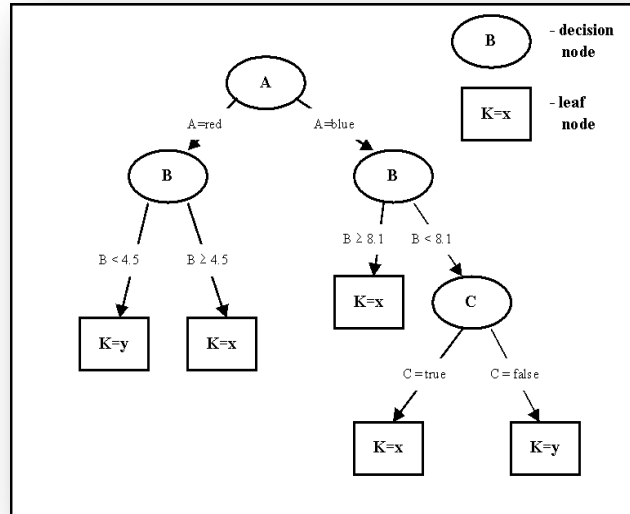


Figura 1.5 Ejemplo de árbol de decisión simple.

1.3.4.4 K vecinos más cercanos

El algoritmo de k-vecinos más cercanos (k nearest neighbor “*k*-NN”) es un método para clasificar los objetos en función de la formación de elementos más cercanos al espacio de características. Los ejemplos de entrenamiento son vectores en un espacio de características multidimensionales, cada uno con una etiqueta de clase. La fase de entrenamiento del algoritmo consiste solamente en almacenar los vectores de características y etiquetas de clase de los ejemplos de entrenamiento.

En la fase de clasificación, *k* es una constante definida por el usuario, y un vector no etiquetados (una consulta o un punto de prueba); está clasificada por la asignación de la etiqueta que es más frecuente entre la formación *k* muestras más cercanas a ese punto de la consulta [14].

Por lo general, la distancia Euclidiana se utiliza como la métrica de distancia, sin embargo esto sólo es aplicable a las variables continuas. En casos como el de clasificación de textos se puede utilizar otra métrica como la superposición métrica (o distancia de Hamming). A menudo, la exactitud de la clasificación de “*k*-NN” se puede mejorar significativamente si la métrica a la distancia se aprende con algoritmos especializados, como por ejemplo grandes márgenes de vecino más cercano o análisis de componentes de vecindad [14].

Una desventaja en la clasificación básica es que las clases con los ejemplos más frecuentes tienden a dominar la predicción del nuevo vector, ya que tienden a surgir en los *k* vecinos más cercanos cuando los vecinos se calculan debido a su gran número. Una forma de superar este problema es el peso de la clasificación teniendo en cuenta la distancia desde el punto de prueba para cada uno de sus *k* vecinos más cercanos. *k*-NN es un caso especial de un ancho de banda variable, la densidad del núcleo funciona como estimador con un núcleo uniforme [14].

1.3.4.5 Support Vector Machine (SVM)

Es un conjunto de algoritmos de aprendizaje supervisado, usados para la clasificación y regresión. En este tipo de algoritmos se visualiza la entrada de datos como dos conjuntos de vectores en un espacio n -dimensional, así un SVM construirá una separación en hiperplano en ese espacio, que maximicen el margen entre dos conjuntos de datos. Para calcular el margen, se construyen dos líneas paralelas una a cada lado de la frontera del clasificador dando dos hiperplanos, positivo y negativo respectivamente [15].

Se considera un buen clasificador porque cuenta con un margen que crece antes de chocar con un punto, siendo los vectores de soporte los puntos sobre los cuales se apoya el margen máximo, y lo de *machine* es por la máquina de aprendizaje automático de Vapnik.

Para explicar lo anterior, en forma matemática tenemos lo siguiente:

$$\text{Hiperplano positivo} = \{x: w \cdot x + b = +1\}$$

$$\text{Hiperplano negativo} = \{x: w \cdot x + b = -1\}$$

$$\text{Clasificamos } +1 \text{ si } w \cdot x + b \geq 1$$

$$-1 \text{ si } w \cdot x + b \leq -1$$

siendo nuestro campo de trabajo o universo a recorrer $-1 < w \cdot x + b < 1$

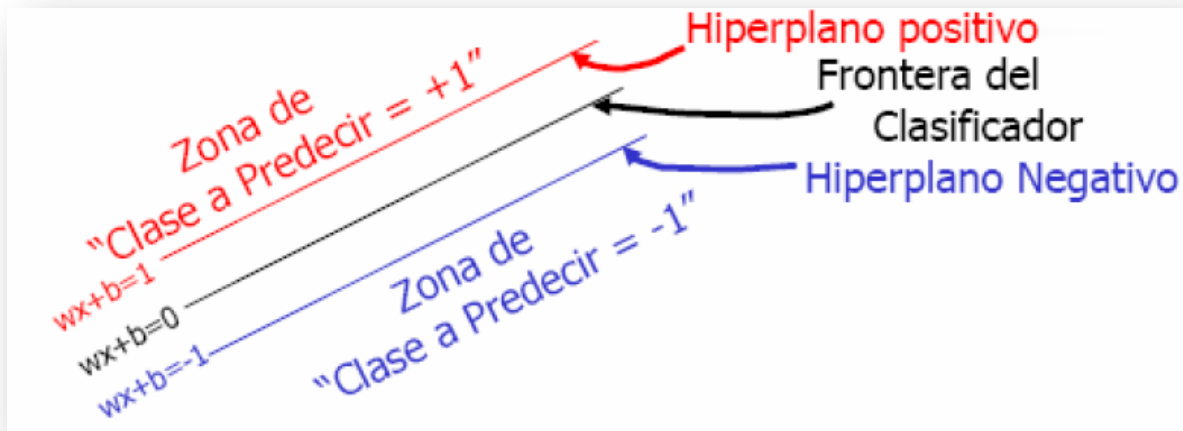


Figura 1.6 Clasificación por medio de SVM.

Existen muchos software que utilizan este método para la clasificación, reconocimiento y minería de datos, entre los que se encuentran SVM Toolbox Matlab, SVM ^{light} (más usado y utilizado para grandes bases de datos), LSVM (Lagrangian Support Vector Machine), ASVM (Active Support Vector Machine), PSVM (Proximal Support Vector Machine), etc. Todos ellos aplicando la técnica de "máquina de vectores de soporte" en diferentes ambientes, java, Matlab, C++, y algunos con ambiente propio.

1.3.4.6 Redes Neuronales Artificiales (bases)

El modelo de una neurona artificial es una imitación del proceso de una neurona biológica, puede también asemejarse a un sumador hecho con un amplificador operacional tal como se ve en la siguiente figura (1.7).

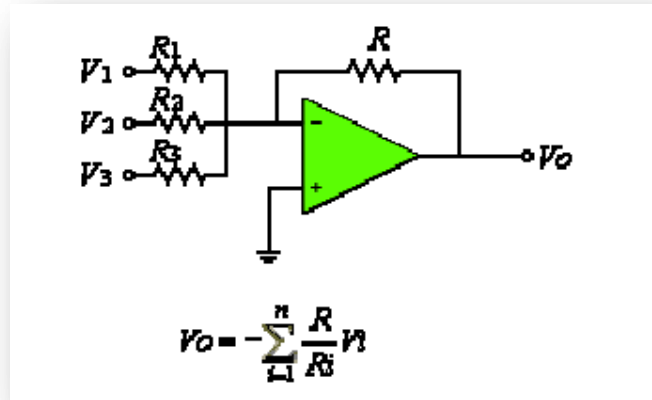


Figura 1.7. Sumador hecho con un amplificador operacional.

Cada señal de entrada pasa a través de una ganancia o peso, llamado peso sináptico o fortaleza de la conexión cuya función es análoga a la de la función sináptica de una neurona biológica. Los pesos pueden ser positivos (excitatorios) o negativos (inhibitorios), el nodo sumatorio acumula todas las señales de entradas multiplicadas por los pesos o ponderadas y la pasa a la salida a través de una función umbral o función de transferencia [18]. La entrada neta a cada unidad puede escribirse de la siguiente manera:

$$neta_i = \sum_{i=1}^n W_i X_i = \vec{X} \vec{Y} \dots \dots \dots (5)$$

En general las redes neuronales se pueden clasificar de diversas maneras: según su topología, forma de aprendizaje (supervisado o no supervisado), tipos de funciones de activación, o valores de entrada (binarios o continuos); un resumen de esta clasificación se observa en la siguiente figura (1.8) [18].

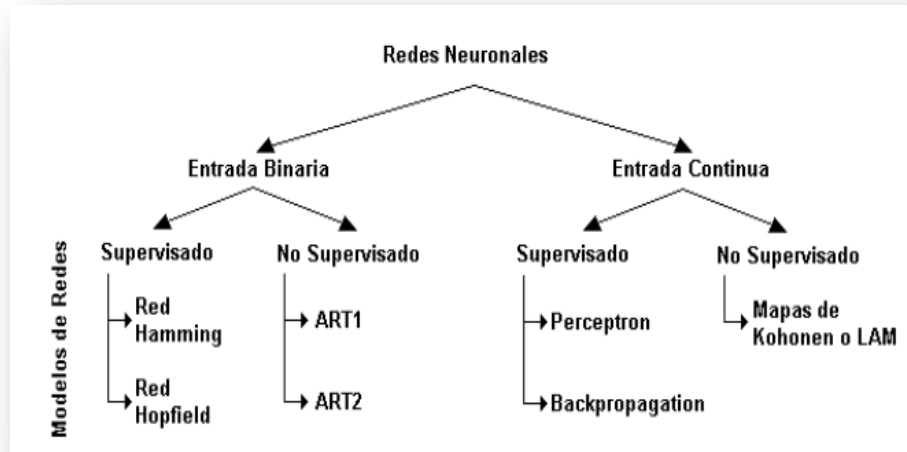


Figura 1.8 Clasificación de RNA.

1.3.5 Campos de aplicación

El reconocimiento de patrones es una técnica de la inteligencia artificial empleada por tecnologías como el procesamiento del lenguaje natural, la visión computacional y diagnóstico. El reconocimiento de patrones se apoya de otras técnicas de la inteligencia artificial como: lógica difusa, minería de datos y redes neuronales. Además se apoya en técnicas de otras ciencias como: estadística, geometría, teoría de lenguajes, lógica simbólica, entre otras.

Pese a los grandes conocimientos que tienen los diagnósticos, en muchas ocasiones se ven limitadas sus capacidades, es ahí cuando se necesitan herramientas de mayor precisión que les permitan realizar un análisis más profundo y detallado para tener diagnósticos más precisos y por lo tanto determinar el tratamiento correcto.

Con el uso del reconocimiento de patrones se pretende automatizar la identificación de patrones simples o estructurados y modelar diversos procesos perceptuales como lo son la visión y la audición [15].

El uso de reconocimiento de patrones se ha visto reflejado en terapias de lenguaje mediante el reconocimiento de la voz y sonidos, en las imágenes de rayos X, de positrones y de resonancia magnética nuclear (RMN), de ultrasonido (donde se desarrollan técnicas para mejorar la identificación y seguimiento de formas, como la actividad del feto en el útero) o la cirugía plástica, en la cual se puede generar una reconstrucción tridimensional de la parte del cuerpo a intervenir y así mostrar diferentes vistas al paciente y explicarle mejor en qué consistirá su tratamiento, el resultado final y las modificaciones que se le aplicarían durante una operación.

Hasta ahora no se ha utilizado ampliamente el reconocimiento de patrones en el diagnóstico de turbinas de gas, lo cual representa un área de estudio con alto potencial de crecimiento y desarrollo a futuro [15].

1.3.6 Algunos Desarrollos

Raymond Kurzweil nace en 1948 en Queens, New York. A los 15 años diseñó un sistema de reconocimiento de patrones en el cual una computadora analizaba patrones dentro de la música clásica para después componer sus propios trabajos basados en lo evaluado. Con ello ganó el primer lugar en el Concurso Internacional de Ciencia [15].

Posteriormente fundó su propia compañía donde creó una máquina de lectura, la cual reunía tres inventos: un software de reconocimiento óptico de caracteres, un dispositivo similar a un escáner y un sintetizador completo de texto a deletreo. Ésta máquina era capaz de detectar y corregir errores además de la pronunciación de palabras. La primera versión comercial del software fue usado por Lexis Nexis para crear su base de datos [15].

En Cuba se está desarrollando un proyecto llamado “Fundamento del Reconocimiento de Patrones: Programa Integral de Formación de Especialistas” en el cual se pretende hacer una proyección unificadora del reconocimiento de patrones y la formación de un programa de formación post-graduada.

Este proyecto surge por la necesidad de formar más especialistas en reconocimiento de patrones y minería de datos debido a la gran cantidad de problemas prácticos que son necesarios resolver y que han sido detectados en importantes ramas como la energía, la salud, la defensa, el deporte y otras. Problemas relacionados con clasificación, diagnóstico, pronóstico, determinación de factores distintivos, procesamiento de grandes volúmenes de datos (textos, imágenes, señales, páginas Web), detección de asociaciones entre datos, descubrimiento de regularidades, entre muchos otros [15].

Debido a varias razones es necesario un enfoque integral de la disciplina: muchos problemas prácticos han sido resueltos gracias a la interacción de áreas de trabajo aparentemente lejanas. La tendencia actual en las investigaciones muestra una mayor vinculación entre reconocimiento de patrones e investigaciones teóricas. Esto obedece a la necesidad de explorar nuevos horizontes en busca de herramientas más poderosas. En el ámbito microscópico, se ha trabajado en la reconstrucción tridimensional de núcleos celulares, en particular de la cromatina, que forma el núcleo en células animales, y en neuronas dopaminérgicas de la sustancia nigra de pacientes con mal de Parkinson [15].

Al mismo tiempo se desarrollaron las redes neuronales como herramienta en el reconocimiento de patrones. McCulloch y Pitts dieron origen a los modelos conexionistas definiendo formalmente la neurona en 1943 como una máquina binaria con varias entradas y salidas. Hebb, definió en 1949 conceptos muy importantes y fundamentales que han pesado en el campo de las redes neuronales, basándose en investigaciones psicofisiológicas. 1959, Widrow, Teoría sobre la adaptación neuronal, el Adaline (Adaptative Linear Neuron) y el Madaline (Multiple Adaline). Es la primera aplicación de las redes a problemas reales: filtros adaptativos para eliminar ecos en las líneas telefónicas. 1962, Roseblatt, el Perceptrón es un identificador de patrones ópticos binarios y salida binaria. Dio lugar a la regla de aprendizaje delta, que permitía emplear señales continuas de entrada y salida. 1969, Minsky y Papert, una seria crítica del Perceptrón que dada su naturaleza lineal tenía bastantes limitaciones, que provocó una caída en picada de las

investigaciones y una época gris para las redes neuronales. En la actualidad, gracias a diversos grupos de investigación repartidos por universidades de todo el mundo, las redes neuronales han alcanzado una madurez muy aceptable y se usan en todo tipo de aplicaciones entre las que podemos citar: reconocimiento de patrones, voz, vídeo, compresión de imágenes, estudio y predicción de sucesos muy complejos, aplicaciones de apoyo a la medicina, todo tipo de aplicaciones que necesiten el análisis de grandes cantidades de datos, etc.

Conclusiones

Conocer la historia y evolución de las turbinas nos da una idea de cuán compleja ha llegado a ser. Se hizo un recorrido por la historia y evolución del reconocimiento de patrones y las aplicaciones que éste puede llegar a tener. De igual manera se mostro cuál es la situación de México al respecto, tanto en el área de reconocimiento de patrones como en la utilización de los motores de turbina de gas. Las redes neuronales forman parte del reconocimiento de patrones, así que se dio un recorrido por las generalidades de las redes neuronales y sus fundamentos.

CAPÍTULO II
REDES NEURONALES APLICADAS
Y ALGORITMOS DE
ENTRENAMIENTO

Introducción

Resulta irónico pensar que máquinas de cómputo capaces de realizar 100 millones de operaciones en punto flotante por segundo, no sean capaces de entender el significado de las formas visuales o de distinguir entre distintas clases de objetos. Hoy en día se ha logrado desarrollar algoritmos que son capaces de entender tales conceptos, aunque todavía lo hacen muy por debajo del cerebro de los seres humanos. Los sistemas de computación secuencial son exitosos en la resolución de problemas matemáticos o científicos, en la creación, manipulación y mantenimiento de bases de datos, en comunicaciones electrónicas, en el procesamiento de textos, gráficos, etc., pero definitivamente tienen una gran incapacidad para interpretar el mundo.

Esta dificultad de los sistemas de cómputo que trabajan bajo la filosofía de los sistemas secuenciales, desarrollados por Von Neuman, han hecho que un gran número de investigadores centren su atención en el desarrollo de nuevos sistemas de tratamiento de información, que permitan solucionar problemas cotidianos, tal como lo hace el cerebro humano. Este órgano biológico cuenta con varias características deseables para cualquier sistema de procesamiento digital, tales como:

- Es robusto y tolerante a fallas, diariamente mueren miles de neuronas sin afectar su desempeño.
- Es flexible, se ajusta a nuevos ambientes por medio de un proceso de aprendizaje sin tener que programarlo.
- Puede manejar información difusa, con ruido o inconsistente.
- Es altamente paralelo, capaz de resolver gran cantidad de operaciones al mismo tiempo.

Basados en la eficiencia de los procesos llevados a cabo por el cerebro e inspirados en su funcionamiento, varios investigadores han desarrollado desde hace más de 40 años la teoría de la Redes Neuronales Artificiales (RNA) [16].

Las aplicaciones más exitosas de RNA son:

1. Procesamiento de imágenes y de voz.
2. Reconocimiento de patrones (que es el de nuestro interés)
3. Planeamiento
4. Interfaces adaptativas para sistemas hombre/máquina.
5. Predicción
6. Control y optimización
7. Filtrado de señales

Las RNA son una teoría que aun está en proceso de desarrollo, su verdadera potencialidad no se ha alcanzado todavía, aunque los investigadores han desarrollado potentes algoritmos de aprendizaje de gran valor práctico, las representaciones y procedimientos de que se sirve el cerebro, son aún desconocidas. Tarde o temprano los estudios computacionales del aprendizaje con RNA acabarán por converger a los métodos descubiertos por la evolución, cuando esto suceda, muchos datos empíricos concernientes al cerebro comenzarán súbitamente a adquirir sentido y se tornarán factibles muchas aplicaciones desconocidas de las redes neuronales.

2.1 Funcionamiento de una red biológica

El cerebro consta de un gran número de elementos llamados neuronas (aproximadamente 10^{11}) altamente interconectados (aproximadamente 10^4 conexiones por elemento). Estas neuronas tienen tres componentes principales: las dendritas, el cuerpo de la célula o soma y el axón. Las dendritas son el árbol receptor de la red, son fibras nerviosas que cargan de señales eléctricas al cuerpo de la célula y realiza la suma de éstas señales de entrada. El axón es una fibra larga que lleva la señal desde el cuerpo de la célula hacia otras. El punto de contacto entre un axón de una célula y una dendrita de otra célula es llamado sinapsis; la longitud de la sinapsis es determinada por la complejidad del proceso químico que estabiliza la función de la red neuronal. Un esquema simplificado de la interconexión de dos neuronas biológicas se observa en la figura 2.1.

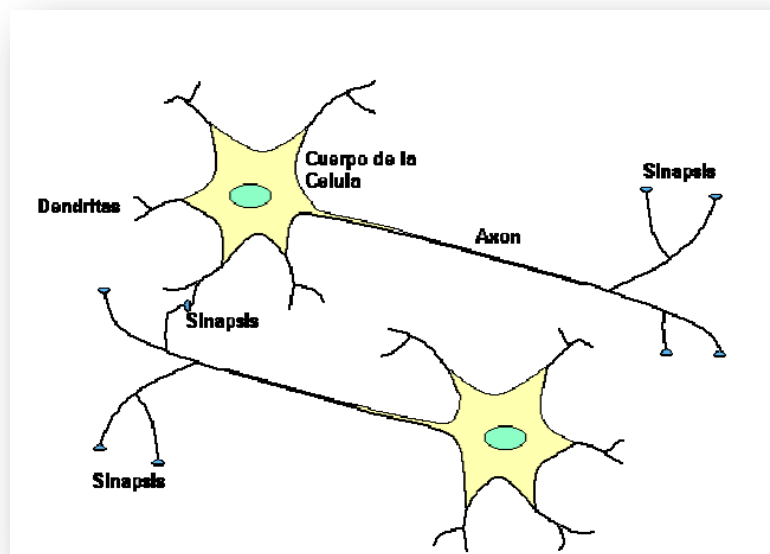


Figura 2.1 Interconexión de dos neuronas biológicas.

Algunas de las estructuras neuronales son determinadas en el nacimiento, otra parte es desarrollada a través del aprendizaje, proceso en que nuevas conexiones neuronales son realizadas, mientras que otras se pierden por completo. El desarrollo neurológico se hace crítico durante los primeros años de vida; por ejemplo, se ha demostrado que si a un cachorro de gato se le impide usar uno de sus ojos durante un periodo corto de tiempo, él nunca desarrollará una visión normal en ese ojo.

Las estructuras neuronales continúan cambiando durante toda la vida, estos cambios consisten en el refuerzo o debilitamiento de las uniones sinápticas; por ejemplo, se cree que nuevas memorias son formadas por la modificación de esta intensidad entre sinapsis, así el proceso de recordar el rostro de un nuevo amigo consiste en alterar varias sinapsis [18].

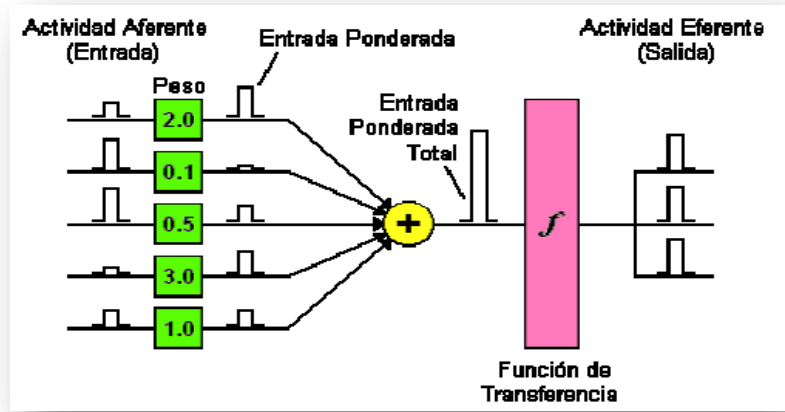


Figura 2.3 Proceso de “sinapsis” en una neurona artificial.

Una vez que se ha calculado la activación del nodo, el valor de salida equivale a:

$$x_i = f_i(neta_i) \dots \dots \dots (7)$$

Donde f_i representa la función de activación para esta unidad, que corresponde a la función escogida para transformar la entrada $neta_i$ en el valor de la salida x_i , y que depende de las características específicas de cada red [18].

2.3 Funciones de transferencia

Un modelo más académico que facilita el estudio de una neurona, puede visualizarse en la figura siguiente (2.4).

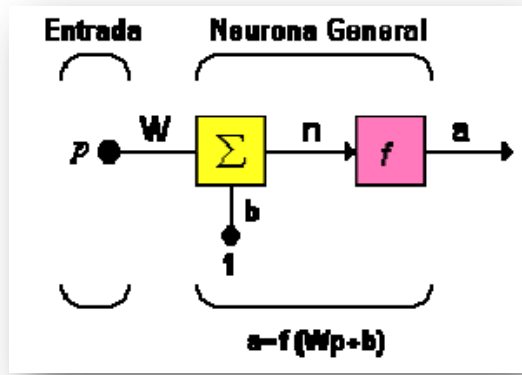


Figura 2.4 Modelo de una neurona artificial.

Las entradas a la red serán ahora representadas en el vector p , que para el caso de una sola neurona contiene un solo elemento, w representa los valores de los pesos y la entrada b es una ganancia que refuerza la salida del sumador n , la cual es la salida neta de la red; la

salida total está determinada por la función de transferencia f , la cual puede ser una función lineal o no de n , y que es escogida dependiendo de las especificaciones del problema que la neurona tenga que resolver; aunque las RNA se inspiren en modelos biológicos, no existe ninguna limitación para realizar modificaciones en las funciones de salida, así que se encontrarán modelos artificiales que nada tienen que ver con las características del sistema biológico [18].

2.3.1 Limitador fuerte (hardlim)

La siguiente figura (2.5), muestra cómo esta función de transferencia acerca la salida de la red a cero, si el argumento de la función es menor que cero y la lleva a uno si este argumento es mayor que uno. Esta función crea neuronas que clasifican las entradas en dos categorías diferentes, características que le permiten ser empleada en la red tipo perceptrón [19].

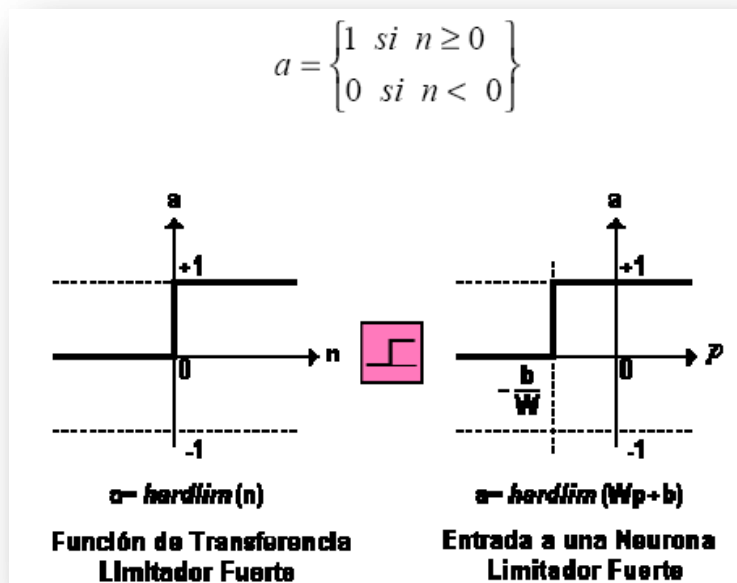


Figura 2.5 Representación gráfica de la función de transferencia con limitador fuerte.

De esta forma el icono para la función hardlim reemplazará a la letra f en la expresión general, cuando se utilice esta función.

Una modificación a esta función puede verse en la figura 2.6, la que representa la función de transferencia hardlims que restringe el espacio de salida a valores entre 1 y -1 [19].

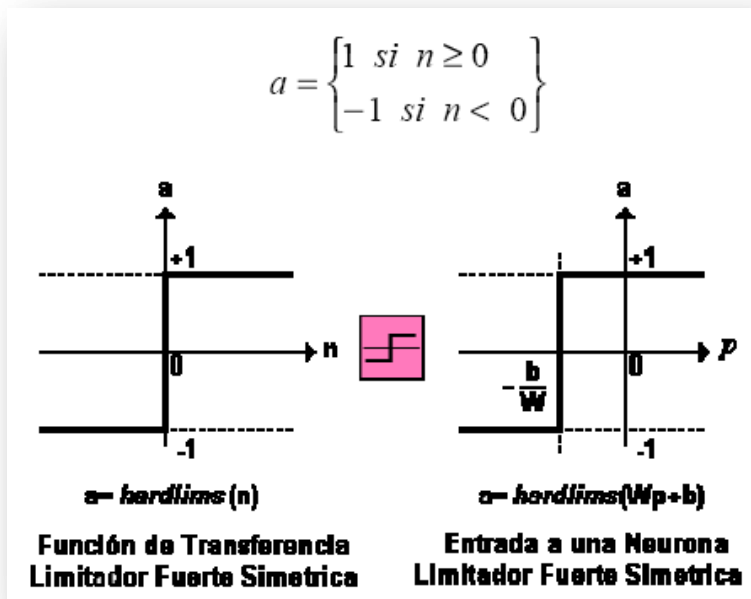


Figura 2.6 Representación gráfica de la función de transferencia Hardlims.

2.3.2 Función de transferencia lineal (purelin)

La salida de una función de transferencia lineal es igual a su entrada.

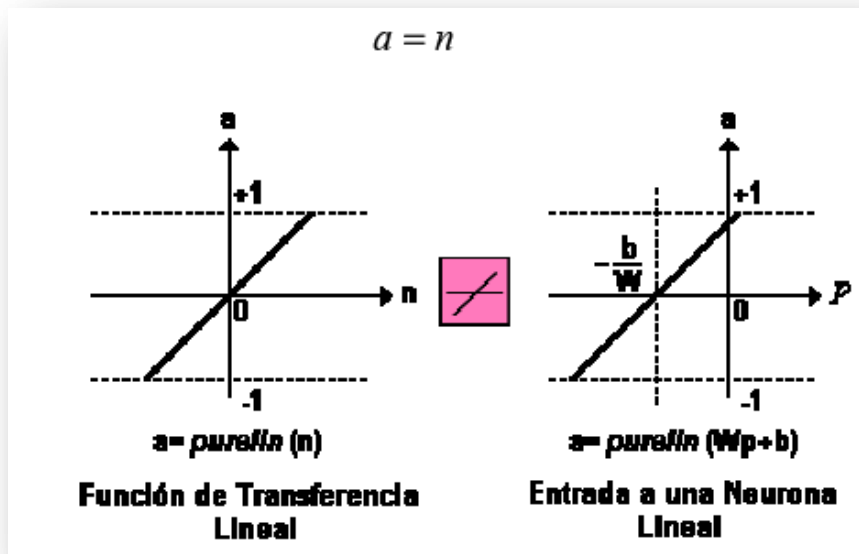


Figura 2.7 Salida de una función de transferencia lineal.

En la gráfica del lado derecho de la figura 2.7, puede verse la relación entre la salida a de la red, y la entrada p , teniendo en cuenta el valor de la ganancia b ; las neuronas que emplean este tipo de función de transferencia son utilizadas en la red tipo adaptativa de elemento lineal (ADALINE) [19].

2.3.3 Función de transferencia tipo logarítmica sigmoidea (logsig)

Esta función toma los valores de entrada, los cuales pueden oscilar entre mas y menos infinito, y restringe la salida a valores entre cero y uno de acuerdo a la expresión (ecuación 8):

$$a = \frac{1}{1+e^{-n}} \dots \dots \dots (8)$$

Ésta función es comúnmente usada en redes multicapa como retropropagación (backpropagation), porque la función logsig es diferenciable [19].

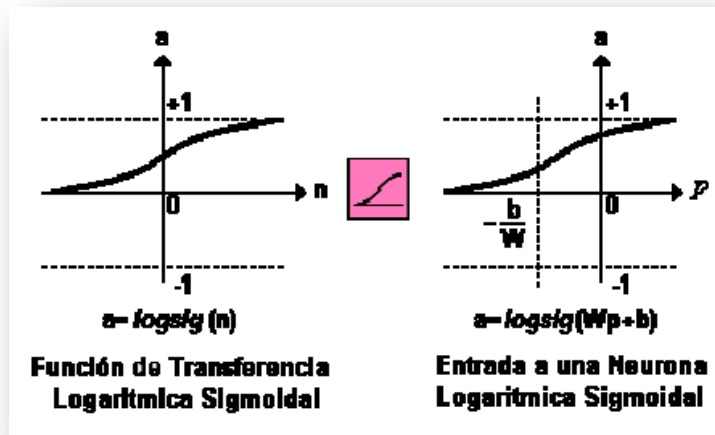


Figura 2.8 Salida de una función de transferencia logarítmica sigmoidea.

2.3.4 Principales funciones de transferencia

La siguiente figura (2.9) es una relación de las principales funciones de transferencia empleadas en redes neuronales. Cabe aclarar que para el uso con redes de retropropagación las principales funciones utilizadas son la sigmoidea y la tangente sigmoidea, porque para redes multicapa se necesitan funciones de transferencia diferenciables [18].

Nombre	Relación Entrada /Salida	Icono	Función
Limitador Fuerte	$a = 0 \quad n < 0$ $a = 1 \quad n \geq 0$		hardlim
Limitador Fuerte Simétrico	$a = -1 \quad n < 0$ $a = +1 \quad n \geq 0$		hardlims
Lineal Positiva	$a = 0 \quad n < 0$ $a = n \quad 0 \leq n$		poslin
Lineal	$a = n$		purelin
Lineal Saturado	$a = 0 \quad n < 0$ $a = n \quad 0 \leq n \leq 1$ $a = 1 \quad n > 1$		satlin
Lineal Saturado Simétrico	$a = -1 \quad n < -1$ $a = n \quad -1 \leq n \leq 1$ $a = +1 \quad n > 1$		satlins
Sigmoidal Logarítmico	$a = \frac{1}{1 + e^{-n}}$		logsig
Tangente Sigmoidal Hiperbólica	$a = \frac{e^n - e^{-n}}{e^n + e^{-n}}$		tansig
Competitiva	$a = 1$ Neuronas con n max $a = 0$ El resto de neuronas		compet

Figura 2.9 Funciones de transferencia utilizadas en redes neuronales.

2.4 Topología de una red

Típicamente una neurona tiene más de una entrada; en la siguiente figura se observa una neurona con R entradas; las entradas individuales p_1, p_2, \dots, p_R son multiplicadas por los pesos correspondientes $w_{1,1}, w_{1,2}, \dots, w_{1,R}$ pertenecientes a la matriz de pesos W [18].

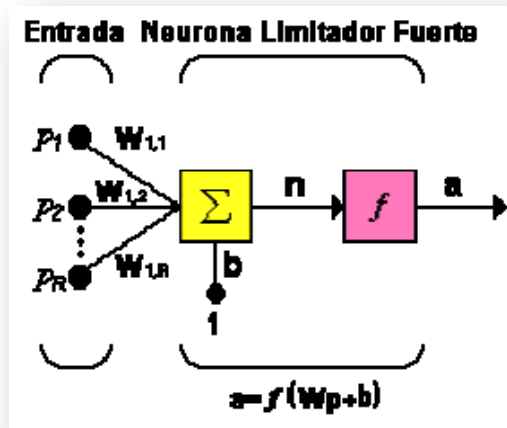


Figura 2.10 Topología de una red neuronal.

La entrada tiene una ganancia b , la cual llega al mismo sumador al que llegan las entradas multiplicadas por los pesos, para formar la salida n .

$$n = w_{1,1}p_1 + w_{1,2}p_2 + \dots + w_{1,R}p_R + b \dots \dots \dots (9)$$

Esta expresión puede ser escrita en forma matricial:

$$n = Wp + b \dots \dots \dots (10)$$

Esta convención se hace más útil cuando hay más de una neurona, o cuando se tiene una red con demasiados parámetros; en este caso la notación de la figura anterior puede resultar inapropiada y se prefiere emplear la notación compacta de la siguiente figura [18].

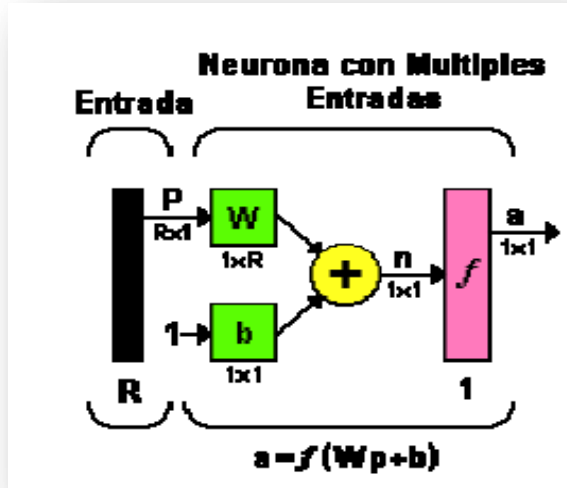


Figura 2.11 Notación compacta de una red neuronal.

Dentro de una red neuronal, los elementos de procesamiento se encuentran agrupados por capas, una capa es una colección de neuronas; de acuerdo a la ubicación de la capa de RNA, ésta recibe diferentes nombres. Capa de entrada: recibe las señales de la entrada de la red, algunos autores no consideran el vector de entrada como una capa pues allí no se lleva a cabo ningún proceso, en este trabajo se seguirá este criterio. Capas ocultas: estas capas son aquellas que no tienen contacto con el medio exterior, sus elementos pueden tener diferentes conexiones y son éstas las que determinan las diferentes topologías de la red. Capa de salida: recibe la información de la capa oculta y transmite la respuesta al medio externo.

Una red de una sola capa con un número S de neuronas, se observa en la figura 2.12, en la cual, cada una de las R entradas es conectada a cada una de las neuronas; la matriz de pesos tiene ahora S filas. La capa incluye la matriz de pesos, los sumadores, el vector de ganancias, la función de transferencia y el vector de salida. Esta misma capa se observa en notación abreviada en la figura (2.13).

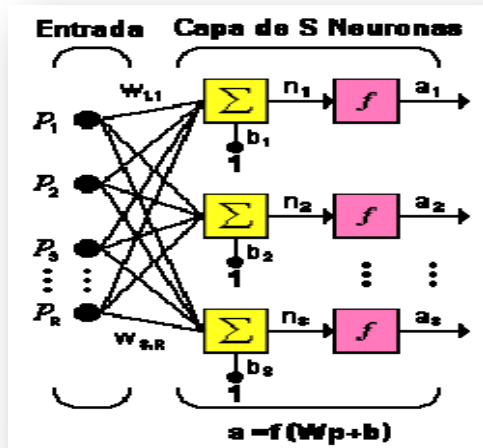


Figura 2.12 Red neuronal de una capa oculta.

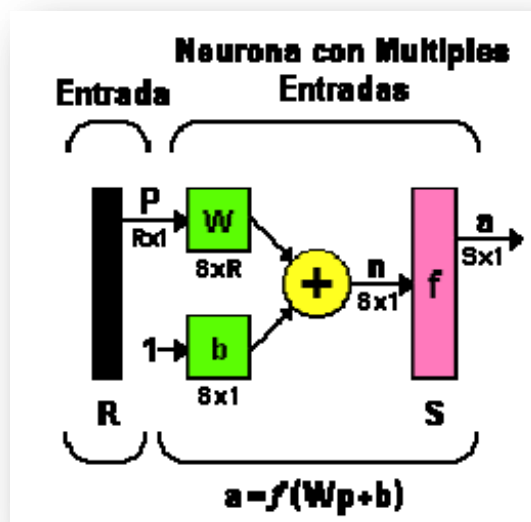


Figura 2.13 Notación compacta de una RNA de una capa oculta.

En la figura 2.14, si se considera una red con varias capas (o red multicapa) cada capa tendrá su propia matriz de pesos W , su propio vector de ganancias b , un vector de entradas n , y un vector de salida a . La versión completa y la versión en notación abreviada de una red de tres capas, pueden ser visualizadas en las siguientes figuras (2.14 y 2.15). Para esta RNA se tienen R entradas, S^1 neuronas en la primera capa, S^2 neuronas en la segunda capa, las cuales pueden ser diferentes; las salidas de las capas 1 y 2 son las entradas a las capas 2 y 3 respectivamente. La capa 2 puede ser vista como una red de una capa con $R=S^1$ entradas, $S^1=S^2$ neuronas y una matriz de pesos W^2 de dimensiones $S^1 \times S^2$ [18].

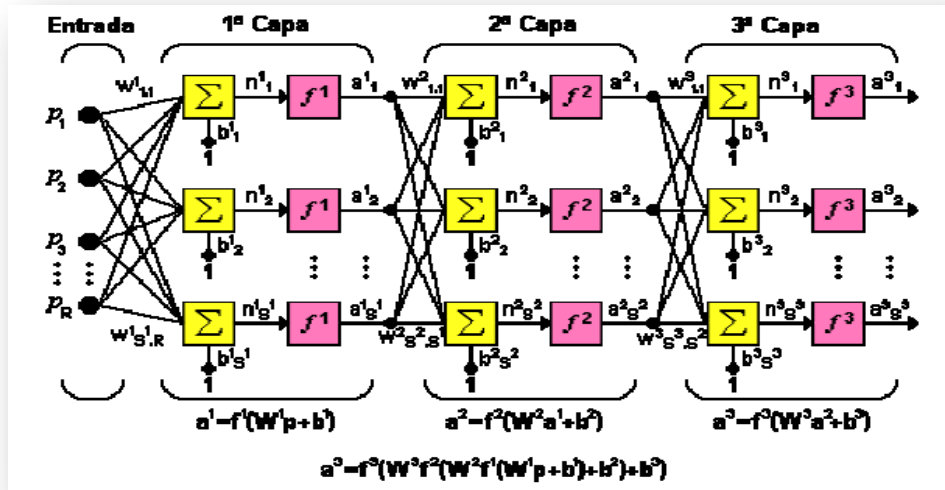


Figura 2.14 Red neuronal de tres capas.

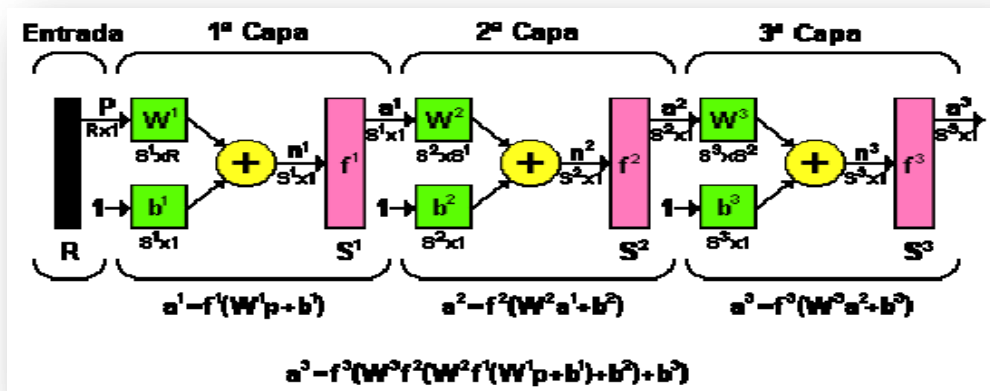


Figura 2.15 Representación abreviada de una RNA de tres capas.

Las redes multicapa son más poderosas que las redes de una sola capa, por ejemplo, una red de dos capas que tenga una función sigmoidea en la primera capa y una función lineal en la segunda, puede ser entrenada para aproximar muchas funciones de forma aceptable, una red de una sola capa no podía hacer esto.

2.5 Perceptrón

La red tipo perceptrón fue inventada por el psicólogo Frank Rosenblatt en el año 1957. Su intención era ilustrar algunas propiedades fundamentales de los sistemas inteligentes en general, sin entrar en mayores detalles con respecto a condiciones específicas y desconocidas para organismos biológicos concretos [20]. Todo el esquema de conexiones se describe en forma general en un diagrama de Venn, para un perceptrón sencillo con dos unidades de respuesta como se muestra en la figura 2.19 [18].

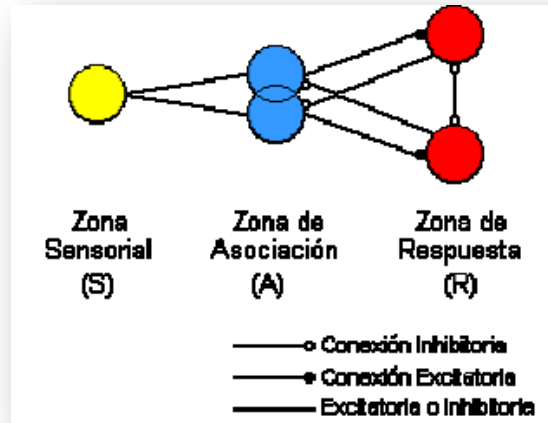


Figura 2.16 Diagrama de Venn, para un perceptrón simple.

El perceptrón era inicialmente un dispositivo de aprendizaje, en su configuración inicial no estaba en capacidad de distinguir patrones de entrada muy complejos, sin embargo, mediante un proceso de entrenamiento era capaz de adquirir esta capacidad. En esencia, el entrenamiento implica un proceso de refuerzo mediante el cual la salida de las unidades se incrementa o disminuye dependiendo de si las unidades A contribuían o no a las respuestas correctas del Perceptrón de una entrada dada.

La red tipo perceptrón emplea principalmente dos funciones de transferencia, *hardlim* con salidas 1,0 o *hardlims* con salidas -1, 1; su uso depende del valor de salida que se espera para la red, es decir, si la salida de la red es unipolar o bipolar.

Una técnica utilizada para analizar el comportamiento de redes como el perceptrón es presentar en un mapa las regiones de decisión creadas en el espacio multidimensional de entradas de la red, en estas regiones se visualiza qué patrones pertenecen a una clase y cuáles a otra. El perceptrón separa las regiones por un hiperplano cuya ecuación queda determinada por los pesos de las conexiones y el valor umbral de la función de activación de la neurona; en este caso los valores de los pesos pueden fijarse o adaptarse empleando diferentes algoritmos de entrenamiento [19].

2.6 Retropropagación (backpropagation)

La regla de aprendizaje del perceptrón de Rosenblatt y el algoritmo LMS de Widrow y Hoff fueron diseñados para entrenar redes de una sola capa. Estas redes tienen la desventaja de que solo pueden resolver problemas linealmente separables, fue esto lo que llevó al surgimiento de las redes multicapa, para superar esta dificultad en las redes hasta entonces conocidas.

El primer algoritmo de entrenamiento para redes multicapa fue desarrollado por Paul Werbos en 1974, éste se desarrolló en un contexto general, para cualquier tipo de redes neuronales sin ninguna aplicación especial, razón por la cual el algoritmo no fue aceptado dentro de la comunidad de desarrolladores de redes neuronales. Fue solo hasta mediados de

los años 80 cuando el algoritmo de propagación inversa fue redescubierto al mismo tiempo por varios investigadores: David Rumelhart, Geoffrey Hinton, Ronal Williams, David Parker y Yann Le Cun. El algoritmo se popularizó cuando fue incluido en el libro “Parallel Distributed Processing Group” por los psicólogos David Rumelhart y James McClelland. La publicación de éste trajo consigo un auge en las investigaciones con redes neuronales, siendo la retropropagación una de las redes más ampliamente empleadas, aún en nuestros días [21].

La retropropagación es un tipo de RNA de aprendizaje supervisado, que emplea un ciclo de propagación-adaptación en dos fases. Una vez que se ha aplicado un patrón a la entrada de la red como estímulo, éste se propaga desde la primera capa a través de las capas superiores de la red, hasta generar una salida. La señal de salida se compara con la salida deseada y se calcula una señal de error para cada una de las salidas.

Las salidas de error se propagan hacia atrás, partiendo de la capa de salida, hacia todas las neuronas de la capa oculta que contribuyen directamente a la salida. Sin embargo, las neuronas de la capa oculta sólo reciben una fracción de la señal total del error, basándose aproximadamente en la contribución relativa que haya aportado cada neurona a la salida original. Este proceso se repite, capa por capa, hasta que todas las neuronas de la red hayan recibido una señal de error que describa su contribución relativa al error total. Basándose en la señal de error percibida, se actualizan los pesos de conexión de cada neurona, para hacer que la red converja hacia un estado que permita clasificar correctamente todos los patrones de entrenamiento [21].

La estructura típica de una red multicapa se observa en la figura 2.17.

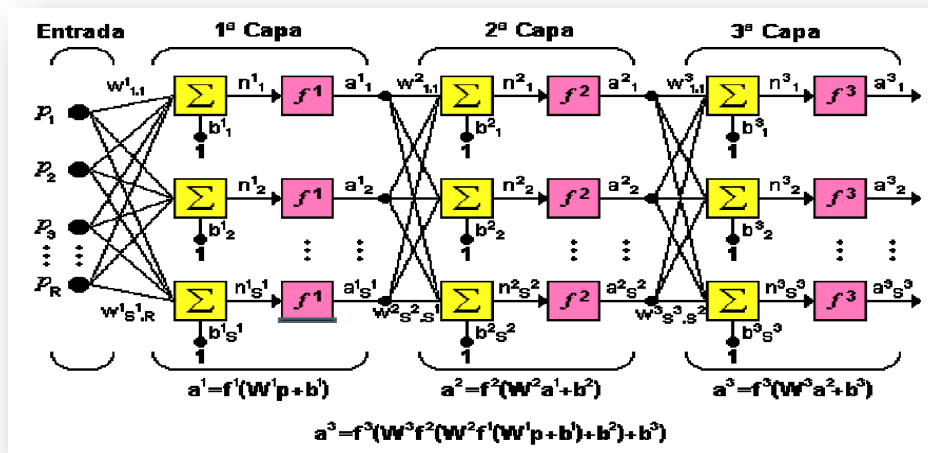


Figura 2.17 Estructura general de una red neuronal multicapa.

Puede notarse que esta red de tres capas equivale a tener tres redes tipo perceptrón en cascada; la salida de la primera red es la entrada a la segunda y la salida de la segunda es la entrada de la tercera. Cada capa puede tener diferente número de neuronas, e incluso distinta función de transferencia [18].

En la figura anterior W^l representa la matriz de pesos para la primera capa, W^2 los pesos de la segunda capa y así sucesivamente para todas las capas que incluyan la red.

El procedimiento de retropropagación es una forma relativamente eficiente de calcular qué tanto se mejora el desempeño con los cambios individuales en los pesos. Se conoce como procedimiento de retropropagación porque primero calcula cambios en la capa final, reutiliza gran parte de los mismos cálculos para calcular los cambios de los pesos de la penúltima capa y, finalmente, regresa a la capa inicial.

Al comparar la señal de salida con una respuesta deseada o salida objetivo, $d(t)$, se produce una señal de error, $e(t)$. Esta señal de error en la neurona de salida j en la iteración t produce $e(t)=d(t) - y(t)$ donde t denota el tiempo discreto. El error total estará dado por:

$$E(t) = \frac{1}{2} \sum_{j=1}^m e_j^2(t) \dots \dots \dots (11)$$

Entonces el error cuadrático medio será:

$$E_{prom} = \frac{1}{q} \sum_{j=1}^q E(t) \dots \dots \dots (12)$$

La corrección de los pesos $\Delta W_{ji}(t)$ aplicada a $W_{ji}(t)$ se define por la regla delta:

$$\begin{pmatrix} \text{Corrección} \\ \text{de peso} \\ \Delta w_{ij}(t) \end{pmatrix} = \begin{pmatrix} \text{parámetro} \\ \text{tasa de aprendizaje} \\ \eta \end{pmatrix} \cdot \begin{pmatrix} \text{gradiente} \\ \text{local} \\ \delta_j(t) \end{pmatrix} \cdot \begin{pmatrix} \text{señal de entrada} \\ \text{de la neurona } j \\ y_i(t) \end{pmatrix}$$

Regla delta generalizada

$$\Delta w_{ji}(t) = \alpha \Delta w_{ji}(t - 1) + \eta \delta_j(t) y_j(t) \dots \dots \dots (13)$$

α : Constante de momento;

η : Tasa de aprendizaje.

Se hacen iterar los cálculos hacia delante y hacia atrás, presentando nuevas épocas de ejemplos de entrenamiento hasta que el criterio de parada se cumpla. El orden de presentación de los ejemplos de entrenamiento se deben hacer al azar entre estados (épocas).

Existen dos modos de entrenamiento: secuencial (en línea), donde la actualización de los pesos es después de presentar cada ejemplo de entrenamiento; y por lotes (Batch) en donde la actualización de los pesos se hace después de presentar todos los ejemplos de entrenamiento.

2.7 Redes neuronales de base radial (RBR)

En funciones de base radial la salida depende de la distancia a un punto denominado centro. Éstas tienen un carácter de local, pues son funciones que alcanzan un nivel cercano máximo

de su recorrido cuando el patrón de entrada X_n está próximo al centro de la neurona. A medida que el patrón se aleja del centro, el valor de la función va reduciendo [19].

Las salidas son por tanto una combinación lineal de funciones Gaussianas, cada una de las cuales se activa para una determinada porción del espacio definido por los patrones de entrada.

Las funciones RBR tienen como características: ser simétricas respecto a un punto, se definen con al menos dos parámetros (centro y anchura o magnitud de la variación de la función según se aleja del centro), las capas de entrada reciben las señales directamente del exterior, no realizan ningún pre-procesado, la capa oculta recibe las señales de la capa de entrada y realiza una transformación local y no lineal sobre dichas señales, en la capa de salida se realiza una combinación lineal de las actividades de las neuronas de la capa oculta y actúa como salida de la red [19].

El entrenamiento de este tipo de redes, determina todos los parámetros de la red. Parámetros de la capa de salida, pesos, parámetros de la capa oculta, centros, y desviaciones asociadas [22]. La determinación de los parámetros de la capa oculta se realiza mediante la optimización en el espacio de las entradas, ya que cada neurona va a representar una zona diferente en dicho espacio. La determinación de los parámetros de la capa de salida y la optimización se realiza en base a las salidas que se desea obtener ya que en su globalidad, las redes de base radial se utilizan para aproximar relaciones entre el conjunto de variables de entrada y salida que definen el problema [22].

Cada elemento de proceso calcula su valor neto como una combinación lineal de las salidas de los elementos de proceso de la capa oculta. La función de activación y transferencia son lineales por lo tanto, para un patrón n , será $X(n)=(x1(n), x2(n), \dots, xp(n))$. La salida de la red asociada a cada elemento k de la capa de salida y se obtiene de la siguiente manera:

$$y_k(n) = \sum_{i=1}^m w_{ik} z_i(n) + \mu_k \quad \text{para } k = 1, 2, \dots, r \dots \dots \dots (14)$$

Donde los w_{ik} son los pesos asociados al elemento k de la capa de salida y el elemento i de la capa oculta, que ponderan cada uno las salidas $z_i(n)$ del elemento de procesado de la capa oculta correspondiente [19].

El término μ_k es un término denominado umbral y está asociado a cada uno de los elementos de proceso de la capa de salida.

Cada elemento i de proceso de la capa oculta tiene asociada una función de base radial de tal manera que representa una clase o categoría, donde dicha clase viene dada por (C_i, d_i) . C_i representa un centro del cluster (pesos asociados a cada neurona i) y d_i representa la decisión, anchura o dilatación de la función de base radial asociada a dicho elemento.

La salida de cada elemento de la capa oculta $z_i(n)$ se calcula como la distancia que existe entre el patrón de entrada X_n y el centro del cluster C_i , ponderada inversamente por d_i y aplicando después a ese valor una función de base radial, tenemos:

$$z_t(n) = \Phi \left[\frac{\left(\sum_{j=1}^p (x_j(n) - c_{ji})^2 \right)^{1/2}}{d_i} \right] \quad \text{para } i = 1, 2, \dots, n \dots \dots \dots (15)$$

Donde Φ es una función de base radial; dentro de estas, la más utilizada es la función Gaussiana $\Phi(r) = e^{\left(\frac{-r^2}{2}\right)}$. La estructura general de una red neuronal de base radial puede observarse en la figura 2.18.

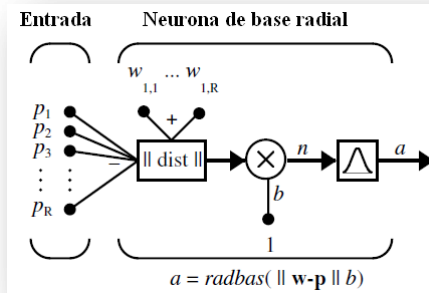


Figura 2.18 Arquitectura de una red de base radial.

Las redes neuronales de base radial pueden requerir más neuronas que las redes de retropropagación estándar, pero a menudo pueden diseñarse en una fracción del tiempo que se tarda en entrenar redes de retropropagación estándar. Funcionan mejor cuando hay muchos vectores de formación disponibles. En este trabajo se utilizaron dos variantes de redes de base radial de regresión generalizada (GRNN) y redes neuronales probabilísticas (PNN). Entre los métodos de diseño para GRNN están newrbe y newrb, para los métodos PNN están newgrnn y newpnn [19], ambas variantes serán descritas en los próximos subcapítulos.

2.7.1 Diseño más eficiente

Esta función crea iterativamente una red de base radial con una neurona a la vez. Las neuronas son sumadas hasta que la suma de errores cuadrados alcanza el error mínimo o un máximo de neuronas ha sido alcanzado. El método de diseño es similar a las redes de diseño exacto. La diferencia es que el diseño eficiente crea una neurona a la vez. En cada iteración el vector de entrada que resulta trata de disminuir el error de la red. El error es revisado, si es menor que el objetivo, el entrenamiento es terminado; de lo contrario se agrega una nueva neurona. Este proceso se repite hasta que el error máximo permitido es alcanzado o el número de neuronas alcanzado. Es importante que el parámetro de dispersión sea lo suficientemente grande para que las neuronas radbas respondan a la superposición de regiones en el espacio de entrada, pero no tan grande porque las neuronas pueden responder esencialmente de la misma manera [19].

2.7.2 Redes neuronales probabilísticas

Cuando se presenta una entrada, la primera capa calcula la distancia entre el vector de entrenamiento y el vector de entrada, después produce un vector cuyos elementos indican cuán cerca están los vectores de entrada a los vectores de entrenamiento. La segunda capa suma estas contribuciones para cada clase de entrada, con el objetivo de producir un vector de probabilidades de salida de su red. Finalmente, sobre el resultado de la segunda capa una función de transferencia completa toma el máximo de estas probabilidades y produce un 1 de esa clase y un 0 para las otras clases. La arquitectura para este sistema se muestra abajo (figura 2.23).

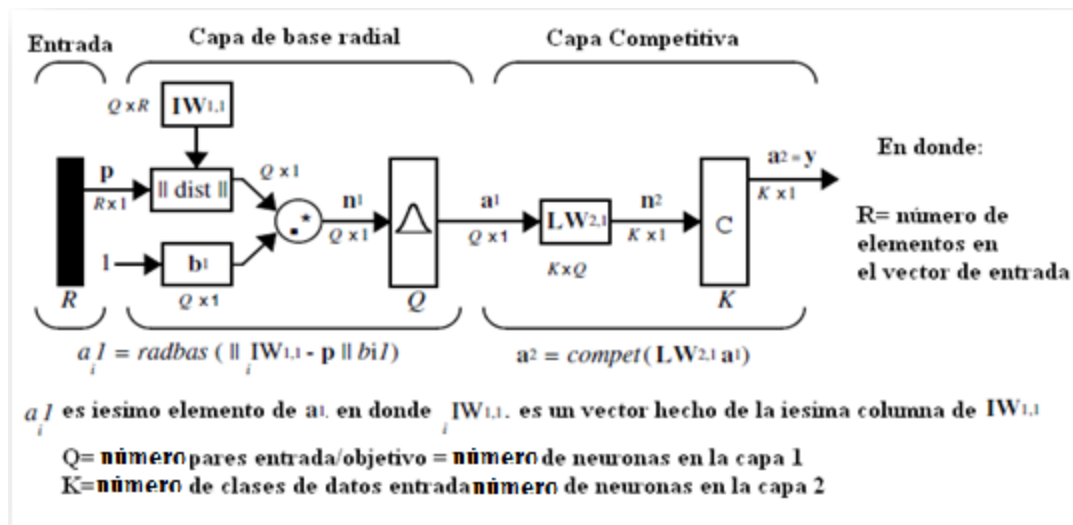


Figura 2.19 Estructura de una red probabilística.

Hay Q pares de vector de entrada vector/objetivo. Cada vector objetivo tiene K elementos. Uno de estos elementos es 1 y el resto son 0. Por lo tanto, cada vector de entrada está asociado con una de las K clases.

Los pesos de entrada de la primera capa, $IW^{1,1}$ ($net.IW\{1,1\}$), se establecen en la transpuesta P' de la matriz que se formó a partir de los pares de formación de Q . Cuando se presenta una entrada, la indicación $|| dist ||$ produce un vector cuyos elementos indican cuán cerca de la entrada esta de los vectores del conjunto de la formación. Estos elementos son multiplicados, elemento por elemento, por la parcialidad y se enviarán a la función de transferencia de radbas. Un vector de entrada cercano a un vector de formación está representado por un número cercano a 1 en el a^1 vector de salida. Si una entrada está cerca de varios vectores de entrenamiento de una sola clase, es representada por varios elementos de a^1 que están cerca de 1.

Los pesos de la segunda capa, $LW^{1,2}$ ($net.LW\{2,1\}$), se establecen en la matriz T de vectores objetivo. Cada vector tiene un 1 solo en la fila asociada con esa clase particular de entrada y 0 en otros lugares. Se utilizó la función $ind2vec$ para crear los vectores adecuados. La multiplicación de T por a^1 , suma los elementos de a^1 debido a cada una de las clases de entrada de K . Por último, la función de transferencia de segundo nivel, produce un 1

correspondiente al elemento más grande de n_2 y del 0 que en otros lugares. Por lo tanto, la red clasifica el vector de entrada en una clase específica de K debido a que la clase tiene la máxima probabilidad de ser correcta.

2.7.3 Regresión generalizada

A continuación se muestra la arquitectura para la GRNN, figura 2.20. Es similar a la red de base radial, pero la segunda capa es diferente. Aquí el cuadro de $nprod$ que se muestra (código de función *normprod*) produce elementos de S_2 en vector n_2 . Cada elemento es el producto escalar de una fila de LW_2 , el 1 y el a_1 de vector de entrada, todo es normalizado por la suma de los elementos de a_1 [19].

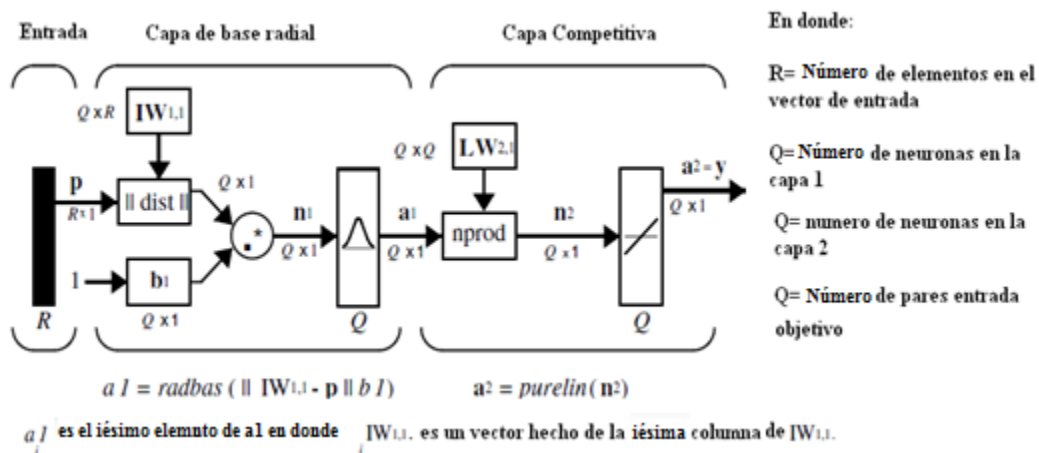


Figura 2.20 Arquitectura de las redes de regresión generalizada

Las redes GRNN tienden a tener en muchas ocasiones más neuronas que las comparables con retropropagación con funciones de transferencia tangente sigmoidea y logarítmica sigmoidea en las capas ocultas. Esto es debido a que en las redes GRNN las neuronas pueden tener salidas en una región amplia del espacio de entrada, mientras que las redes radiales solo responden a relativamente pequeñas regiones del espacio de entrada. El resultado es que entre más grande sea el espacio de entrada, más neuronas se requieren en la base radial. Por otro lado, el diseño de este tipo de redes usualmente toma mucho menos tiempo que el entrenamiento de una red lineal sigmoidea.

Conclusiones

Las redes neuronales artificiales fueron basadas en el comportamiento de una neurona biológica, la cual fue presentada para entender el funcionamiento y después se presentó la analogía con la red artificial. Una vez que ésta es comprendida en los niveles superficiales es posible entender más a fondo el funcionamiento de cada función de entrenamiento; así se presentó cada una de las redes utilizadas y la arquitectura de las mismas. Se describió la estructura de las redes de base radial y retropropagación. Como se mencionó estas redes se aplican en el reconocimiento de patrones, en el caso de estudio cada patrón representa un comportamiento del motor, los cuales en conjunto reflejan el estado del motor; así por medio de las redes neuronales es posible identificar las fallas de turbina de gas.

CAPÍTULO III
DESCRIPCIÓN DE LOS
ALGORITMOS

Introducción

Como ya se mencionó, las redes neuronales son capaces de identificar patrones de comportamiento pero para ello se debe utilizar un algoritmo universal de clasificación para la diagnosis, que posteriormente pueda ser utilizado por las redes para dar un diagnóstico. En este apartado se explican los criterios utilizados para la utilización de dichos algoritmos.

Las clases son simuladas utilizando modelos estáticos no lineales. Cada clase es representada por un conjunto de mediciones muestra (patrones) que, por ser mediciones reales hacen factible un error. Estas muestras son introducidas a redes neuronales usadas posteriormente para la diagnosis. Las redes neuronales entrenadas son utilizadas posteriormente para calcular probabilidades de diagnóstico por medio de una prueba de modelo estático.

3.1 Enfoque de diagnóstico

Existen dos tipos de modelos de diagnóstico del conducto de flujo. No lineal y lineal. Un modelo termodinámico, multi-régimen, no lineal, puede ser representado por la siguiente expresión:

$$\vec{Y} = F(\vec{\Theta}, \vec{U}) \dots\dots\dots(16)$$

en donde los parámetros del conducto de flujo \vec{Y} dependen del vector de régimen de operación del motor, las condiciones atmosféricas \vec{U} y el vector de los parámetros de estado $\vec{\Theta}$, que son usados para describir fallas del motor. Entre los parámetros de estado del motor se eligen parámetros de desempeño de los componentes del mismo, los cuales son capaces de describir desviaciones en desempeño de los componentes y de simular fallas.

El modelo lineal

$$\delta \vec{Y} = H \delta \vec{\Theta} \dots\dots\dots(17)$$

conecta las desviaciones relativas de los parámetros de estado $\delta \vec{\Theta}$ con las desviaciones relativas $\delta \vec{Y}$ de los parámetros del conducto de flujo por la matriz de coeficientes de influencia H en un régimen de operación constante. Los valores \vec{Y} generados por los modelos descritos, difieren de las mediciones \vec{Y}^* debido a los errores de medición $\vec{\epsilon}$:

$$\vec{Y}^* = \vec{Y} + \vec{\epsilon} \dots\dots\dots(18)$$

En la diagnosis técnica se supone generalmente que el objeto en análisis únicamente puede pertenecer a una clase q de las clases D_1, D_2, \dots, D_q . Cada clase D_j se describe en la región Ω_j^* del espacio de diagnóstico elegido \vec{Z} por la medición la función de densidad de distribución $f(\vec{Z}^*/D_j)$. El parámetro \vec{Z} representa los parámetros de conducto de flujo convertidos acorde a la expresión:

$$Z_i = (Y_i - Y_{0i}) / \sigma Y_i^* , \dots\dots\dots (19)$$

en donde, σY_i^* es el factor de dispersión de errores aleatorios de medición. El modelo lineal es aplicado para determinar los cambios en los parámetros $(Y_i - Y_{0i})$ inducidos por las fallas. Las hipótesis también fueron tomadas de una distribución uniforme por medio de la función $f(\vec{Z}/D_j)$ y la distribución normal para la función $f(\vec{Z}^*/\vec{Z})$. Todas estas hipótesis son simplificadas y calculadas por medio de la función resultante:

$$f(\vec{Z}^*/D_j) = \int_{\Omega_j} f(\vec{Z}^*/\vec{Z}) f(\vec{Z}/D_j) d\Omega_j \dots\dots\dots (20)$$

Y la representación de las clases seria como en la figura 3.1.

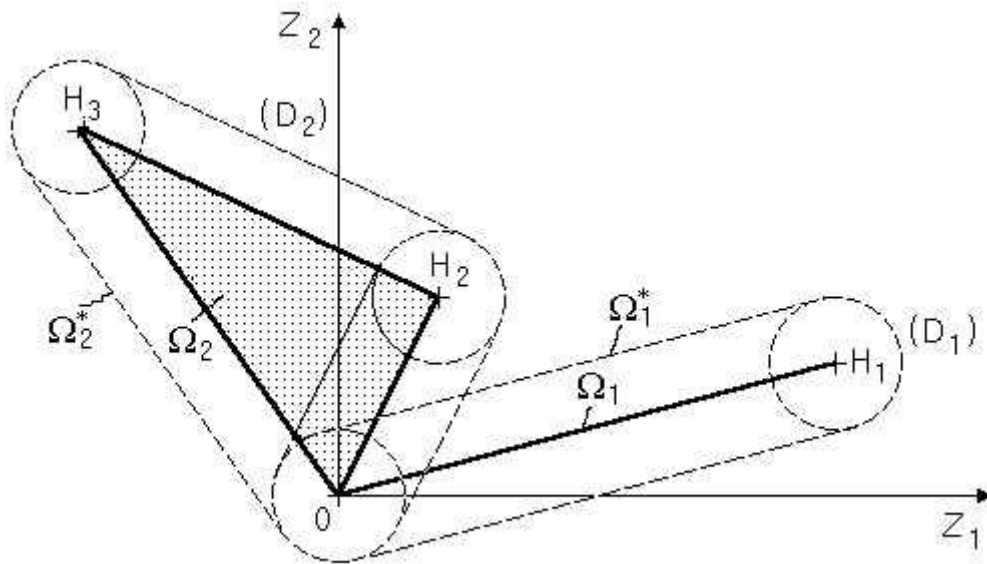


Figura 3.1 Representación de las clases del conducto de flujo.

El acierto en la diagnosis está determinado en base a la clasificación descrita arriba y a la probabilidad posterior $P(d_j/\vec{Z}^*)$ acorde a la formula de Bayes, mencionada en el capítulo 1.

3.2 Estructura del algoritmo

En el presente capítulo se muestra el desarrollo del algoritmo utilizado, el cual incluye una interface usuario que permite hacer cálculos más eficientemente. Esta interface utiliza dos tipos de redes neuronales para el diagnóstico. Estos dos tipos de redes tienen ciertas variantes. Las de redes de tipo retropropagación incorporan 12 funciones de entrenamiento, mientras que por las redes del tipo radial se presentan tres redes particulares.

La interface usuario es inicializada por medio de MatLab. Una vez que la interface usuario es inicializada, permite al operador introducir las constantes y definir el tipo de entrenamiento, al final arroja valores de probabilidad y el tiempo de cálculo.

El programa está desarrollado para valorar por medio de las 15 variables de redes mencionadas, la probabilidad de la diagnosis correcta de falla del motor. El cálculo considera 9 tipos de defectos, utilizando 6 parámetros medidos, 11 regímenes de operación del motor y 21 puntos de interpolación por desarrollo de cada defecto (clase). En general el algoritmo está estructurado acorde a la figura 3.2.

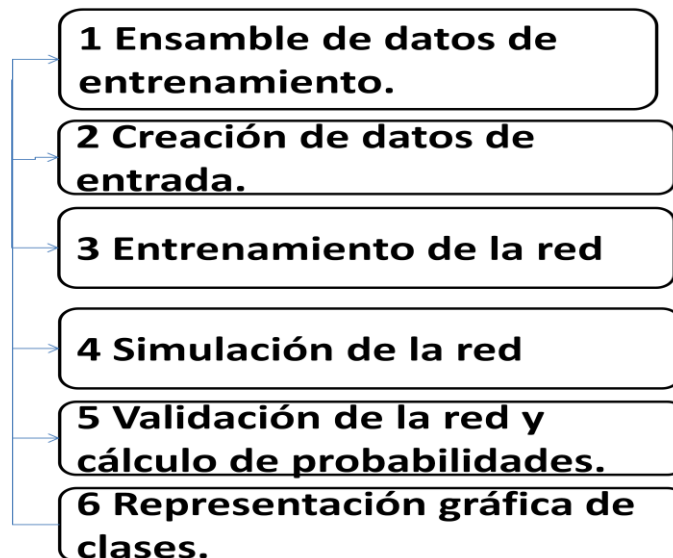


Figura 3.2 Estructura del algoritmo.

3.3 Ensamblajes de datos de entrenamiento

La primera fase es el ensamble de los datos de entrenamiento y análisis. La base de datos está compuesta por datos obtenidos de lecturas de una turbina real (turbina de dos ejes utilizada para el bombeo de gas natural), estos datos están contenidos en una matriz de [9000 x 6]; los datos deben ser ensamblados de tal manera que el compilador, al seguir las instrucciones, sepa a qué datos específicos nos referimos para el análisis. Dentro de esta etapa del algoritmo se especifican ciertas sub-etapas, las cuales serán descritas en los posteriores subcapítulos. En la figura 3.3 se muestra el procedimiento seguido en la primera etapa del algoritmo.

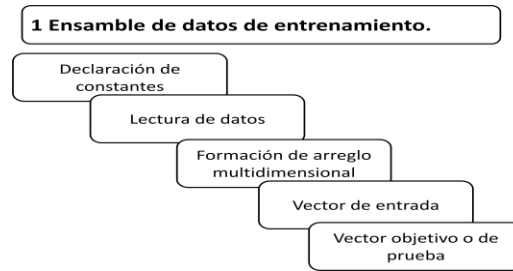


Figura 3.3 Etapas en el ensamble de datos de entrenamiento.

3.3.1 Declaración de contantes

En la primera parte del algoritmo se especifican las constantes a utilizar; para ello se consideran 11 regímenes de operación (krg), hay que especificar cuál de ellos se utilizará para el cálculo (nreg), nueve defectos o clases (kdf), 21 puntos considerados para el desarrollo de la falla (kdl), la cual considera 0-5 % del desarrollo del defecto, el número de puntos por clase a simular (ktd), los números llave (o semilla) a utilizar para la generación de la nube de puntos (nr1 y nr2) y amplitud de los errores aleatorios (DY).

Los parámetros medidos son: presión de compresor, presión de turbina, temperatura de compresor, temperatura de turbina, temperatura de turbina de potencia, caudal de combustible. Los parámetros que ayudan a describir los defectos son: flujo de compresor, eficiencia de compresor, flujo de turbina de alta presión, eficiencia de turbina de alta presión, flujo de turbina de potencia, eficiencia de turbina de potencia, parámetro de recuperación de presión total de la cámara de combustión, eficiencia de combustión y parámetro de recuperación de presión total.

```

krg=11;
nreg=1;
kdf=9;
kdl=21;
kdf0=9;
kdf1=9;
idef2=0;
kdf2=4;
kpr0=6;
ktd=1000;
nr1=0; nr2=1;
DY=[0.015 0.015 0.025 0.015 0.02 0.02];
krdd=krg*kdf0*kdl ;
  
```

Figura 3.4 Declaración de constantes.

La declaración de las constantes es llamada por la interface usuario, en ella se consideran solo los parámetros a cambiar por el usuario, hay algunos parámetros como en número de clases y el número de datos medidos que no cambian en cada experimento, por lo tanto, solo fueron declarados en el código interno de la interface y no es necesario cambiarlos.

Los datos como régimen de operación, número de puntos a simular y números llave, son parámetros definidos por el usuario.

3.3.2 Lectura de datos

Los datos son lecturas de sensores en una turbina real, estos datos están contenidos en una matriz de [9000 x 6], lo primero que hace el compilador es leer estos datos, los cuales son valores absolutos, se puede asumir que los errores de medición sistemáticos han sido eliminados y que el valor de las desviaciones (δY^*) solo incluye los errores aleatorios dados que presentan una amplitud dada por el vector DY. Empleando esta hipótesis se obtienen las desviaciones normalizadas (Z_i^*) en base a:

$$Z_i^* = \delta Y^* / DY \dots\dots\dots(21)$$

Lo cual es el último paso durante la lectura de datos. La sintaxis del procedimiento aparece en la figura 3.5.

```

fid=fopen('dpar.dat','rt')
BT=fscanf(fid,'%g',[kpr0,krdd])
BB=BT'
for ipr=1:kpr0
    BB(:,ipr)=BB(:,ipr)/DY(ipr)
end
    
```

Figura 3.5 Lectura de datos.

3.3.3 Formación del arreglo multidimensional

La matriz contenida debe ser interpretada correctamente: como se mencionó anteriormente, es una sola matriz de [9000 x 6], lo primero es organizar los paquetes de matrices para realizar cálculos por separado en cada régimen de operación, cada matriz contendrá los seis parámetros medidos en 21 puntos diferentes, cada una de estas matrices bidimensionales será para un defecto diferente, el conjunto de 9 defectos será para cada régimen de operación; al final serán 11 paquetes de datos, uno por cada régimen de operación. Esquemáticamente se puede ver en la figura 3.6, en donde 1 es el inicio de la matriz, 6 el número de filas (correspondientes a los parámetros medidos), 21 el número de columnas (desarrollo de la falla), 9 matrices (una por cada defecto) y 11 el número de regímenes de operación del motor.

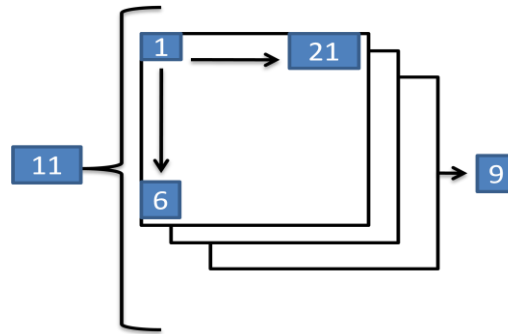


Figura 3.6 Esquema de organización de los datos.

Para MatLab se sigue el código mostrado en la figura 3.7, en donde los ciclos anidados forman parte fundamental del programa. Al final del acomodo de estos datos, se da la indicación de qué régimen de operación se debe utilizar en los cálculos posteriores.

```

for irg=1:krq
  for idf=1:kdf0
    for idl=1:kdl
      irdd=(irg-1)*kdf0*kdl+(idf-1)*kdl+idl
      B(:,idl,idf,irg)=BB(irdd,:)
    end
  end
end
% Regime determination
D=B(:, :, :, nreg)
  
```

Figura 3.7 Código de organización de datos.

3.3.4 Generación de vectores de entrada

En este paso del procedimiento se crearon, de manera estadística, por medio de una función gaussiana, nubes de puntos alrededor de una clase; en él se considera el número de puntos dado por el usuario. Esta creación de nubes permite expandir el espacio de análisis para considerar una ampliación en las mediciones del defecto, de manera que en un espacio tridimensional las clases pueden ser amplia y fácilmente identificables. Para este caso de análisis se considera la formación de clases múltiples y en la futura clasificación los puntos podrán pertenecer a una o más clases, agregando la intercepción de clases y por ende la clasificación del desarrollo de más de un defecto en paralelo. El código utilizado en MatLab se muestra en la figura 3.8.

El factor importante que determina la posición de los puntos generados son los números llave (randn), los cuales ayudan a generar un vector de números aleatorios; si estos números aleatorios son generados en diferente forma en cada experimento, sería difícil tener un punto de comparación, entonces, si estos números aleatorios son generados con el mismo número llave, aseguramos que, cambiando las condiciones de entrenamiento, se mantenga la misma nube de puntos y los resultados sean comparables.


```

ZE=ones(kpr0,1);
ZD_1=ones(kpr0,1);
ZD_2=ones(kpr0,1);
rand('state',nr1); randn('state',nr1);           % Números llave
if idf2==0
    kdf=kdf1;
else
    kdf=kdf2;
end;
for idf=1:kdf
    for itd=1:ktd
        ZE=randn(size(ZE))/3.0;
        dzn=randn/3.0;
        if idf2==0
            ld=rand*(kdl-1); ii=floor(ld)+1; alfa=ld-ii+1;
            ZD1(:,itd,idf)=D(:,ii,idf)*(1-alfa)+D(:,ii+1,idf)*alfa+ZE(:);
        else
            idf1=2*idf-1; idf2=2*idf;
            ld1=rand*(kdl-1); ld2=rand*(kdl-1);
            if (ld1+ld2)>(kdl-1)
                ld1=(kdl-1)-ld1; ld2=(kdl-1)-ld2;
            end;
            ii1=floor(ld1)+1; alfa1=ld1-ii1+1; ii2=floor(ld2)+1; alfa2=ld2-ii2+1;
            ZD_1(:)=D(:,ii1,idf1)*(1-alfa1)+D(:,ii1+1,idf1)*alfa1;
            ZD_2(:)=D(:,ii2,idf2)*(1-alfa2)+D(:,ii2+1,idf2)*alfa2;
            ZD1(:,itd,idf)=ZD_1(:)+ZD_2(:)+ZE(:);
        end
    end
end
end

```

Figura 3.8 Código de generación de vectores de entrada.

3.3.5 Generación de vectores de prueba

La diferencia entre estos vectores y los generados en el punto anterior, es la llave de inicialización de los generadores de nubes de puntos. Las llaves o semillas 1 y 2 son diferentes, ya que con la primera se entrena la red y con la segunda se prueba la misma, así podemos observar si la red entrenada obtiene probabilidades de aserción adecuadas. El código utilizado para la generación de los vectores de prueba se muestra en la figura 3.9.

```

rand('state',nr2); randn('state',nr2);           % State fixing of generators
for idf=1:kdf
  for itd=1:ktd
    ZE=randn(size(ZE))/3.0;
    dzn=randn/3.0;
    if idef2==0
      ld=rand*(kdl-1); ii=floor(ld)+1; alfa=ld-ii+1;
      ZD2(:,itd,idf)=D(:,ii,idf)*(1-alfa)+D(:,ii+1,idf)*alfa+ZE(:);
    else
      idf1=2*idf-1; idf2=2*idf;
      ld1=rand*(kdl-1); ld2=rand*(kdl-1);
      if (ld1+ld2)>(kdl-1)
        ld1=(kdl-1)-ld1; ld2=(kdl-1)-ld2;
      end;
      ii1=floor(ld1)+1; alfa1=ld1-ii1+1; ii2=floor(ld2)+1; alfa2=ld2-ii2+1;
      ZD_1(:,itd,idf1)=D(:,ii1,idf1)*(1-alfa1)+D(:,ii1+1,idf1)*alfa1;
      ZD_2(:,itd,idf2)=D(:,ii2,idf2)*(1-alfa2)+D(:,ii2+1,idf2)*alfa2;
      ZD2(:,itd,idf)=ZD_1(:)+ZD_2(:)+ZE(:);
    end;
  end;
end;
end;

```

Figura 3.9 Generación de vectores de prueba.

3.4 Formación de entradas a la red

Del total de los vectores generados, se debe especificar cuál de ellos se utilizara como vector de entrada ($P1$) y cuál de ellos será el vector objetivo ($T1$). La matriz de entrada $P1$ es una matriz de $R \times Q1$ con $Q1$ elementos representativos de los R vectores de entrada. Mientras que $T1$ es una matriz de $SN \times Q2$ con $Q2$ elementos representativos del SN elemento de los vectores objetivo. Se considero el total de defectos analizados (kdf) y el total de puntos generados en la nube (ktd), para el régimen específico mencionado al inicio del programa. El código utilizado para estos valores es mostrado en la figura 3.10.

```

for idf=1:kdf
  for itd=1:ktd
    n=(idf-1)*ktd+itd
    P1(:,n)=ZD1(:,itd,idf)
    T1(idf,n)=1
  end
end
for idf=1:kdf
  for itd=1:ktd
    n=(idf-1)*ktd+itd
    P2(:,n)=ZD2(:,itd,idf)
    T2(idf,n)=1
  end
end
end

```

Figura 3.10 Formación de entradas a la red.

3.5 Entrenamiento de la red (redes de retropropagación)

En la parte superior de la interface usuario se muestran los dos tipos de RNA utilizados (retropropagación y base radial). Para cada red, el algoritmo de entrenamiento puede ser seleccionado por el usuario.

El entrenamiento se detiene cuando alguna de las siguientes condiciones ocurre:

- 1) Se alcanza el máximo número de épocas.
- 2) El tiempo límite es alcanzado.
- 3) El valor mínimo de desempeño es alcanzado
- 4) El gradiente de desempeño cae por debajo de MINGRAD.
- 5) MU excede MU_MAX.
- 6) El desempeño de validación ha incrementado más que MAX_FAIL cuando se utiliza el valor de validación.

Este algoritmo puede entrenar cualquier red siempre y cuando la función de transferencia tenga funciones derivativas. A continuación se especifican todos los algoritmos de entrenamiento (funciones de Matlab) utilizados.

3.5.1 Algoritmo de entrenamiento Levenberg Marquart

Este es el método más rápido en el toolbox de MatLab, el problema es que utiliza grandes cantidades de memoria. Esta función de entrenamiento actualiza los pesos y las ganancias de acuerdo a la optimización Levenberg-Marquardt.

La retropropagación es usada para calcular el Jacobiano jX del desempeño con respecto a el peso y las ganancias variables X . Cada variable es ajustada de acuerdo al Levenberg-Marquardt,

$$\begin{aligned} jj &= jX * jX \\ je &= jX * E \\ dX &= -(jj+I*\mu) \setminus je \end{aligned} \dots\dots\dots (22)$$

en donde: E son los errores e I es la matriz identidad.

El valor adaptativo MU incrementa por medio de MU_INC hasta que el cambio de los resultados reduce el valor de desempeño. El cambio es hecho a la red y MU es disminuido por el valor de MU_DEC.

El parámetro MEM_REDUCE indica cómo usar la memoria y la velocidad para calcular el Jacobiano jX . Si MEM_REDUCE es 1, entonces TRAINLM corre rápido, pero puede requerir demasiada memoria. Incrementando MEM_REDUCE a 2, corta parte de la memoria requerida por un factor de 2, pero disminuye la velocidad. Los valores más altos continúan disminuyendo la cantidad de memoria necesaria e incrementa las veces de entrenamiento.

TRAINLM asume que la red neuronal tiene el error cuadrático medio como función de desempeño.

En la figura 3.11 se muestra la sintaxis del código para el algoritmo TRAINLM, la primera línea muestra la estructura básica de la red. La función newff crea una red de retropropagación, dentro del paréntesis, la primera indicación es la matriz de mínimos y máximos de el vector de entrada, dentro de los corchetes se indica el tamaño de la i-ésima capa para N capas, después se indican las funciones de transferencia utilizadas para cada capa, y por último la función de entrenamiento; esta primera línea es igual para todos los entrenamientos, lo que cambia es la función de entrenamiento que se utiliza. En la segunda línea se indica el numero de iteraciones o épocas, las cuales, como se menciono anteriormente, también es una condición de paro. En la tercera línea se pide la actualización de la gráfica acorde a cierto número de épocas.

```
net=newff(MiMa,[12,9],{'tansig','logsig'},'trainlm')
net.trainParam.epochs=100
net.trainParam.show=5
net.trainParam.mem_reduc=2
```

Figura 3.11 Código para entrenamiento por medio de trainlm.

3.5.2 Gradiente descendente

Este algoritmo ajusta las variables acorde al gradiente descendente:

$$dX = lr * dperf/dX \dots\dots\dots(23)$$

Las condiciones de paro son las mismas que para cualquiera de los algoritmos de retropropagación. En la figura 3.12 se muestra la sintaxis del código utilizado, la primera línea es exactamente igual a la sintaxis del método anterior, ya que el algoritmo sigue siendo del tipo retropropagación, la tasa de aprendizaje (lr) se multiplica por el gradiente negativo para determinar los nuevos valores de pesos y ganancias. Cuanto mayor sea la tasa de aprendizaje, mayor es el paso. Si la tasa de aprendizaje se hace demasiado grande, el algoritmo se vuelve inestable. Si la tasa de aprendizaje es demasiado pequeña, el algoritmo tarda mucho tiempo a converger. La tasa de aprendizaje es indicada en la segunda línea. Para este entrenamiento se requiere mayor numero épocas, por ende, para la actualización de la gráfica la muestra se toma con mayor intervalo.

```
net=newff(MiMa,[12,9],{'tansig','logsig'},'traingd')
net.trainParam.lr=0.1
net.trainParam.epochs=8000
net.trainParam.show=50
```

Figura 3.12 Sintaxis para el entrenamiento por medio de la función traingd.

3.5.3 Gradiente descendente con momento

Gradiente descendente con momento, ejecutado por `traingdm`, permite a una red responder no sólo al gradiente local, sino también a las tendencias recientes en la superficie de error. Actuando como un filtro de paso bajo, el momento de la red permite hacer caso omiso de pequeñas características en la superficie de error. Sin momento una red puede quedarse atrapada en un mínimo local. Con momento la red puede desplazarse a través de tal mínimo y salir de él. El parámetro MC es el momento constante que define la cantidad de momento. MC se encuentra entre 0 (sin movimiento) y valores cercanos a 1 (mucho movimiento). Con un momento constante de 1 la red es totalmente insensible a los gradientes locales y por lo tanto la red no aprende correctamente. La figura 3.13 muestra el algoritmo utilizado para el entrenamiento de la red con gradiente descendente con momento.

```
net =newff(MiMa,[12,9],{'tansig','logsig'},'traingdm')
net.trainParam.lr=0.1
net.trainParam.mc=0.5
net.trainParam.epochs=8000
net.trainParam.show=50
```

Figura 3.13 Sintaxis para el entrenamiento por medio de la función `traingdm`.

3.5.4 Tasa de aprendizaje adaptativa

Este tipo de red trata de mantener el tamaño del paso de aprendizaje tan grande como sea posible, manteniendo el aprendizaje estable. La tasa de aprendizaje se hace sensible a la complejidad de la superficie del error local. Un aprendizaje adaptativo requiere algunos cambios en el procedimiento de entrenamiento usado por `traingd`. Primero, se calcula la salida inicial de la red así como el error. En cada época los pesos y las ganancias son calculados usando la tasa de aprendizaje actual, después se calculan las salidas y los errores. Igual que en el caso en el que se tiene momento, si el nuevo error excede el anterior por más que una relación predefinida, `max_perf_inc` (típicamente 1.04), los nuevos pesos y ganancias son descartados. En adición, la tasa de aprendizaje es reducida (típicamente multiplicando por `lr_dec = 0.7`). De otra manera los valores de las variables son mantenidos. Si el nuevo error es menor que el anterior, la tasa de aprendizaje es incrementado (típicamente multiplicando por `lr_inc = 1.05`). El algoritmo que describe esta función de entrenamiento se presenta en la figura 3.14.

```
net =newff(MiMa,[12,9],{'tansig','logsig'},'traingda')
net.trainParam.lr=0.1
net.trainParam.lr_inc=1.05
net.trainParam.epochs=1000
net.trainParam.show=50
```

Figura 3.14 Sintaxis para el entrenamiento por medio de la función `traingda`.

3.5.5 Gradiente descendente con momento y tasa de aprendizaje adaptiva

La función `traingdx` combina el aprendizaje adaptativo con el entrenamiento con momento. La sintaxis es igual que para `traingda`, excepto que el coeficiente de momento es un parámetro adicional. La sintaxis es descrita en la figura 3.15.

```
net =newff(MiMa,[12,9],{'tansig','logsig'},'traingdx')
net.trainParam.lr=0.1
net.trainParam.mc=0.5
net.trainParam.lr_inc=1.02
net.trainParam.epochs=3000
net.trainParam.show=50
```

Figura 3.15 Sintaxis para el entrenamiento por medio de la función `traingdx`.

3.5.6 Retropropagación resiliente

El propósito del algoritmo de entrenamiento de retropropagación resiliente es eliminar los efectos perjudiciales de la magnitud de las derivadas parciales. Sólo con el signo de la derivada se puede determinar la dirección de actualización de los pesos; la magnitud de la derivada no tiene efectos en la actualización de los pesos. El tamaño de los pesos es determinado separando el valor actualizado. En general este algoritmo no requiere demasiadas épocas, por ende el tiempo de convergencia se ve seriamente beneficiado. En la figura 3.16 se muestra la sintaxis de este tipo de entrenamiento.

```
net =newff(MiMa,[12,9],{'tansig','logsig'},'trainrp')
net.trainParam.epochs=300
net.trainParam.show=20
```

Figura 3.16 Sintaxis para el entrenamiento por medio de la función `trainrp`.

3.5.7 Gradiente conjugado

Este método, así como los próximos 4, son métodos considerados como algoritmos de gradiente conjugado. Todos estos algoritmos comienzan por buscar la dirección más inclinada del gradiente descendente. Una línea de búsqueda es desarrollada para determinar la distancia óptima para moverse a lo largo de la actual dirección de búsqueda:

$$x_{k+1} = x_k + \alpha_k p_k \dots\dots\dots(24)$$

A continuación, se determina la nueva dirección, por lo que conjugando los resultados con la anterior búsqueda de direcciones. El procedimiento general para determinar la nueva dirección de búsqueda consiste en combinar la nueva dirección de ascendencia más pronunciada con la dirección de búsqueda anterior:

$$p_k = -g_k + \beta_k p_{k-1} \dots\dots\dots(25)$$

Existen varias versiones del algoritmo de gradiente conjugado, las cuales son disminuidas por la manera en la cual la constante β es calculada. Para la actualización por medio del procedimiento Fletcher-Reeves es:

$$\beta_k = \frac{g_k^T g_k}{g_{k-1}^T g_{k-1}} \dots\dots\dots(26)$$

Esta es la relación de la normal cuadrada del gradiente actual a la normal cuadrada del gradiente previo. Estos algoritmos son usualmente más rápidos que los de tasa de aprendizaje, sin embargo los resultados dependen del problema que se esté resolviendo. Estos algoritmos requieren un poco más de espacio de almacenamiento que los algoritmos más simples, así entonces, estos algoritmos son buenos para redes con gran número de pesos. La figura 3.17 muestra la sintaxis de este tipo de algoritmo.

```
net=newff(MiMa,[12,9],{'tansig','logsig'},'traincgf');
net.trainParam.epochs=200;
net.trainParam.show=20;
```

Figura 3.17 Sintaxis para el entrenamiento por medio de la función traincgf.

3.5.8 Gradiente conjugado propuesto por Polak y Ribière

Otra versión del algoritmo de gradiente conjugado fue propuesta por Polak y Ribière. Como con el algoritmo de Fletcher-Reeves, la dirección de búsqueda en cada iteración está determinada por la ecuación 17. Para la actualización de Polak-Ribière, la β constante es calculado por:

$$\beta_k = \frac{\Delta g_{k-1}^T g_k}{g_{k-1}^T g_{k-1}} \dots\dots\dots(27)$$

Este es el producto interior del cambio anterior en el degradado con el degradado actual, dividido por la norma al cuadrado del degradado anterior. La rutina de traincgp tiene un rendimiento similar a traincgf. Es difícil predecir qué algoritmo realizará mejor un determinado problema. Los requisitos de almacenamiento para Polak-Ribière (cuatro vectores) son ligeramente más grandes que para Fletcher-Reeves (tres vectores). La figura 3.18 muestra la sintaxis.

```
net=newff(MiMa,[12,9],{'tansig','logsig'},'traincgp');
net.trainParam.epochs=200;
net.trainParam.show=20;
```

Figura 3.18 Sintaxis para el entrenamiento por medio de la función traincgp

3.5.9 Gradiente conjugado propuesto por Beal

Para todos los algoritmos de gradiente conjugados, la dirección de búsqueda se restablece periódicamente a la dirección del gradiente negativo. El punto de restablecimiento estándar se produce cuando el número de iteraciones es igual al número de parámetros de red (pesos y sesgos), pero hay otros métodos de reinicio que pueden mejorar la eficiencia de la formación. Uno de esos métodos de restablecimiento fue propuesto por Powell, basado en una versión anterior, propuesta por Beale. Esta técnica se reinicia si hay muy poca ortogonalidad entre el gradiente actual y el anterior. Esto es probado con la siguiente desigualdad:

$$|g_{k-1}^T g_k| \geq 0.2 \|g_k\|^2 \dots\dots\dots (28)$$

Si se cumple esta condición, la dirección de búsqueda se restablece a la dirección negativa del gradiente. El código siguiente entrena la red usando la versión de Powell-Beale del algoritmo de gradiente conjugado. Los parámetros de la formación para `traincgb` son las mismas que las de `traincgf`. La figura 3.19 muestra la sintaxis utilizada en este tipo de entrenamiento.

```
net =newff(MiMa,[12,9],{'tansig','logsig'},'traincgb');
net.trainParam.epochs=200;
net.trainParam.show=20;
```

Figura 3.19 Sintaxis para el entrenamiento por medio de la función `traincgb`.

3.5.10 Gradiente conjugado degradado

Cada uno de los algoritmos de gradiente conjugados requiere una búsqueda de línea en cada iteración. Esta búsqueda de línea es costosa computacionalmente, porque requiere calcular varias veces la respuesta de la red para todas las entradas de formación. El algoritmo escalado conjugado degradado (CGS), desarrollado por Moller, fue diseñado para evitar la búsqueda en la línea de tiempo. Este algoritmo combina el enfoque de la región de modelo de confianza (utilizado en el algoritmo de Levenberg-Marquardt, descrito en "Levenberg-Marquardt (`trainlm`)", con el enfoque de gradiente conjugado. La rutina de `trainscg` puede requerir más iteraciones para converger que los otros algoritmos de gradiente conjugados, pero el número de cálculos en cada iteración se ha reducido significativamente porque no se lleva a cabo ninguna búsqueda de línea. Los requisitos de almacenamiento de información para el algoritmo de gradiente conjugado escalado son muy similares a las de Fletcher-Reeves. La sintaxis es mostrada en la figura 3.20.

```
net =newff(MiMa,[12,kdf],{'tansig','logsig'},'trainscg');
net.trainParam.epochs=200;
net.trainParam.show=20;
```

Figura 3.20 Sintaxis para el entrenamiento por medio de la función `trainscg`.

3.5.11 Método de Newton

Método de Newton es una alternativa a los métodos de gradiente conjugado para optimización rápida. El paso básico del método de Newton es:

$$x_{k+1} = x_k - A_k^{-1} g_k \dots\dots\dots(29)$$

donde A_k^{-1} es la matriz Hessiana (segunda derivada) del índice de rendimiento en los valores actuales de los pesos y las ganancias. El método de Newton a menudo converge más rápido que los métodos de gradiente conjugados, por desgracia, es complejo y costoso calcular la matriz Hessiana para redes neuronales. Hay una clase de algoritmos que se basa en el método de Newton, pero que no requieren cálculos de la segunda derivada. Estos son llamados métodos quasi-Newton (o secante), en los cuales se actualiza una matriz Hessiana aproximada en cada iteración del algoritmo. La actualización se calcula en función del degradado. Este algoritmo está implementado en la rutina de `trainbfg`. Los parámetros de la formación para `trainbfg` son los mismos que los de `traincgf`.

Este algoritmo requiere más cómputo en cada iteración y más almacenamiento de información que con los métodos de gradiente conjugado, aunque generalmente converge en menos iteraciones. Debe almacenarse el Hessiano aproximado, y su dimensión es $n \times n$, donde n es igual al número de pesos y ganancias en la red. Para redes muy grandes, sería mejor utilizar `rprop` o uno de los algoritmos de gradiente conjugados. Para redes más pequeñas, sin embargo, la `trainbfg` puede ser una función de formación eficaz.

```
net=newff(MiMa,[12,9],{'tansig','logsig'},'trainbfg');
net.trainParam.epochs=200;
net.trainParam.show=20;
```

Figura 3.21 Sintaxis para el entrenamiento por medio de la función trainbfg.

3.5.12 Método de la secante

Dado que el algoritmo BFGS requiere más almacenamiento y cómputo en cada iteración que los algoritmos de gradiente conjugados, existe la necesidad de una aproximación de la secante con menores requerimientos de almacenamiento de información y de cómputo. El método de la secante de un solo paso (OSS) es un intento de salvar la brecha entre los algoritmos de gradiente conjugados y los algoritmos de quasi-Newton (secante). Este algoritmo no almacena la matriz Hessiana completa; en cada iteración, el anterior Hessiano fue la matriz identidad. Esto tiene la ventaja adicional de que la nueva dirección de búsqueda puede ser calculada sin formar una matriz inversa. El código siguiente reinicializa la red anterior mediante el algoritmo de secante en un solo paso. Los parámetros de la formación para `trainoss` son los mismos que los de `traincgf` (figura 3.22). Requiere un poco más de almacenamiento de información y cómputo de épocas que los algoritmos de gradiente conjugados.

```
net =newff(MiMa,[12,9],{'tansig','logsig'},'trainoss');
net.trainParam.epochs=200;
net.trainParam.show=20;
end;
```

Figura 3.22 Sintaxis para el entrenamiento por medio de la función trainoss.

3.6 Redes de base radial

3.6.1 Diseño más eficiente

La función Newrb crea iterativamente una red de base radial de una neurona a la vez. Las neuronas se añaden a la red una a una hasta que la suma del error cuadrático cae por debajo de la meta o un número máximo de neuronas ha sido alcanzado. La función newrb toma matrices de entrada, vectores objetivo (P , T), parámetros de diseño (meta y propagación) y devuelve la red deseada.

En cada iteración se comprueba el error de la nueva red y si baja suficiente, newrb finaliza; de lo contrario se agrega la siguiente neurona. Este procedimiento se repite hasta que se cumpla la meta de error o se alcanza el número máximo de las neuronas.

Es importante que el parámetro de propagación sea lo suficientemente grande para que las neuronas de base radial respondan a la superposición de las regiones del espacio de entrada, pero no tan grande para que todas las neuronas respondan de la misma manera.

Las redes de base radial tienden a tener más neuronas en la capa oculta que una red de retropropagación comparable. Esto es porque las neuronas sigmoideas pueden tener salidas sobre una gran región del espacio de entrada, mientras que las neuronas de base radial sólo responden a regiones relativamente pequeñas del espacio de entrada. El resultado es que cuanto mayor sea el espacio de la entrada (en términos de número de entrada) más neuronas de base radial son necesarias. Por otra parte, diseñar una red de base radial a menudo toma mucho menos tiempo que la formación de una red sigmoidea lineal. La figura 3.23 muestra la sintaxis.

```
net =newrb(P,T,0.026,1,15,5)
```

Figura 3.23 Sintaxis para el entrenamiento por medio de la función Newrb.

3.6.2 Redes radiales probabilísticas

En el contexto de un problema de clasificación, si podemos construir las estimaciones de funciones de densidad de probabilidad (por sus siglas en inglés PDF) de las clases posibles, podemos comparar las probabilidades de las diferentes clases y seleccionar la más probable. Esto es efectivamente lo que pedimos hacer a una red neuronal cuando resuelve un problema de clasificación, la red intenta aprender una aproximación a PDF.

Un enfoque más tradicional es construir una estimación de las PDF a partir de los datos. La técnica más tradicional es asumir una cierta forma para la PDF (por lo general, una distribución normal) y luego para estimar los parámetros del modelo. La distribución normal utiliza comúnmente como parámetros del modelo media y desviación estándar, que pueden estimarse utilizando técnicas analíticas. El problema es que a menudo no está justificada la asunción de normalidad.

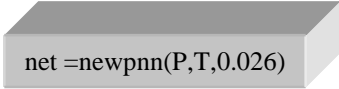
En las redes probabilísticas (probabilistic neural networks PNN), hay al menos tres capas: de entrada, radial y de salida. Las unidades radiales se copian directamente desde los datos de entrenamiento, una por cada caso. Existe una unidad de producción por clase. Cada una está conectada a todas las unidades radiales pertenecientes a su clase, con cero conexiones desde las unidades radiales de otros. Por lo tanto en las unidades de salida, simplemente se suman las respuestas de las unidades pertenecientes a su propia clase. Las salidas son proporcionales a las estimaciones basadas núcleo de los PDF's de las diversas clases, y por la normalización estos deben sumar a 1.

La PNN básica puede ser modificada de dos maneras. En primer lugar, el enfoque básico supone que la representación proporcional de las clases en los datos de entrenamiento coincide con la representación real en la población que está siendo modelada (las probabilidades antes de la llamada). En segundo lugar, cualquier red, para hacer las estimaciones sobre la base de una función que contiene ruido, inevitablemente produce algunos errores de clasificación. Sin embargo, algunas formas de errores de clasificación pueden considerarse como errores más costosos que otros. En tales casos, las probabilidades obtenidas por la red pueden ser ponderadas por los factores de pérdidas, que reflejan los costos de errores de clasificación. Una cuarta capa se puede especificar en las PNNs que incluye una matriz de pérdida. Esto se multiplica por las estimaciones de probabilidad en la tercera capa, y la clase con el menor costo estimado seleccionado (Pérdida matrices también pueden asociarse a otros tipos de red de la clasificación).

El factor de control único que tiene que ser seleccionado para el entrenamiento de redes neuronales probabilísticas es el factor de suavizado (es decir, la desviación radial de las funciones de Gauss). Al igual que con las redes RBR, este factor debe ser seleccionado para causar una cantidad razonable de superposición las desviaciones demasiado pequeñas pueden causar una aproximación muy específica que no se puede generalizar y desviaciones demasiado grandes eliminaran los detalles.

Las mayores ventajas de las PNNs son el hecho de que la salida es probabilística (que hace que la interpretación de la salida fácil) y la velocidad del entrenamiento. La formación de una PNN en realidad consiste en el entrenamiento de una copia de la red, por lo que es lo más cercano que se puede esperar a la función real.

El mayor inconveniente es el tamaño de la red: en realidad contiene el conjunto de casos de entrenamiento, y por lo tanto el espacio es amplio y la ejecución se vuelve lenta. Las PNNs son particularmente útiles para hacer el prototipo de experimentos; como el tiempo de formación es de corta duración, permite realizar un gran número de estudios en un período corto de tiempo.



```
net =newpnn(P,T,0.026)
```

Figura 3.24 Sintaxis para el entrenamiento por medio de la función newpnn.

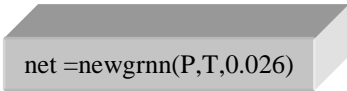
3.6.3 Regresión generalizada

Las redes neuronales de regresión generalizada (GRNNs) trabajan en forma similar a PNNs. Al igual que con las PNNs, las funciones del kernel Gaussiano se encuentran en cada caso de formación. Cada caso puede ser considerado, como prueba de que la superficie de respuesta es una distancia determinada a ese punto en el espacio de entrada, con descomposición progresiva de pruebas en las inmediaciones. La formación de las copias GRNN se utiliza para estimar la respuesta de los nuevos puntos. La salida se calcula utilizando una media ponderada de los resultados de los casos, en donde se cuenta la ponderación de la distancia del punto del punto que se estima (de modo que puntos cercanos más contribuyen en gran medida a la estimación).

La primera capa oculta en las GRNNs contiene las unidades radiales. Una segunda capa oculta contiene unidades que ayudan a calcular la media ponderada (este es un procedimiento especializado): cada salida tiene una unidad especial asignada en esta capa, que forma la suma ponderada de la salida correspondiente, para obtener el promedio ponderado de la suma ponderada, la suma se divide por la suma de los factores de ponderación. Una sola unidad especial en la segunda capa calcula el valor de esta última suma. A continuación, la capa de salida realiza la división real (con unidades especiales de división). Por lo tanto, la segunda capa oculta siempre tiene exactamente una unidad más que la capa de salida.

Las GRNN pueden ser modificados mediante la asignación de unidades radiales que representan a grupos en lugar de cada caso individual de formación: se reduce el tamaño de la red y aumenta la velocidad de ejecución. Centros pueden ser asignados usando cualquier otro algoritmo adecuado (es decir, sub-muestreo, K-medias o de Kohonen).

Un GRNN tiende a ser grande y lenta aunque, a diferencia de las PNNs, no es necesario contar con una unidad radial para cada caso, aún así el número todavía tiene que ser grande. Al igual que una red RBF, una GRNN no extrapola. La sintaxis para la utilización de este tipo de funciones se observa en la figura 3.25.



```
net =newgrnn(P,T,0.026)
```

Figura 3.25 Sintaxis para el entrenamiento por medio de la función newgrnn.

3.7 Generalidades

Independientemente del tipo de red, todas necesitan especificar el valor máximo de error permitido y por último la sintaxis del entrenamiento de la red. El comando TRAIN entrena

una red neuronal acorde a una función de entrenamiento y sus respectivos parámetros, los cuales fueron especificados previamente. La sintaxis para el entrenamiento se muestra en la figura 3.26.



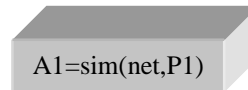
```
[net,tr]=train(net,P1,T1,[ ],[ ],val)
```

Figura 3.26 Entrenamiento de la red.

El entrenamiento se da hasta que un máximo número de épocas es alcanzado, el error máximo permitido es alcanzado o alguna otra de las condiciones de paro ocurra.

3.8 Simulación

La sintaxis para la simulación de la red es sencilla, en ella se llama a la red creada en el paso anterior y se simula con los vectores de entrada. La sintaxis se puede observar a continuación.



```
A1=sim(net,P1)
```

Figura 3.27 Simulación de la red neuronal.

3.9 Cálculo de probabilidades

Los valores de probabilidad de diagnóstico nos dan una validación del entrenamiento de la red. Esto se hace por medio de un procedimiento estadístico. Se calcularon dos matrices de probabilidades de diagnóstico. La primera matriz se determina para el entrenamiento y la segunda matriz es calculada con los datos prueba. Ésta última dará una probabilidad de diagnóstico más acertada, ya que demuestra cual es el comportamiento de la red en condiciones de trabajo diferentes a las que fue entrenada. El código utilizado para hacer el cálculo de probabilidades de diagnóstico se observa en la figura 3.28.

```

PD2=zeros(kdf)
for idf=1:kdf
    for itd=1:ktd
        n=(idf-1)*ktd+itd
        [amax,imax]=max(A2(:,n))
        PD2(imax,idf)=PD2(imax,idf)+1
    end
end
PD2=PD2/ktd
PDT2=diag(PD2)
psr2=mean(PDT2)
PD2T=PD2'
fw2=fopen('pd2.dat','wt')
if idef2==0
    count=fprintf(fw2,'%6.4f %6.4f %6.4f %6.4f %
        6.4f %6.4f %6.4f %6.4f %6.4f\n'
        ,PD2T,PDT2,psr2)
else
    count=fprintf(fw2,'%6.4f          %6.4f          %6.4f
%6.4f\n',PD2T,PDT2,psr2)
end
fclose(fw2)

```

Figura 3.28 Código para el cálculo de probabilidades.

3.10 Gráfica de clases

Es posible visualizar las clases en un espacio multidimensional, en una gráfica de este tipo se pueden considerar las nueve clases. En orden de hacer más claras y diferenciables las clases sólo se incluyeron 4 clases por gráfico, como se muestra en la figura 3.29. Los ejes pertenecen a la presión en el compresor, presión en la turbina y temperatura del compresor.

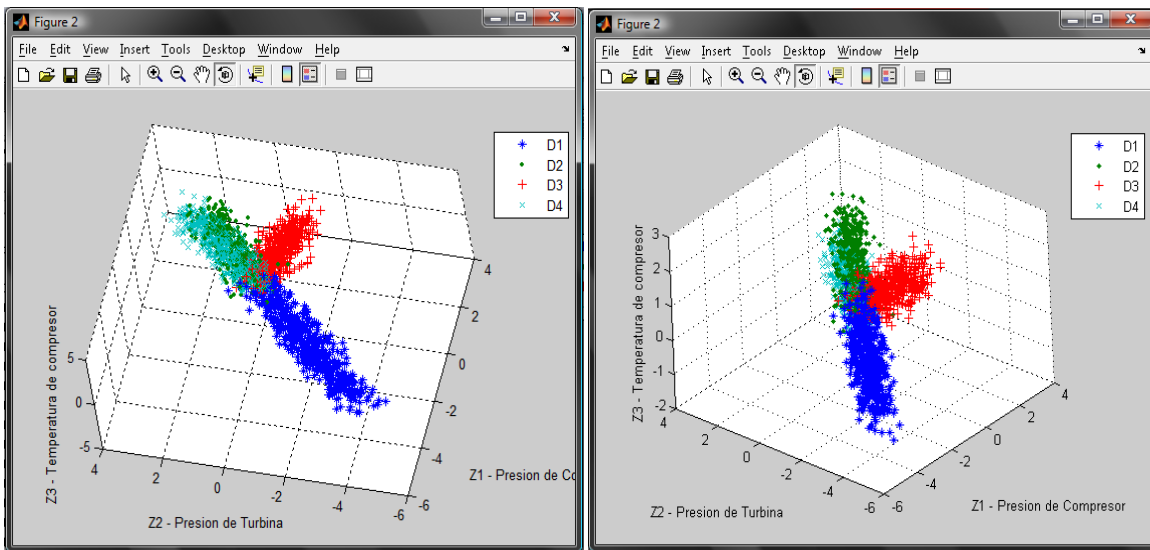


Figura 3.29 Gráfica de clases, diferentes ángulos para tener diferentes vistas de las clases.

El código utilizado para graficar es mostrado en la figura 3.30. Considerando ésta como la “figura (2)” porque la “figura (1)” es la gráfica de desempeño de la red.

```

figure(2);
plot3(ZD1(1, :, 1), ZD1(2, :, 1), ZD1(3, :, 1), '*', ZD1(1, :, 2), ZD1(2, :, 2),
ZD1(3, :, 2), '!', ZD1(1, :, 3), ZD1(2, :, 3), ZD1(3, :, 3), '+',
ZD1(1, :, 4), ZD1(2, :, 4), ZD1(3, :, 4), 'x')
grid on
legend('D1', 'D2', 'D3', 'D4')
xlabel('Z1 - Presión de Compresor')
ylabel('Z2 - Presión de Turbina')
zlabel('Z3 - Temperatura de compresor')

```

Figura 3.30 Código para graficar las clases.

3.11 Interface usuario

Todos los procesos mencionados anteriormente son procesos internos que no cambian en cada experimento, lo que se cambia son las condiciones, las cuales pueden ser sencillamente alteradas por medio de la interface usuario creada. Con esta interface se optimiza el tiempo de cálculo y se muestran todos los resultados en una sola ventana. La figura 3.31 muestra esta interface, en la parte superior de la ventana se muestran las constantes que deben ser introducidas por el usuario y el tipo de red a utilizar; una vez que los cálculos son terminados en la misma ventana, parte inferior, aparece la probabilidad y el tiempo de ejecución (ambos parámetros necesarios para la comparación).

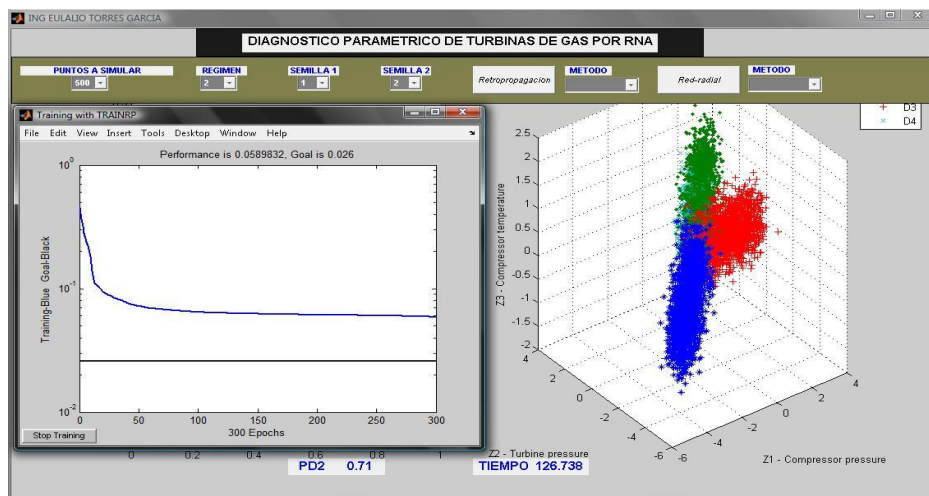


Figura 3.31 Interface de control para la experimentación.

Conclusiones

Los lenguajes de programación representan una herramienta muy poderosa en la mayoría de los campos de ingeniería, facilitan el trabajo e incrementan la posibilidad de desarrollo de más tareas. Los algoritmos presentados en este capítulo fueron codificados por medio de Matlab. Se presentó el desarrollo del algoritmo paso a paso, las especificaciones de cada uno de los parámetros y de cada método utilizado fueron explicadas. Los métodos tienen un fundamento matemático, el cual fue presentado en la mayoría de los casos, en algunos fue obviado ya que era parecido a alguno de los métodos presentados previamente.

CAPÍTULO IV
EXPERIMENTACIÓN Y
RESULTADOS

Introducción

La experimentación consiste en una serie de repeticiones de entrenamiento de las redes que se realiza para observar el comportamiento durante cada experimento bajo diferentes condiciones. Los factores que pueden ser cambiados son: régimen de operación, número de puntos de simulación, número llave de iniciación de los generadores de vectores aleatorios, valores propios de cada función de entrenamiento, etc. Variando estos factores escogemos los mejores para cada red neuronal seleccionada. Las redes optimizadas de esta manera se comparan finalmente para escoger la mejor en la aplicación al diagnóstico de turbinas de gas.

4.1 Procedimiento de experimentación

Cada tipo de red neuronal se llevó a cabo por etapas. En la primera se introdujeron valores a cada función de entrenamiento, con ello se obtuvo un comportamiento inicial; al ver que los tiempos, el desempeño y las probabilidades eran muy variadas se hicieron ajustes en cada una de ellas cambiando el número de neuronas, el número de capas, el número de épocas, la tasa de aprendizaje, el coeficiente de momento, etc. para, disminuir el tiempo de convergencia y aumentar la probabilidad de diagnóstico correcto.

Al cabo de varias iteraciones se lograron valores parecidos, todas estas iteraciones se hicieron con un número constante de puntos de simulación y un régimen constante; para el caso de esta optimización el régimen utilizado fue el número 4 y 500 puntos de simulación por clase.

Con los valores de tiempo y probabilidad de diagnóstico optimizados se hicieron las mismas pruebas a diferentes regímenes de operación, para determinar si el comportamiento de la red era estable o no. Después se incrementó el número de puntos de simulación, es importante recordar que entre mayor sea el número de puntos, el criterio de clasificación es más amplio y más exacto, pero al mismo tiempo se incrementa la complejidad del sistema y es más difícil lograr una red precisa.

La siguiente etapa fue el cambio de números de semilla o números llave, como se mencionó, la generación de puntos al azar depende de estos números llave. Si los números se cambian a pesar de que la clase sea la misma, los puntos estarán en diferente posición. Entonces, la red debe ser capaz de identificar de igual manera las clases, independientemente de dónde se generen los puntos.

La experimentación con redes radiales de diseño más eficiente se comenzó con un gran número de neuronas, empezando con 200, y terminado con un valor de 10; estos experimentos iniciales se corrieron con hasta 500 puntos de simulación, siendo éstos dos factores dependientes (número de puntos y neuronas), si el número de puntos de simulación es grande, el número de neuronas debe ser pequeño y viceversa. Después se hizo el mismo experimento con diferentes regímenes de operación y diferentes números llave.

Para el desarrollo de las redes probabilísticas así como de regresión generalizada, se hizo el experimento cambiando la cantidad de puntos de simulación, como se mencionó en el capítulo 3. Este tipo de redes requieren grandes cantidades de memoria, así que la opción

fue reducir el número de puntos de simulación, hasta el máximo soportado por el computador (con 2 Gb de memoria y 1.8 GHz en velocidad de procesador). Al disminuir el número de puntos es necesario incrementar el número de experimentos, así se puede asegurar que aunque el número de puntos es pequeño, la probabilidad de aserción es la misma que con una red con más de 500 puntos de simulación. La siguiente fase es el cambio de llave para una vez más probar que la red es capaz de clasificar en cualquier ambiente.

4.2 Resultados y comparación, retropropagación

4.2.1 Optimización

La tabla 4.1 muestra los resultados obtenidos para la red de retropropagación, entre los valores iniciales de la red sin optimizar y los de la red optimizada. La tabla incluye todas las variables que podrían ser de importancia para la comparación, como son: tiempo, desempeño (referente al error cuadrático medio ECM), número de épocas, probabilidad con la matriz de vectores de entrenamiento (PD1) y probabilidad con la matriz de vectores de prueba (PD2). Se tienen los resultados para los 12 algoritmos de entrenamiento.

Todos estos experimentos fueron hechos manteniendo el mismo régimen (el número 4) y la misma cantidad de puntos de simulación (500). Los valores marcados con color son aquellos que mostraron menor tiempo y mejor desempeño.

Variante	Método	1	2	3	4	5	6	7	8	9	10	11	12
		trainlm	traingd	traingdm	traingda	traingdx	trainrp	traingcf	traingcp	traingcb	traingcg	traingbf	traingss
Inicial	Tiempo (min)	10.53	15.41	15.20	1.59	6.12	1.15	1.46	1.27	1.21	1.14	1.07	1.41
	ECM	0.0359	0.05	0.05	0.02814	0.02706	0.0272	0.0362	0.04418	0.02709	0.02736	0.054	0.0639
	PD1	0.7451	0.715	0.7142	0.8274	0.827	0.826	0.7491	0.6641	0.8272	0.8262	0.5758	0.4482
	PD2	0.745	0.714	0.7136	0.8178	0.8203	0.8198	0.7417	0.6624	0.8176	0.82216	0.5698	0.4869
Final	Tiempo (min)	7.1400	9.33000	17.4500	1.2400	1.5500	0.3700	1.3600	2.5200	1.0700	1.0500	4.3200	12.0500
	ECM	0.03	0.03130	0.0310	0.0229	0.0279	0.0274	0.0352	0.0276	0.0274	0.0280	0.0274	0.0310
	PD1	0.7444	0.81230	0.8129	0.8220	0.8268	0.8258	0.7507	0.8264	0.8257	0.8241	0.8270	0.8096
	PD2	0.7332	0.81120	0.8100	0.8176	0.8189	0.8202	0.7462	0.8180	0.8204	0.8208	0.8209	0.8067

Tabla 4.1 Tabla comparativa de las redes de retropropagación con y sin optimización.

La llamada gráfica de “performance” o de desempeño se refiere al error cuadrático medio (ECM); si el desempeño reduce, el error se reduce y la probabilidad de diagnóstico correcto aumenta. Por lo tanto, la comparación entre ambas redes se puede hacer o en función del desempeño o en función de la probabilidad. El número de épocas es dependiente del tipo de red, pero este número no es directamente definitorio del tiempo de entrenamiento. Aunque algunos algoritmos tienen demasiadas épocas, el tiempo de entrenamiento al final es comparable. La probabilidad de diagnóstico que es de mayor interés es la probabilidad PD2, puesto que es la probabilidad calculada con los vectores de prueba. Si se considera

sólo la probabilidad PD1, se observa que puede tener un valor demasiado elevado por el posible efecto de sobreentrenamiento.

Como se puede observar en la tabla 4.1 si el ECM es de alrededor de $0.027 \pm 0,003$ se alcanza una probabilidad de $80 \pm 2 \%$. Para comparaciones posteriores se utilizara el ECM para la comparación.

En la figura 4.1 se muestra la comparación entre el desempeño inicial de la red sin optimizar (curva superior) y con la red optimizada (curva inferior). Como se puede observar casi todos los algoritmos alcanzaron después de la optimización un desempeño menor a 0.03 de ECM.

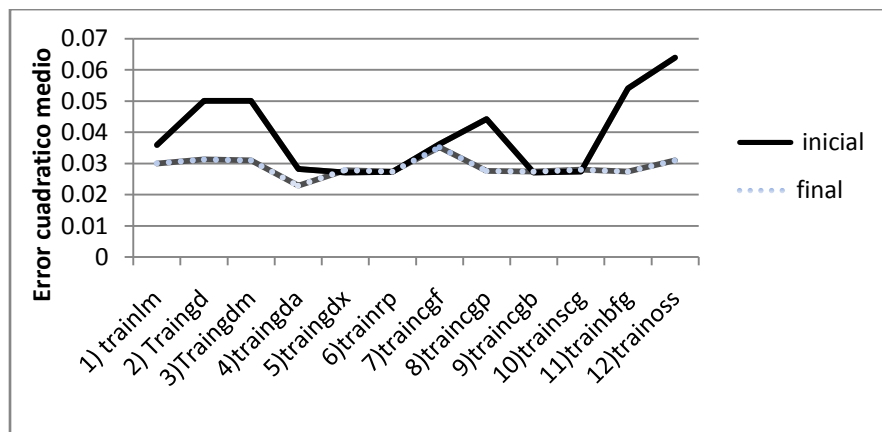


Figura 4.1 Comparación en el desempeño en función del error cuadrático medio (ECM) inicial y el optimizado (final).

De la gráfica anterior se observa que si el error es alto, la probabilidad es baja. Por ejemplo, para el algoritmo de la secante (trainoss) cuando el performance es de alrededor de 0.065, la probabilidad es de 50%. Una vez que la optimización de las redes fue hecha, los valores de probabilidad son comparables (figura 4.2). Hasta este momento se puede decir que de los 12 métodos, son 11 los que tienen el mismo nivel de probabilidad y el mismo valor de ECM, quedando traingcf descartado.

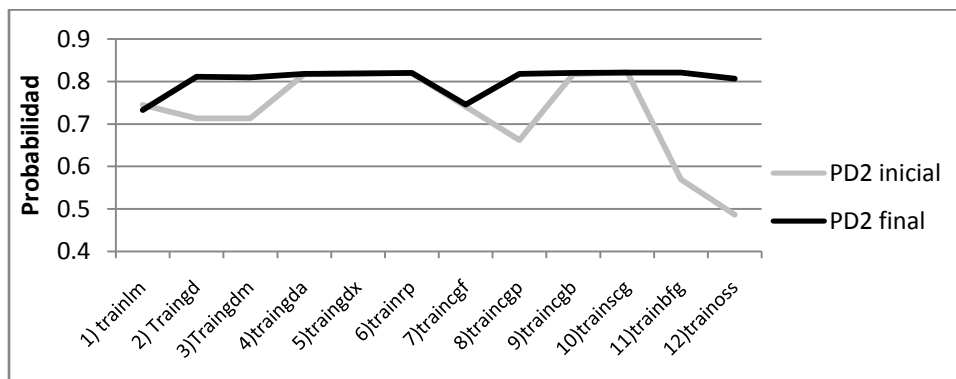


Figura 4.2 Comparación en la probabilidad antes y después de la optimización.

El siguiente elemento a comparar es el tiempo de entrenamiento, el cual es el principal discriminante. En un principio algunos algoritmos se tardaban alrededor de 20 minutos para el entrenamiento, y aunque se buscó reducir el tiempo, aún sigue siendo grande la diferencia entre los que llevan mayor y menor tiempo de entrenamiento. En la figura 4.3 se muestra la comparación en tiempo antes y después de la optimización. En este caso los algoritmos con función de entrenamiento trainlm, traingd, traingdm y trainoss son los que requieren mayor tiempo de entrenamiento.

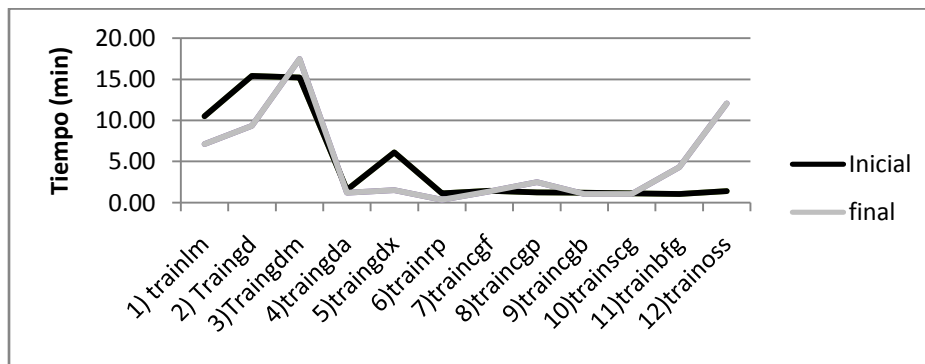


Figura 4.3 Comparación de tiempo de entrenamiento.

Del total de los 12 métodos solo 5, con valores tanto de desempeño como de tiempo mínimos, fueron seleccionados; en la figura 4.4 se muestran estos resultados. El performance alcanzado se encuentra alrededor de 0.025, mientras que el tiempo varía entre 0.5 y 2.5 minutos. El ECM en los 5 métodos seleccionados es suficiente para arrojar probabilidades cercanas al 80 %.

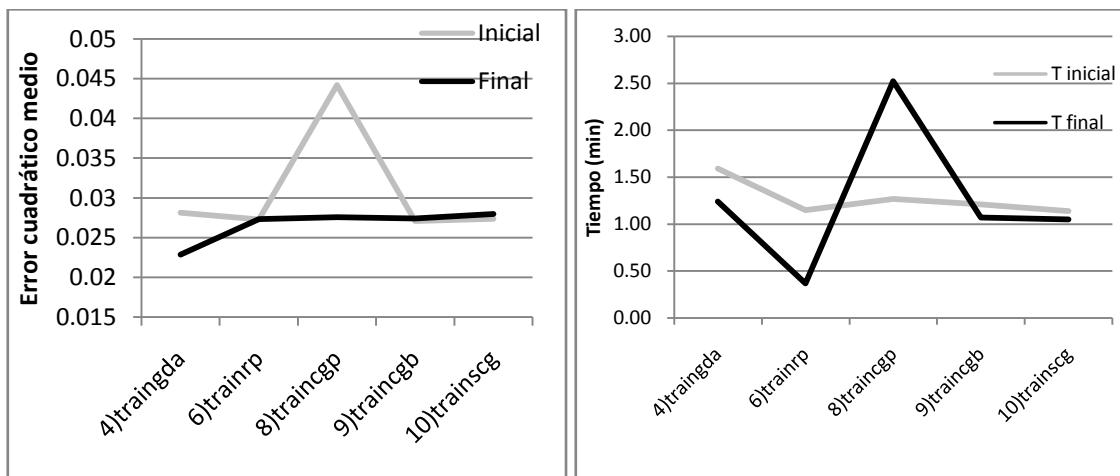


Figura 4.4 Lado izquierdo comparación de 5 métodos con mejor menor ECM alcanzado, del lado derecho el tiempo para los mismos 5 métodos.

4.2.2 Incremento de número puntos de simulación

De los 5 métodos que resultaron más eficientes, solo 4 se tomaron para la siguiente comparación, debido a que el método traingcp mostro un ECM y un tiempo mayor, en este

caso se tomaron los métodos con tiempos menores a dos minutos: `traingda`, `trainrp`, `traingb`, `traincg`. Se hizo el aumento de puntos de simulación. Si una red se comporta estable es utilizable en esta aplicación, de lo contrario debe ser descartada, pues no se puede utilizar en diagnóstico una red neuronal que solo haga la clasificación adecuada con cierto número de puntos de simulación.

La figura 4.5 muestra la comparación con 4 diferentes regímenes de operación. En las gráficas se observa el incremento del número de puntos de simulación, manteniendo constante el método de entrenamiento y el régimen de operación.

Los resultados muestran que de las redes seleccionadas, no todas tienen igual desempeño. Si observamos en la figura 4.5, la red entrenada con la función `traincg` es inexacta para simulaciones con más de 2000 puntos, mientras que `traingda` y `trainrp` se mantiene estables en todos los regímenes de operación y en todos los puntos de simulación. Por otro lado, la función `traingb` es inestable, en el primer régimen con 3000 puntos de simulación el error se incrementa, en el régimen 3 es en 1500 donde se incrementa este error y en el régimen 11, el error disminuye con el incremento de puntos.

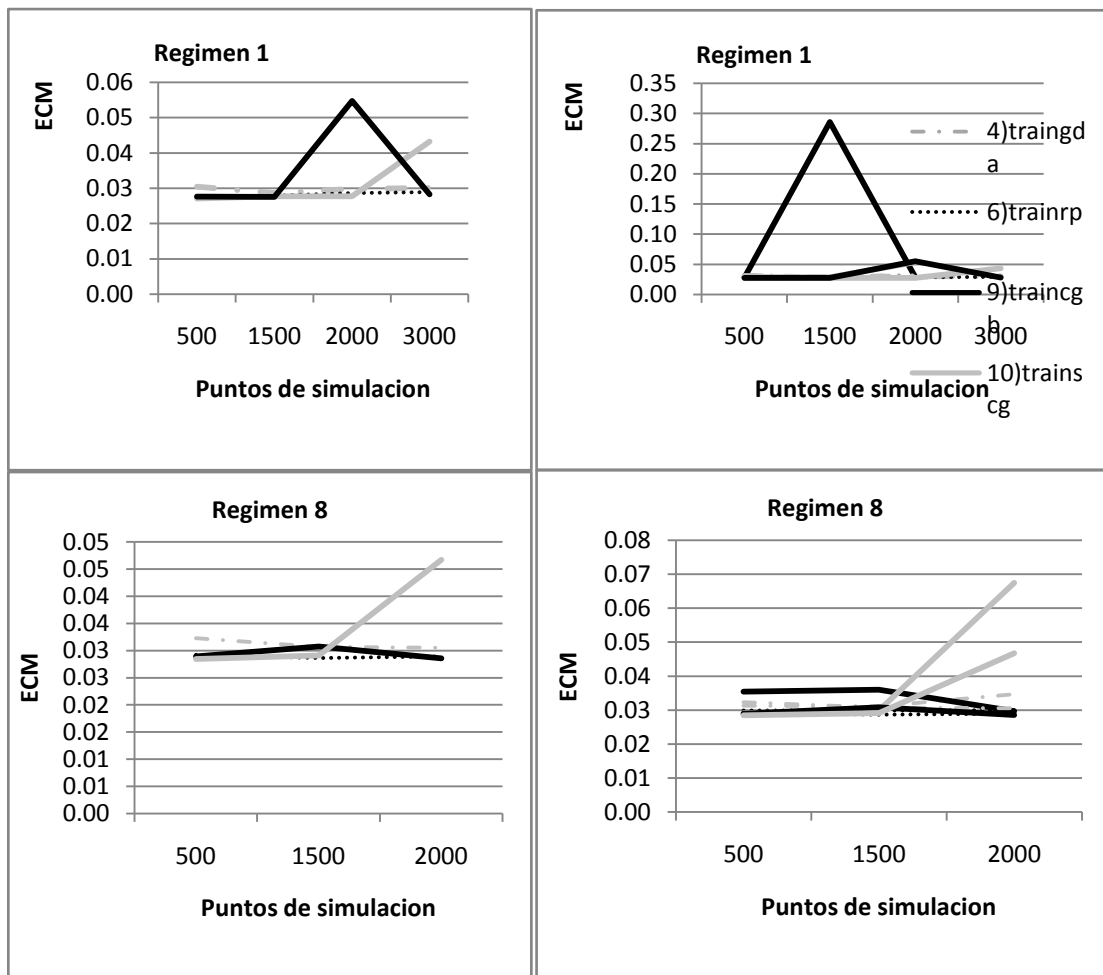


Figura 4.5 Comparación de cambio de régimen e incremento del número de puntos de simulación.

Es posible decir que las redes entrenadas con `trainrp` y `traingda`, se mantienen indiferentes con el aumento de puntos de simulación, mientras que las otras redes dependen íntegramente de las condiciones de entrenamiento, es decir, pierden su capacidad de generalización. Si el grado de exactitud que se requiere es un poco menos exigente, estas redes pueden ser tomadas en cuenta, pues aunque el error aumenta, sigue dando probabilidades del 75 %.

4.2.3 Cambio de semilla (número llave)

Como se explicó anteriormente, la generación de puntos es dependientes de los números de inicialización, así aunque las nubes de puntos están generadas alrededor de cada clase identificada, tendrán una posición dependiendo del número llave de inicialización, si se cambia el número de inicialización los puntos estarán generados en una posición diferente. Al entrenar la red con diferentes semillas se puede concluir si la red es tolerante para la aplicación en diagnóstico de turbinas de gas.

En apartado anterior se mostró cuál es el comportamiento con números llave 1 y 2, en este caso los números llave fueron cambiados por 2 y 3. Se puede observar cuál es el comportamiento de la red bajo diferentes regímenes de operación y con distinta cantidad de puntos de simulación.

Tanto la red entrenada con la función `trainscg` como la red con función `traincgb` son redes que presentaron la mayor cantidad de irregularidades: en algunos regímenes el comportamiento es aparentemente estable, mientras que en otros incrementa demasiado el valor del error. En la figura 4.6 se puede ver que en el régimen 1 son más notorias las inestabilidades. En las redes entrenadas con funciones `traincgb` y `trainscg`, el error cuadrático medio incrementa hasta el 0.055, valor que nos lleva a obtener probabilidades de aserción del 70%. Estas son mucho menores que en las redes con error cuadrático medio de 0.027, con el cual se alcanzan probabilidades de 80 a 82 %.

En el régimen 3, la función `traincgb` tiene mayor valor de error cuadrático medio. Las dos funciones que tienen mayores irregularidades en el entrenamiento son las mismas que en el apartado anterior. Para los regímenes 8 y 11 las mismas funciones incrementan el error con el incremento del número de puntos de simulación.

Las funciones `trainrp` y `traingda` tiene un comportamiento estable en los 4 regímenes de operación tomados como prueba. La función `trainrp` mantiene el valor del error cuadrático medio constante en todos los regímenes y con todos los cambios de semilla. Aunque en la función `traingda` incrementa un poco el valor del error, éste es capaz de arrojar probabilidades del 80 %.

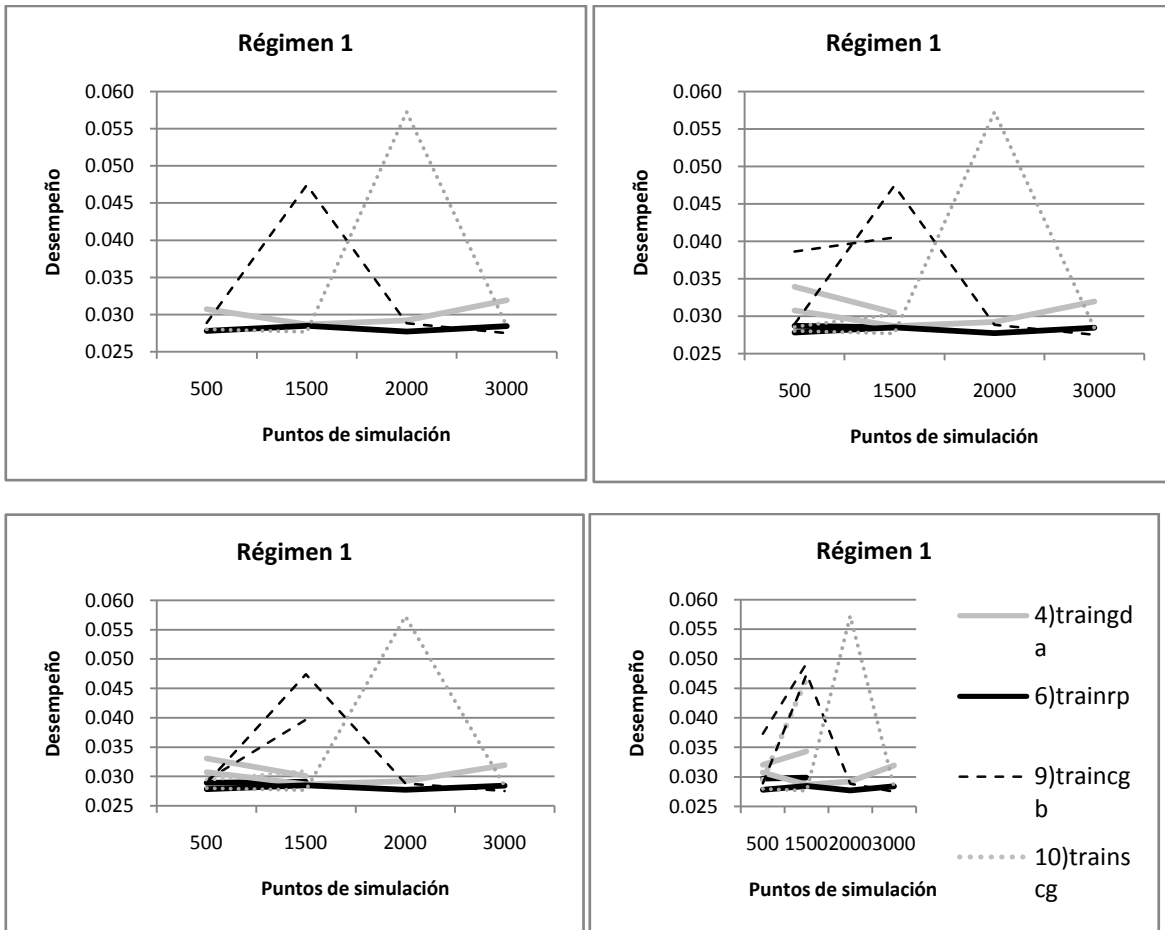


Figura 4.6 Error cuadrático medio (performance) en diferentes regímenes de operación con cambio de semilla.

4.3 Resultados y comparación, cálculos con las redes de base radial

4.3.1 Red de diseño mejorado

En la teoría se mencionó que las redes de base radial de diseño exacto requieren igual número de neuronas de entrenamiento como vectores de entrada. En el caso de estudio se requieren grandes bases de datos, que requieren grandes cantidades de memoria y gran velocidad de procesamiento de cómputo, razón por la cual las redes de diseño exacto no son factibles. Por otro lado, las redes de base radial de diseño mejorado permiten reducir el número de neuronas, aplicando una neurona por cada iteración; así, si se tiene 100 neuronas, se tendrá el mismo número de iteraciones, en esta etapa de experimentación, se hicieron pruebas con diferente número de puntos de operación y con diferente número de neuronas.

En la tabla 4.2 se muestran algunos resultados de la experimentación, en ella se mantiene el régimen constante y los números llave constantes. Los parámetros cambiados fueron: número de neuronas y número de puntos de simulación. Los resultados demuestran que con 200 neuronas se da el fenómeno de sobre-entrenamiento, y el tiempo de entrenamiento se

incrementa hasta 20 minutos. Con 200 neuronas se logran probabilidades de alrededor de 80% más sin embargo es la misma probabilidad que se logra con 20 neuronas. Como se puede observar, con 10 neuronas la probabilidad es baja, entonces no son suficientes para poder clasificar el volumen de datos. Con un número de puntos entre 200 y 500 se logró la misma probabilidad con una desviación de ± 1 %. Se puede observar que 100 neuronas es un valor aceptable, más sin embargo el tiempo de entrenamiento es grande y con neuronas de 20 a 25 se puede lograr la misma probabilidad.

puntos	10 neuronas		100 neuronas		200 neuronas	
	PD2	performance	PD2	performance	PD2	performance
200	0.6828	0.05066	0.81	0.02679	0.8061	0.02604
300	0.687	0.05209	0.8044	0.02948	0.7978	0.026965
500	0.6853	0.04988	0.8038	0.02877	0.8064	0.026774

Tabla 4.2 Newrb con cambio en el número de neuronas y aumento en el número de puntos de simulación. Régimen 3, números llave 1 y 2, constantes.

En la siguiente etapa de experimentación se vio la influencia del cambio de régimen de operación y cambio de semilla de ambos generadores aleatorios. Los experimentos se llevaron a cabo con 20 neuronas y 500 puntos de simulación por clase. Con mayor cantidad de puntos los recursos de cómputo se hicieron insuficientes y con menor cantidad los resultados no serían fiables.

Observando la tabla 4.3 se hace claro que este tipo de red es indiferente al cambio de posición de los puntos de simulación, y al régimen de operación. Todas ellas dan probabilidades de alrededor del 80% y tiempos de entrenamiento de 120 segundos. Los resultados mostrados son el promedio de 5 iteraciones en la misma condición, esto se hizo con el fin de aumentar la fiabilidad de los cálculos ya que con menor cantidad de puntos es mayor la probabilidad de falla, pero si se repite el experimento n veces y se promedia el resultado podemos asegurar que el comportamiento será el mismo.

4.3.2 Red probabilística

El diseño de estas redes requiere un parámetro extra, dispersión (spread), el cual si es cercano a cero actuará como clasificador en la cercanía de la clase, si el valor de la dispersión se hace mayor tomará en cuenta varios vectores con distancias un poco más alejadas de las clases. Se requiere que el valor de los vectores no sea muy pequeño: si éste es cero solo clasificará exactamente sobre la línea de la clase, si el valor de la dispersión es muy grande ésta se vuelve insensible al cambio de las clases y puede no clasificar adecuadamente.

Esta parte de la experimentación se comenzó con un valor de dispersión de 0.026 y 500 puntos de simulación. Como se puede observar este tipo de redes, al igual que la anterior, es insensible al cambio de semilla y al cambio de régimen. El valor de las probabilidades esta dentro de un rango de 0.71 a 0.75 lo cual es constante aún con el cambio de régimen como se puede observar en la tabla 4.4.

Régimen	semilla 1,2		semilla 2,1		semilla 1,3		semilla 2,3		semilla 3,1		semilla 3,2	
	tiempo	PD2	tiempo	PD2	tiempo	PD2	tiempo	PD2	tiempo	PD2	tiempo	PD2
1	121.571	0.8	119.589	0.804	120.76	0.7942	119.979	0.7936	121.461	0.8104	118.591	0.8002
2	121.68	0.798	120.463	0.7969	119.886	0.7924	124.593	0.7927	120.167	0.8071	135.715	0.7993
3	121.196	0.7942	120.962	0.7996	119.87	0.7889	120.307	0.792	119.886	0.8051	127.878	0.7987
4	123.566	0.7942	127.187	0.7938	119.964	0.7893	119.169	0.7891	118.342	0.8064	121.821	0.7989
5	122.336	0.7953	121.257	0.8009	119.777	0.7893	120.51	0.7913	124.831	0.8071	124.16	0.7989
6	121.586	0.7904	119.714	0.7991	122.258	0.7858	120.12	0.7887	119.075	0.8036	122.444	0.7951
7	122.491	0.7944	119.168	0.7991	121.537	0.7889	120.588	0.7889	120.869	0.8033	121.992	0.7942
8	121.228	0.7927	120.073	0.7989	120.198	0.7909	119.262	0.7873	119.636	0.806	121.087	0.7942
9	126.172	0.7909	120.931	0.7991	119.449	0.7876	120.666	0.788	118.014	0.8078	128.934	0.7936
10	120.369	0.7931	120.182	0.7964	119.262	0.788	124.769	0.7824	118.997	0.8062	126.016	0.792
11	120.775	0.7902	121.914	0.7909	120.401	0.7871	123.021	0.7833	121.29	0.8002	121.227	0.7898

Tabla 4.3 Comparaciones con cambio de semilla y cambio de régimen de operación. 500 puntos de simulación por clase.

Régimen	semilla 1,2		semilla 2,1		semilla 1,3		semilla 2,3		semilla 3,1		semilla 3,2	
	tiempo	PD2	tiempo	PD2	tiempo	PD2	tiempo	PD2	tiempo	PD2	tiempo	PD2
1	11,8465	0,7312	11,6437	0,7473	10,9205	0,7344	10,649	0,7329	11,1814	0,7498	10,7784	0,7387
2	10,69	0,7316	10,8181	0,746	10,8845	0,7329	10,5553	0,7318	10,7078	0,7453	11,3761	0,7367
3	10,4654	0,7311	10,8369	0,7418	10,8593	0,7273	10,6967	0,73	10,8225	0,7462	11,1713	0,7316
4	10,515	0,7276	10,6583	0,7396	10,889	0,7273	10,9432	0,7278	11,05	0,7427	10,7683	0,7284
5	10,7128	0,7302	10,4796	0,7402	10,7624	0,7273	11,0764	0,7269	10,8517	0,7436	10,8066	0,7293
6	10,7363	0,726	10,5436	0,7349	10,6631	0,7249	10,8296	0,7233	10,9069	0,7404	10,7349	0,726
7	10,5325	0,726	11,0327	0,7371	10,7978	0,7247	10,7349	0,72	10,8606	0,7413	10,9209	0,7269
8	10,6076	0,7253	10,6742	0,7341	10,8801	0,7233	10,8842	0,7209	10,9745	0,7396	10,9933	0,7278
9	11,0591	0,7273	10,8572	0,7356	10,6671	0,724	10,6497	0,7202	10,7689	0,7416	10,8021	0,7282
10	12,2411	0,7187	10,8862	0,7278	10,7487	0,7169	10,8431	0,7153	10,9483	0,7364	10,824	0,7204
11	10,8238	0,7207	10,8826	0,7271	10,9094	0,7153	10,8166	0,7109	10,9936	0,734	10,7754	0,7191

Tabla 4.4 Newpnn con cambio de semilla y cambio de régimen de operación. Valor de dispersión constante 0.026.

Si el valor de la dispersión se incrementa (en este caso a 1) la probabilidad se debe de incrementar. Con un valor de dispersión de 1 la red es capaz de arrojar valores de probabilidad del 75 %, un poco mejores que las anteriores, aunque cabe aclarar que el hecho de se haya ampliado el rango de las clases hace más fácil el proceso de clasificación y por ende la probabilidad de clasificación correcta es mayor. En esta experimentación también se hizo cambio de semillas y de regímenes de operación. Esta red es capaz de clasificar de igual manera, en todos los regímenes y con todos los cambios de semilla, tabla 4.5

Régimen	semilla 1,2		semilla 2,1		semilla 1,3		semilla 2,3		semilla 3,1		semilla 3,2	
	tiempo	PD2	tiempo	PD2	tiempo	PD2	tiempo	PD2	tiempo	PD2	tiempo	PD2
1	11,9435	0,7551	12,0091	0,7727	11,7233	0,7493	11,71	0,7473	11,4082	0,7709	11,5005	0,758
2	11,5463	0,7536	11,1512	0,7702	11,7756	0,7489	12,105	0,7467	11,6695	0,7696	11,7451	0,7567
3	11,6029	0,7542	11,9957	0,7682	11,1706	0,7487	11,6794	0,7473	11,7055	0,7676	11,5028	0,7549
4	11,6389	0,7538	11,147	0,7676	11,1076	0,7406	11,7434	0,7444	11,6443	0,7669	11,6307	0,754
5	11,8691	0,7533	12,2185	0,7687	12,01	0,7473	11,7699	0,7449	11,6136	0,7682	11,4747	0,7533
6	11,8448	0,7507	11,8425	0,7649	11,15	0,7449	11,9356	0,7429	11,5077	0,7656	12,9255	0,7518
7	11,5278	0,7516	11,342	0,7651	10,2055	0,7433	11,4105	0,7429	11,6197	0,7662	12,6973	0,7527
8	11,6886	0,7511	11,8897	0,7662	11,0712	0,7427	12,0385	0,7418	11,4749	0,7658	11,531	0,7527
9	11,6591	0,7527	11,8633	0,7676	11,1748	0,7424	11,7185	0,7427	11,6835	0,7673	11,6362	0,7533
10	11,6355	0,7466	11,4397	0,7636	11,8527	0,738	11,5151	0,7373	11,5889	0,7631	0,7464	0,7467
11	11,1936	0,7478	11,1194	0,7609	12,0494	0,7373	11,5493	0,7364	11,5233	0,7607	11,62213	0,7467

Tabla 4.5 Newpnn con cambio de semilla y cambio de régimen de operación. Valor de dispersión constante 1.

Incrementando el valor de la dispersión (tabla 4.6) disminuye la probabilidad hasta el 60 %, observando los resultados se puede llegar a la conclusión una vez más de que la red es buen clasificador y es estable, aquí aunque los valores de probabilidad aunque bajos también son constantes.

spread 5	semilla 1,2		semilla 2,1		semilla 1,3		semilla 2,3		semilla 3,1		semilla 3,2	
	tiempo	PD2	tiempo	PD2	tiempo	PD2	tiempo	PD2	tiempo	PD2	tiempo	PD2
1	11,1147	0,6342	11,5575	0,6464	11,6586	0,6224	11,3499	0,622	11,3992	0,6447	11,3812	0,6376
2	11,1139	0,6342	11,5616	0,6462	11,8115	0,6229	11,324	0,6224	11,3887	0,6446	11,68454	0,6376
3	1,1244	0,6333	11,4128	0,6424	11,3571	0,6216	11,3331	0,6216	11,2349	0,644	11,7601	0,6351
4	11,2019	0,6329	10,5254	0,6422	11,728	0,6244	11,1902	0,6218	11,375	0,6433	11,6254	0,6327
5	11,0903	0,6342	10,8764	0,6467	11,2364	0,6244	11,5481	0,6236	11,2073	0,646	11,3325	0,6351
6	11,0996	0,6304	11,5663	0,644	11,2808	0,6222	11,4059	0,6207	11,2542	0,6416	11,3278	0,6318
7	11,2624	0,6313	11,6379	0,6442	11,161	0,6236	11,6719	0,622	11,3363	0,6433	11,4102	0,6316
8	11,1029	0,6336	11,5742	0,646	11,3353	0,6247	11,4468	0,6236	11,5898	0,6476	11,4085	0,632
9	11,2518	0,6364	11,3102	0,6516	11,2946	0,6273	11,3492	0,6258	11,3955	0,6518	11,7058	0,6371
10	11,0375	0,6311	11,1891	0,6462	11,28	0,6229	11,4028	0,6207	11,194	0,6467	11,7588	0,6324
11	11,0097	0,6313	10,715	0,6431	11,505	0,6249	11,7906	0,6229	11,4229	0,646	11,6087	0,6311

Tabla 4.6 Newpnn con cambio de semilla y cambio de régimen de operación. Valor de dispersión constante 5.

Como se puede observar, independientemente del valor de dispersión y de las semillas, esta red en general tiene un tiempo de entrenamiento relativamente bajo, pues 12 segundos es incluso un poco menor que la mejor de las redes de retropropagación (trainrp).

4.3.3 Red de regresión generalizada

Esta red de regresión generalizada, al igual que la red tipo probabilística, requiere grandes cantidades de memoria, así que en la primera etapa de experimentación fue necesario determinar cuál es el valor máximo de puntos de operación soportados por el compilador. La tabla 4.7 muestra la comparación con diferente número de puntos de simulación. Como se puede observar la red incrementa la probabilidad con el incremento de puntos de simulación, el tiempo también se ve afectado siendo incrementado de manera proporcional al número de puntos. El compilador fue incapaz de realizar la clasificación con 500 puntos de simulación, el máximo soportado fue de 450, por lo cual, todos los experimentos posteriores fueron realizados con este número de puntos de simulación, mismos con los cuales se alcanza una probabilidad del 73 %, lo suficientemente buena para ser considerable, además el tiempo de entrenamiento con esta red es relativamente bajo.

Semilla 0,1	100 pts		200 puntos		300 Puntos		350 Puntos		450 Puntos	
Régimen	Tiempo	PD2	Tiempo	PD2	Tiempo	PD2	Tiempo	PD2	Tiempo	PD2
1	3,4787	0,6856	7,3862	0,7333	13,8809	0,7237	18,12	0,739	61,1761	0,7432
2	3,462	0,6833	7,4132	0,7294	13,8007	0,7193	17,7236	0,7362	46,0648	0,739
3	3,3522	0,6889	7,4019	0,7289	13,9196	0,71	18,1549	0,7317	49,5452	0,7358
4	3,4157	0,6889	7,37	0,7306	14,0136	0,71	18,3282	0,7302	61,6775	0,7363
5	3,4019	0,6856	7,2876	0,7294	13,783	0,7126	18,2767	0,7308	61,2707	0,7378
6	3,4357	0,6867	7,3789	0,7289	13,9042	0,7093	18,1905	0,7308	47,303	0,7323
7	3,443	0,6856	7,343	0,73	13,8717	0,7089	18,2442	0,7298	49,9093	0,7321
8	3,3936	0,6822	7,284	0,7294	13,9428	0,7107	18,2199	0,7311	35,9907	0,7326
9	3,2803	0,6856	7,3647	0,7311	13,8743	0,7052	18,2846	0,7321	25,7156	0,7341
10	3,4325	0,6856	7,402	0,7272	14,1046	0,7	18,3216	0,7286	49,2592	0,7286
11	3,4538	0,6822	7,4009	0,7228	13,986	0,7022	18,375	0,7248	30,5932	0,7257

Tabla 4.7 Newgrnn con variación en el número de puntos de simulación y régimen de operación, numero llave 0,1, dispersión 0.026.

Observando la tabla anterior es posible darse cuenta que las probabilidades con 350 puntos son parecidas a las de 450 puntos, la única diferencia es el tiempo de entrenamiento: entre más puntos de simulación, mayor es el tiempo de entrenamiento. El siguiente análisis hecho fue manteniendo 350 puntos de simulación, manteniendo el valor de la dispersión constante y haciendo el cambio de semilla, con ello fue posible darse cuenta que este cambio no es determinante en el comportamiento de la red. La red sigue dando valores de probabilidad cercanos al 75% y tiempos de operación alrededor de los 20 segundos.

350 Puntos, spread 0.026										
	semilla 01		semilla 12		semilla 21		semilla 31		semilla 32	
Régimen	Tiempo (s)	PD2	Tiempo (s)	PD2	Tiempo (s)	PD2	Tiempo (s)	PD2	Tiempo (s)	PD2
1	18,12	0,739	18,111	0,741	18,2097	0,7483	18,2172	0,7517	18,5933	0,7492
2	17,7236	0,7362	18,2938	0,7359	18,5967	0,7435	18,3993	0,7486	18,8031	0,7438
3	18,1549	0,7317	18,2881	0,7327	18,1081	0,7432	18,248	0,7406	19,4297	0,741
4	18,3282	0,7302	18,4115	0,7333	18,2931	0,7425	18,4487	0,7425	18,5787	0,7413
5	18,2767	0,7308	18,3395	0,7317	18,2321	0,7419	18,4434	0,7444	18,8455	0,7438
6	18,1905	0,7308	18,3654	0,7276	18,2434	0,739	18,4254	0,7406	18,8407	0,7371
7	18,2442	0,7298	18,145	0,7276	18,579	0,7371	18,6444	0,74	18,5155	0,7387
8	18,2199	0,7311	18,4344	0,7305	18,2907	0,7371	18,4156	0,7403	18,4291	0,7352
9	18,2846	0,7321	18,1785	0,7267	18,3467	0,7371	18,4932	0,7419	18,2755	0,7378
10	18,3216	0,7286	18,315	0,7194	18,6105	0,7343	18,4603	0,7384	18,4945	0,7267
11	18,375	0,7248	18,3002	0,7197	18,4881	0,734	18,7264	0,7343	18,4004	0,7267

Tabla 4.8 Newgrnn con cambio de semilla y cambio de régimen de operación. Valor de dispersión constante 0.026 y 350 puntos de simulación.

El mismo análisis anterior fue hecho pero con 450 puntos de simulación, la probabilidad incrementa aproximadamente un 2 %, pero el tiempo incrementa de 20 a 55 segundos (tabla 4.9). La red mostro ser estable, a pesar del cambio de régimen de operación y cambio de semilla, la matriz muestra un comportamiento estable.

450 Puntos, spread 0.026										
	semilla 01		semilla 12		semilla 21		semilla 31		semilla 01	
Régimen	Tiempo	PD2	Tiempo	PD2	Tiempo	PD2	Tiempo	PD2	Tiempo	PD2
1	61,1761	0,7432	53,1183	0,7348	53,7176	0,7541	33,9258	0,7462	44,7479	0,7373
2	46,0648	0,739	79,8544	0,7319	41,2561	0,7511	37,1206	0,7447	33,7699	0,7323
3	49,5452	0,7358	49,9261	0,7279	29,7512	0,7422	29,3706	0,7415	46,4225	0,7279
4	61,6775	0,7363	47,077	0,7259	32,6582	0,74	56,104	0,7407	31,3833	0,7289
5	61,2707	0,7378	126,7171	0,7257	34,7604	0,7402	46,0183	0,7427	41,3248	0,7299
6	47,303	0,7323	56,384	0,7225	25,1282	0,7375	39,5296	0,7388	49,8648	0,7257
7	49,9093	0,7321	62,8276	0,7225	32,0667	0,737	55,2873	0,739	43,5724	0,7259
8	35,9907	0,7326	64,2082	0,7212	37,7736	0,7388	33,997	0,7385	34,7818	0,7262
9	25,7156	0,7341	47,8297	0,7217	30,3167	0,736	35,0194	0,7388	57,553	0,7284
10	49,2592	0,7286	47,3287	0,717	63,005	0,7316	57,1206	0,7304	55,2597	0,724
11	30,5932	0,7257	41,7122	0,7138	45,4154	0,7299	42,9216	0,7284	42,1161	0,7198

Tabla 4.9 Newgrnn con cambio de semilla y cambio de régimen de operación. Valor de dispersión constante 0.026 y 450 puntos de simulación

Al igual que con la red de retropropagación, se hizo el análisis incrementando el valor de la dispersión a un valor de 1, con lo cual la red da valores de probabilidad parecidos a las

redes de retropropagación. Incrementa el valor de la probabilidad hasta 77 % (solo en algunos casos), el tiempo de entrenamiento ronda el minuto (tabla 4.10).

450 Puntos, spread 1										
Régimen	semilla 01		semilla 12		semilla 21		semilla 31		semilla 32	
	Tiempo	PD2	Tiempo	PD2	Tiempo	PD2	Tiempo	PD2	Tiempo	PD2
1	65,166	0,7689	59,0548	0,7595	33,2525	0,7728	41,935	0,776	23,4506	0,7637
2	37,1517	0,7677	61,6509	0,7575	63,3585	0,7701	33,4224	0,7738	34,3354	0,761
3	57,4673	0,7669	55,4378	0,7565	42,1495	0,7679	27,8734	0,7709	40,6192	0,7598
4	32,4478	0,7662	43,0012	0,7563	53,9362	0,7667	58,0237	0,7709	42,4366	0,7595
5	32,4645	0,7659	65,0911	0,7541	40,0086	0,7686	64,5969	0,7726	50,0397	0,7602
6	30,4338	0,7612	46,6225	0,7519	47,4909	0,764	51,2447	0,7681	34,351	0,7568
7	37,8703	0,7607	68,2375	0,7514	38,9032	0,7642	38,4449	0,7681	49,7398	0,7565
8	36,2208	0,7612	57,4784	0,7521	31,0906	0,7644	43,402	0,7681	35,9523	0,7573
9	38,8759	0,7635	50,9287	0,7536	39,9514	0,7659	43,2928	0,7694	40,669	0,7565
10	36,546	0,758	39,1711	0,7469	29,8302	0,7602	42,1351	0,7654	36,425	0,7523
11	49,123	0,7546	49,471	0,7464	30,0975	0,7573	38,8879	0,7637	37,0266	0,7504

Tabla 4.10 Newgrnn con cambio de semilla y cambio de régimen de operación. Valor de dispersión constante 1 y 450 puntos de simulación.

Por último se analizó el comportamiento con un valor de dispersión de 5 (tabla 4.10); al igual que las redes probabilísticas, disminuye la probabilidad en comparación con los valores de la tabla anterior.

450 Puntos spread 5										
Régimen	semilla 01		semilla 12		semilla 21		semilla 31		semilla 32	
	Tiempo	PD2	Tiempo	PD2	Tiempo	PD2	Tiempo	PD2	Tiempo	PD2
1	72,8239	0,6417	59,0548	0,7595	33,2525	0,7728	41,935	0,776	23,4506	0,7637
2	68,1892	0,6427	61,6509	0,7575	63,3585	0,7701	33,4224	0,7738	34,3354	0,761
3	49,0879	0,639	55,4378	0,7565	42,1495	0,7679	27,8734	0,7709	40,6192	0,7598
4	52,5562	0,6385	43,0012	0,7563	53,9362	0,7667	58,0237	0,7709	42,4366	0,7595
5	34,1864	0,6415	65,0911	0,7541	40,0086	0,7686	64,5969	0,7726	50,0397	0,7602
6	27,7976	0,6385	46,6225	0,7519	47,4909	0,764	51,2447	0,7681	34,351	0,7568
7	30,5653	0,6378	68,2375	0,7514	38,9032	0,7642	38,4449	0,7681	49,7398	0,7565
8	34,6803	0,64	57,4784	0,7521	31,0906	0,7644	43,402	0,7681	35,9523	0,7573
9	23,4769	0,6447	50,9287	0,7536	39,9514	0,7659	43,2928	0,7694	40,669	0,7565
10	37,8054	0,6427	39,1711	0,7469	29,8302	0,7602	42,1351	0,7654	36,425	0,7523
11	23,1841	0,6398	49,471	0,7464	30,0975	0,7573	38,8879	0,7637	37,0266	0,7504

Tabla 4.11 Newgrnn con cambio de semilla y cambio de régimen de operación. Valor de dispersión constante 5 y 450 puntos de simulación

4.4 Comparación y resultados de las mejores redes

Para la comparación de las redes con mejores resultados, se hicieron dos análisis. El primero de ellos, el análisis para la identificación de clases simples y el segundo para la identificación de clases múltiples. Al referir clases múltiples, se plantea el desarrollo de más de una falla en paralelo.

La tabla 4.12 muestra los resultados obtenidos de los cálculos para clasificación de fallas simples. En la tabla se puede observar que la función trainrp de retropropagación, así como la función de base radial newrb, tienen valores de probabilidad más altos y parecidos entre ellas, sin embargo el tiempo de cálculo es mayor en el caso de la función newrb. Las funciones tanto probabilísticas como de regresión mostraron tener una probabilidad de diagnóstico parecida, del 75 %. Otra característica interesante es que las probabilidades se mantienen constantes con el cambio de semilla, así como con el cambio de régimen de operación.

Reg.	Variables Metodo	semilla 1,2		semilla 2,1		semilla 1,3		semilla 2,3		semilla 3,1		semilla 3,2	
		T	PD2	T	PD2	T	PD2	T	PD2	T	PD2	T	PD2
2	trainrp	7,1138	0,8104	8,082	0,8169	7,8127	0,8047	7,8064	0,8085	7,7561	0,8162	7,7456	0,8102
	newrb	121,68	0,798	120,463	0,7969	119,886	0,7924	124,593	0,7927	120,167	0,8071	135,715	0,7993
	newpnn	11,5463	0,7536	11,1512	0,7702	11,7756	0,7489	12,105	0,7467	11,6695	0,7696	11,7451	0,7567
	newgrnn	75,4349	0,7575	70,7678	0,7701	57,3249	0,7494	42,3877	0,7472	50,0758	0,7738	38,4773	0,761
4	trainrp	6,8999	0,8082	7,9393	0,8113	7,6205	0,7969	7,8284	0,802	7,784	0,8149	7,6476	0,808
	newrb	123,566	0,7942	127,187	0,7938	119,964	0,7893	119,169	0,7891	118,342	0,8064	121,821	0,7989
	newpnn	11,6389	0,7538	11,147	0,7676	11,1076	0,7406	11,7434	0,7444	11,6443	0,7669	11,6307	0,754
	newgrnn	73,9894	0,7563	63,0691	0,7667	79,8563	0,7474	48,094	0,743	48,2959	0,7709	37,807	0,7595
8	trainrp	7,4105	0,8062	7,9187	0,8127	7,6653	0,7964	7,5294	0,7996	7,6556	0,81	7,8712	0,8073
	newrb	121,228	0,7927	120,073	0,7989	120,198	0,7909	119,262	0,7873	119,636	0,806	121,087	0,7942
	newpnn	11,6886	0,7511	11,8897	0,7662	11,0712	0,7427	12,0385	0,7418	11,4749	0,7658	11,531	0,7527
	newgrnn	60,3891	0,7521	50,6365	0,7644	59,9075	0,7425	51,1254	0,7405	46,4046	0,7681	44,6493	0,7573
11	trainrp	7,9275	0,8011	7,8748	0,8029	7,7685	0,7878	7,5564	0,7913	7,8688	0,8071	7,6736	0,8011
	newrb	120,775	0,7902	121,914	0,7909	120,401	0,7871	123,021	0,7833	121,29	0,8002	121,227	0,7898
	newpnn	11,1936	0,7478	11,1194	0,7609	12,0494	0,7373	11,5493	0,7364	11,5233	0,7607	11,62213	0,7467
	newgrnn	72,7591	0,7464	61,7146	0,7573	43,2649	0,7378	62,7971	0,7343	53,858	0,7637	42,1649	0,7504

Tabla 4.12 Comparación de redes en clasificación con clases simples, con 500 puntos de operación.

Si comparamos sólo las probabilidades, se puede observar que la red entrenada con la función trainrp, es la que mejor desempeño tiene y que la función newrb tiene probabilidades parecidas. La figura 4.7 muestra esta comparación para una sola combinación de semillas.

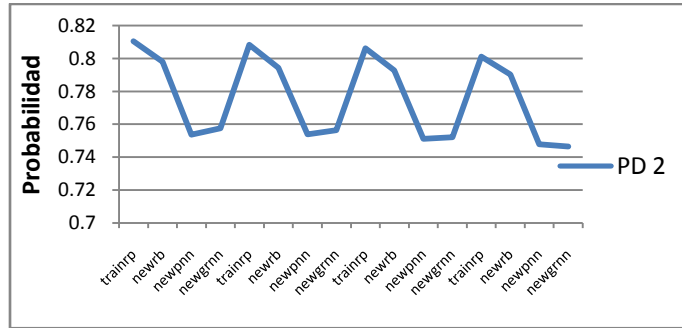


Figura 4.7 Probabilidad en distintos regímenes y con cambio de números semilla, para clases simples.

Por otro lado, comparando el tiempo, es la función probabilística la que tiene resultados comparables con la función trainrp. En este caso la cúspide la tiene el entrenamiento con la función trainrp, en donde el tiempo de entrenamiento es de aproximadamente 120 segundos. La figura 4.8 muestra el tiempo de cálculo para cada régimen de operación en una sola combinación de semillas.

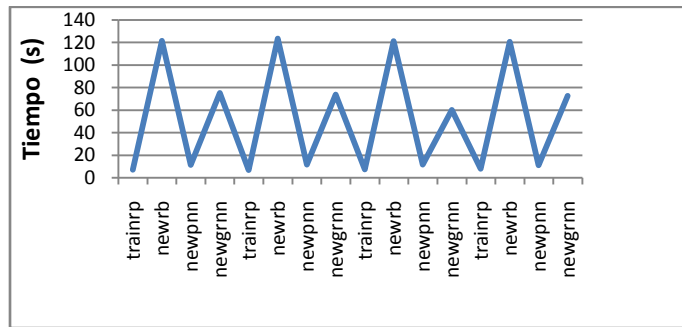


Figura 4.8 Tiempo de cálculo para una sola combinación de semillas en diferentes regímenes de operación. Clases simples.

De lo anterior se puede concluir que es la red tipo retropropagación, entrenada con la función resiliente, la que presenta mejores características, tanto en tiempo de cálculo como en probabilidad de diagnóstico.

La tabla 4.13, muestra los mismos cálculos, pero en este caso para clasificación de falla múltiple, las probabilidades de diagnóstico incrementan, lo cual es lógico pensar si se considera que la clasificación considera la intersección entre las clases. En este caso, la red da probabilidades de aserción de alrededor del 85 %, siendo en este caso las redes de base radial las que tienen mayor probabilidad.

Reg.	Variables Método	semilla 1,2		semilla 2,1		semilla 1,3		semilla 2,3		semilla 3,1		semilla 3,2	
		T	PD2	T	PD2	T	PD2	T	PD2	T	PD2	T	PD2
2	trainrp	3,8668	0,856	3,8106	0,8665	3,8441	0,865	3,7963	0,865	3,8611	0,8645	3,8471	0,864
	newrb	6,4794	0,87	6,3501	0,8605	6,4153	0,865	6,4183	0,8585	6,3898	0,8625	6,3916	0,8675
	newpnn	2,937	0,839	2,9368	0,823	2,9059	0,823	2,8899	0,825	2,8947	0,833	2,9426	0,847
	newgrnn	3,1017	0,839	3,1116	0,823	3,047	0,823	3,1341	0,825	3,1701	0,833	3,1365	0,847
4	trainrp	3,8867	0,8635	3,8078	0,8665	3,8673	0,868	3,7731	0,8695	3,8767	0,8715	3,8221	0,864
	newrb	6,6373	0,867	6,606	0,864	6,3535	0,862	6,3557	0,853	6,4037	0,8595	6,4032	0,86
	newpnn	2,9454	0,834	2,9314	0,8255	2,8925	0,821	2,909	0,826	2,9194	0,836	2,9198	0,849
	newgrnn	3,1218	0,834	3,1077	0,8255	3,0707	0,821	3,1349	0,826	3,1591	0,836	3,1585	0,849
8	trainrp	3,8568	0,8605	3,8198	0,8635	3,779	0,865	3,8032	0,865	3,8384	0,8585	3,8463	0,8625
	newrb	6,6477	0,8685	6,2963	0,8585	6,3888	0,866	6,4065	0,8575	6,4013	0,854	6,3883	0,858
	newpnn	2,9636	0,8295	2,9422	0,8215	2,8911	0,8185	2,8724	0,82	2,9887	0,8295	2,9905	0,8405
	newgrnn	3,061	0,8295	3,127	0,8215	3,0563	0,8185	3,1937	0,82	3,128	0,8295	3,1446	0,8405
11	trainrp	3,8516	0,8565	3,8651	0,86	3,8131	0,8625	3,8074	0,8615	3,8229	0,863	3,8351	0,855
	newrb	6,4157	0,8665	6,3491	0,848	6,4132	0,8605	6,4033	0,8505	6,3974	0,8505	6,4027	0,857
	newpnn	2,9604	0,82	2,9236	0,8165	2,9618	0,81	2,8478	0,817	2,978	0,825	2,9307	0,8355
	newgrnn	3,0905	0,82	3,1413	0,8165	3,1378	0,81	3,036	0,817	3,1615	0,825	3,2022	0,8355

Tabla 4.13 Comparación de redes en clasificación con clases múltiples, con 500 puntos de operación.

En este caso es la red radial (newrb) la que tiene mayor índice de confianza, sin embargo la red trainrp, alrededor del 82 %. Como se puede ver, estos valores de probabilidad se mantienen constantes, una vez más, con el cambio de régimen. Nota: esta gráfica corresponde a un solo valor de semilla, para el resto se observaron resultados similares.

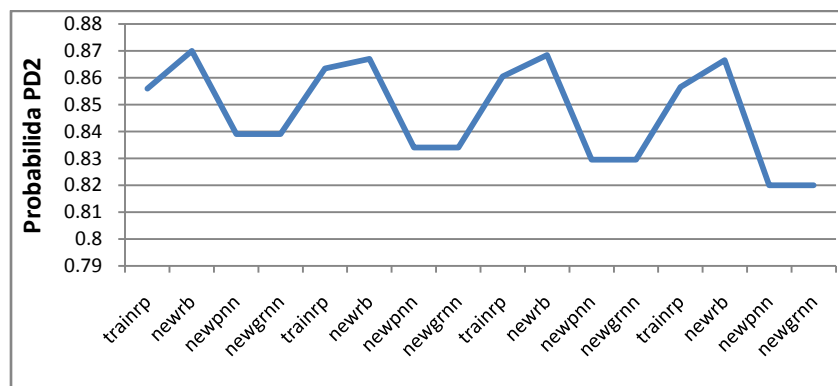


Figura 4.9 Probabilidad en distintos regímenes y sin cambio de números semilla para clases múltiples.

En este caso es la red probabilística la que toma menor tiempo para el cálculo, y la red de base radial la que toma el mayor tiempo de cálculo, como en la previa gráfica.

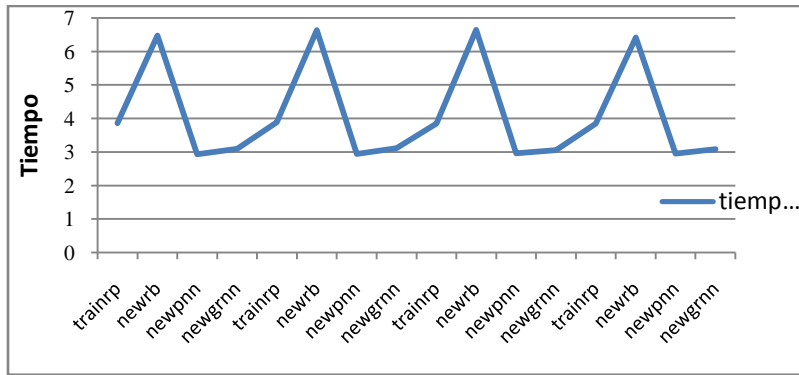


Figura 4.10 Tiempo de cálculo para una sola combinación de semillas en diferentes regímenes de operación. Clase múltiple.

Conclusiones

Se analizaron en total 15 métodos de clasificación aplicados al diagnóstico paramétrico. De las 12 funciones de entrenamiento de retropropagación, fueron sólo 2 las que mostraron tiempos de entrenamiento menor a dos minutos, dando con ello una probabilidad del 80 %. Estas redes mostraron comportamiento estable ocupando hasta 2000 puntos de simulación, lo cual es benéfico porque pueden ser aplicadas en un rango amplio de datos. Los tres métodos restantes basados en redes neuronales del tipo radial, mostraron ser unos excelentes clasificadores con tiempo de operación entre 10 y 60 segundos, aunque se sacrifica un poco la probabilidad, entre 71 y 76 %, más sin embargo los resultados son aceptables para un diagnóstico correcto. La última fase de experimentación muestra que la red de retropropagación entrenada con la función trainrp es la que tiene mejores resultados en todas las condiciones: es estable, tienen un tiempo de entrenamiento de solo algunos segundos y no es costosa computacionalmente hablando.

CONCLUSIONES GENERALES

En general, para las redes que contienen hasta unos cien pesos de conexión, el algoritmo de Levenberg-Marquardt tendrá la convergencia más rápida. En muchos casos, la *trainlm* es capaz de obtener ECM inferiores que cualquiera de los otros algoritmos probados. Sin embargo, a medida que aumenta el número de pesos de la red, disminuye la ventaja de *trainlm*. Además, el rendimiento de *trainlm* es relativamente pobre en problemas de reconocimiento de patrones.

La función de *trainrp* es el algoritmo más rápido. Su rendimiento también se degrada cuando se reduce la meta de error. Los requisitos de memoria para este algoritmo son relativamente pequeños en comparación con los otros algoritmos considerados. Los algoritmos de gradiente conjugados CGS son casi tan rápidos como el algoritmo de LM y casi tan rápidos como *trainrp*. Su rendimiento no se degrada rápidamente igual rendimiento de *trainrp* cuando el error se reduce. Los algoritmos de gradiente conjugado tienen requisitos de memoria relativamente modestos. El rendimiento de *trainbfg* es similar a la de *trainlm*, pero el tiempo de cómputo necesario aumenta debido al cálculo de la matriz inversa en cada iteración. La variable *traingdx* es generalmente mucho más lenta que los otros métodos y tiene los mismos requisitos de almacenamiento de información que *trainrp*.

Los algoritmos de redes radiales mostraron un buen desempeño cuando trabajan con un volumen de datos pequeño, al incrementar el volumen de datos se incrementa la cantidad de memoria necesaria y por ende no se puede trabajar con grandes volúmenes de datos. La red radial de diseño exacto es inútil en este caso, ya que al tener un valor grande en los puntos de simulación, la red exige una matriz de igual tamaño, lo cual implica grandes volúmenes de datos imposibles de procesar en una computadora común (2Gb RAM, 1.8 MHz).

Si la red radial es limitada en el número de neuronas como en el caso de la red eficiente, se demostró que con un máximo de 20 neuronas se pueden lograr aproximaciones casi iguales a las de redes de retropropagación. Las redes probabilísticas trabajan de igual manera que las redes de regresión generalizada, éstas solo pueden trabajar con un máximo de 500 puntos de simulación con una computadora de 2 Gb en RAM y una velocidad de procesador de 1.8 MHz. La principal limitación es la memoria RAM.

Las probabilidades de diagnóstico son consideradas como buenas arriba del 75 %, con ello se asegura tener un buen reconocimiento de fallas, sin problemas de sobre entrenamiento y sin problemas por la falta de él. Los tiempos de ejecución en los métodos que requieren mayor tiempo de entrenamiento llegan a alcanzar hasta algunas horas, más sin embargo, en el presente trabajo no fueron considerados; se describieron los métodos con tiempos de entrenamiento menor a 20 minutos. Y para comparaciones, sólo de los que tiene tiempos menores a 2.5 minutos fueron expuestos.

De los resultados se puede decir que las redes *trainrp*, radial de diseño más eficiente, probabilísticas son capaces de arrojar valores de probabilidad y tiempo parecidos. Si se considera utilizar gran cantidad de puntos de entrenamiento, la función *trainrp* es la mejor opción, referente a probabilidad y tiempo, más sin embargo si se tiene relativamente pocos

puntos de simulación y se quiere hacer el proceso en forma rápida, las redes radiales también son opción.

RECOMENDACIONES

Otros algoritmos como los de SVM están ganando terreno hoy en día, sería interesante aplicar éstos a diagnóstico paramétrico y comparar con los experimentos realizados. Estos algoritmos están diseñados para trabajar bajo diferentes ambientes, así que se recomienda hacerlos trabajar en Matlab, para tener un punto de comparación.

Con los experimentos mostrados es posible aplicar los algoritmos con mejor desempeño a sistemas de control que puedan reaccionar en forma rápida y eficiente para diagnosticar un motor de turbina de gas. Así pues, se recomienda trabajar en un sistema de control automático que utilice estos algoritmos y que sea capaz de dar un pronóstico en base al monitoreo y diagnóstico hecho por el mismo.

Todos los experimentos fueron desarrollados con un solo motor que aunque considera diferentes condiciones de operación se recomienda hacer los mismos experimentos con otro motor.

BIBLIOGRAFÍA

- [1] Avilan Ivan, *Historia de la turbina de gas*, disponible en: <http://www.monografias.com/trabajos/turbinagas/turbinagas.shtml>, consultado en septiembre 2008.
- [2] Herb Sarvanamutto, *Gas turbine theorie*, 5ª edition
- [3] Sophie Germain & Francis Stein, *Turbinas de gas*, disponible en: <http://ingenieriayciencia.blogspot.com/2007/02/turbinas-de-gas.html>, publicado 20 febrero 2007, consultado octubre 2008.
- [4] William W. Bathie, *Fundamentals of gas turbines*, 2 edition (November 29, 1995)
- [5] UPC, *Turbinas de gas*, disponible en: <http://www.edicionsupc.es/ftppublic/pdfmostra/EM04001M.pdf>, 2002, consultado en octubre 2008.
- [6] Y. A. Çengel and M.A. Boles, *Thermodynamics: an engineering approach*, 5th edicion, Mc Graw Hill, 2006.
- [7] Igor Loboda, IPN, México, Confiabilidad y diagnóstico paramétrico, cátedras (2008).
- [8] Michael Romen and Gregory J Kacprzyński; *Advanced Diagnosis and Pronostic for gas turbine engine risk assessment*; Impact technologies, LLC, Turbo Expo Munich Germany, Mayo 2000.
- [9] Igor Loboda, *An overview of gas turbine health monitoring methods*, Congreso y exposición latinoamericana de turbomaquinaria 2008.
- [10] Sergios Theodoquidis, Konstantinos Kourtroumbus, *Pattern Recognition*, Ed. Academic Press 2º Edición 2003.
- [11] Duda, Richard O., Hart, Peter E., Stork, David G., *Pattern Classification*. John Wiley & Sons, Toronto. 2001.
- [12] Rafael C. Gonzalez, Richard E. Woods, *Procesamiento digital de imágenes*, 2005.
- [13] Rudjer Boskovic Institute, http://dms.irb.hr/tutorial/tut_applic_ref.php, *Desition trees*, 2009.
- [14] Francisco José Cortijo Bon. Técnicas supervisadas II: Aproximación paramétrica, Octubre del 2001. Consultado en: http://iie.fing.edu.uy/ense/assign/recpat/material/tema3_00-01/tema3_00-01_www.html
- [15] Universidad la Salle, Laboratorio de cómputo, <http://www.lci.ulsal.mx/Material/pdf/Reconocimiento%20de%20Patrones.pdf>, 2004.

- [16] Hertz, Krogh y Palmer Introduction to the theory of neural computation - Addison-Wesley, 2007.
- [17] Baéz Cervantes Alida Elizabeth, tesis “Identificación de firmas por medio de redes neuronales”, IPN, 2006.
- [18] Universidad tecnológica de Pereira, www.ohm.utp.edu.co/neuronales, “Manual de redes neuronales”, 2000.
- [19] Howard Demuth, Mark Beale, Martin Hagan, Neural Network Toolbox 6, 2008
- [20] Bonifacio Martín del Río, Alfredo Sanz Molina, “Redes Neuronales y Sistemas Difusos”. Segunda edición, Ed. Alfaomega. 2004.
- [21] Enrique Castillo, Ángel Cobo, José Manuel Gutiérrez, Rosa Eva Pruneda, “Introducción a las Redes Funcionales con Aplicaciones”. Ed. Paraninfo, 2002.
- [22] Gutiérrez Flores Víctor Hugo, tesis “Desarrollo y análisis de los algoritmos del diagnóstico de un motor turboeje”, 2008.

APENDICE A: ALGORITMOS

**PRINCIPAL.

Este es un programa que permite comparar el comportamiento de diferentes redes neuronales con aplicación al diagnóstico paramétrico de turbinas de gas.

El programa se abre con el programa llamado **principal**, en la primera ventana se observarán los valores que se pueden cambiar para realizar las diferentes pruebas. Los datos fueron obtenidos de una turbina de gas en 11 diferentes regímenes de operación, tomando un total de 21 puntos de operación, alrededor de los cuales dándoles una dispersión normal, se obtiene el total de puntos de operación (también son dados por el usuario). Después de que se han dado el total de los datos necesarios, se procede a elegir el tipo de entrenamiento, se puede elegir entre 12 diferentes tipos, una vez que se escoge el método, el programa comienza a calcular; una vez que termina el entrenamiento, se grafican las clases del lado derecho, y en la ventana sobrepuesta se observa el resultado del entrenamiento. También se puede elegir una red del tipo radial, en cuyo caso solo se tiene que elegir el número de neuronas que se quiere que tenga el proceso y de igual forma comienza a realizar los cálculos; en la parte inferior de la ventana, aparecerá el tiempo total que se llevó el entrenamiento, así como la probabilidad de diagnóstico correcto de la red.

```
***Principal****
clc;
clear all;
close all;
h_fig=figure(1);
set(h_fig,'Position',[20,70,1200,650],...
'Name', 'ING EULALIO TORRES GARCIA',
'NumberTitle','off', 'MenuBar','none');
BC=get(h_fig,'Color');
%%encabezado
h_frame= uicontrol(h_fig,'style','frame',...
'Units','normalized',...
'Interruptible','off',...
'Position',[0.20,0.94,0.55,0.06],...
'BackgroundColor',[0.1,0.1,0.1]);
uicontrol(h_fig,'style','text',...
'Units','normalized',...
'Position',[0.25,0.95,0.45,0.04],...
'ForegroundColor',[0,0,0],...
'FontWeight','bold',...
'FontSize',12,...
'String','DIAGNÓSTICO PARAMETRICO DE
TURBINAS DE GAS POR RNA');
%%puntos de simulacion
h_frame= uicontrol(h_fig,'style','frame',...
'Units','normalized',...
'Interruptible','off',...
'Position',[0,0.84,1,0.1],...
'BackgroundColor',[.5,0.5,0.3]);
h_frame= uicontrol(h_fig,'style','text',...
'Units','normalized',...
'Position',[0.01,0.9,0.165,0.02],...
'ForegroundColor',[0,0,1],...
'FontWeight','bold',...
'String','PUNTOS A SIMULAR');
h_pop1= uicontrol(h_fig,'style','popupmenu',...
'Units','normalized',...
'Position',[0.065,0.85,0.04,0.05],...
'String','10|20|100|350|500|1000|1500|2000',...
'BackgroundColor',[0.5,0.5,0.5],...
'ForegroundColor',[1,1,1],...
'FontWeight','bold',...
'Callback','DostNeuro02');
%%regimen de operacion
h_frame= uicontrol(h_fig,'style','text',...
'Units','normalized',...
'Position',[0.2,0.9,0.055,0.02],...
'ForegroundColor',[0,0,1],...
'FontWeight','bold',...
'String','REGIMEN');
h_pop2= uicontrol(h_fig,'style','popupmenu',...
'Units','normalized',...
'Position',[0.205,0.85,0.04,0.05],...
'String','1|2|3|4|5|6|7|8|9|10|11',...
'BackgroundColor',[0.5,0.5,0.5],...
'ForegroundColor',[1,1,1],...
'FontWeight','bold',...
'Callback','DostNeuro02');
%%semilla 1
h_frame= uicontrol(h_fig,'style','text',...
'Units','normalized',...
'Position',[0.3,0.9,0.055,0.02],...
'ForegroundColor',[0,0,1],...
'FontWeight','bold',...
'String','SEMILLA 1');
h_pop3= uicontrol(h_fig,'style','popupmenu',...
'Units','normalized',...
'Position',[0.31,0.84,0.035,0.06],...
'String','1|2|3',...
'BackgroundColor',[0.5,0.5,0.5],...
```

```

    'ForegroundColor',[1,1,1],...
    'FontWeight','bold',...
    'Callback','DostNeuro02');
%%% semilla 2
h_frame= uicontrol(h_fig,'style','text',...
    'Units','normalized',...
    'Position',[0.4,0.9,0.055,0.02],...
    'ForegroundColor',[0,0,1],...
    'FontWeight','bold',...
    'String','SEMILLA 2');
h_pop4= uicontrol(h_fig,'style','popupmenu',...
    'Units','normalized',...
    'Position',[0.41,0.84,0.035,0.06],...
    'String','1|2|3',...
    'BackgroundColor',[0.5,0.5,0.5],...
    'ForegroundColor',[1,1,1],...
    'FontWeight','bold',...
    'Callback','DostNeuro02');
%%% tipo de red a entrenar%%%
h_push1=uicontrol(h_fig,'style','pushbutton',...
    'Units','normalized',...
    'Position',[0.5,0.86,0.09,0.06],...
    'String','Retropropagacion',...
    'FontAngle','Italic',...
    'Callback','inicio3');
h_push2=uicontrol('parent',h_fig,'style','pushbutton',...
    'Units','normalized',...
    'Position',[0.7,0.86,0.09,0.06],...
    'String','Red-radial',...
    'FontAngle','italic',...
    'Callback','inicio4');
Handles.pop1=h_pop1;
Handles.pop2=h_pop2;
Handles.pop3=h_pop3;
Handles.pop4=h_pop4;
Handles.push1=h_push1;
Handles.push2=h_push2;
set(h_fig,'UserData',Handles);
*** Dots Neuro02***
h_fig=gcbf;
H=get(h_fig,'UserData'); %datos guardados en figura
h_pop1=H.pop1;
h_pop2=H.pop2;
h_pop3=H.pop3;
h_pop4=H.pop4;
%%% puntos de simulacion
puntos_select=get(h_pop1,'Value');
switch puntos_select
    case 1
        ktd=10;
    case 2
        ktd=20;
    case 3
        ktd=100;
    case 4
        ktd=350;
    case 5
        ktd=500;
    case 6
        ktd=1000;
    case 7
        ktd=1500;
    case 8
        ktd=2000;
end
% Cantidad de puntos a similar por clase
krg=11; % cantidad de regimenes.
% regimen a analizar
regimen_select=get(h_pop2,'Value');
switch regimen_select
    case 1
        nreg=1;
    case 2
        nreg=2;
    case 3
        nreg=3;
    case 4
        nreg=4;
    case 5
        nreg=5;
    case 6
        nreg=6;
    case 7
        nreg=7;
    case 8
        nreg=8;
    case 9
        nreg=9;
    case 10
        nreg=10;
    case 11
        nreg=11;
    case 12
        nreg=12;
end
kdf=9; % Cantidad de clases
kdl=21; % Cantidad de puntos de interpolación para el
desarrollo de cada clase.
kdf0=9; % Número máximo de clases
kdf1=9; % Número de clases simples
idef2=0; kdf2=4; % clases multiples
kpr0=6; % Cantida de parmetros
%%% iniciacion de generadores aleatorios
semilla1=get(h_pop3,'Value');
switch semilla1
    case 1,
        nra=1;
    case 2
        nra=2;
    case 3
        nra=3;
end
nr1=nra;
semilla2=get(h_pop4,'Value');
switch semilla2
    case 1,
        nrb=1;
    case 2
        nrb=2;
    case 3
        nrb=3;
end
nr2=nrb;
%

```



```

DY=[0.015 0.015 0.025 0.015 0.020 0.020]; %
parametros de error máximo
krdd=krp*kdf0*kdl;
i_EarlyStop=0; % Opcion Early Stopping
% EJEMPLOS DE ENTRENAMIENTO
% Lectura del archivo
fid=fopen('dpar.dat','rt');
BT=fscanf(fid,'%g',[kpr0,krdd]);
BB=BT;
for ipr=1:kpr0
    BB(:,ipr)=BB(:,ipr)/DY(ipr);
end;
% Formación del arreglo multidimensional
for irg=1:krp
    for idf=1:kdf0
        for idl=1:kdl
            irdd=(irg-1)*kdf0*kdl+(idf-1)*kdl+idl;
            B(:,idl,idf,irg)=BB(irdd,:);
        end;
    end;
end;
% Determinación del regimen de operación
D=B(:,:,:,nreg);
% Ejemplos de entrenamiento
ZE=ones(kpr0,1);
ZD_1=ones(kpr0,1);
ZD_2=ones(kpr0,1);
rand('state',nr1); randn('state',nr1); % Numeros
llave de los generadores aleatorios
if idef2==0
    kdf=kdf1;
else
    kdf=kdf2;
end;
for idf=1:kdf
    for itd=1:ktd
        ZE=randn(size(ZE))/3.0;
        dzn=randn/3.0;
        if idef2==0
            ld=rand*(kdl-1); ii=floor(ld)+1; alfa=ld-ii+1;
            ZD1(:,itd,idf)=D(:,ii,idf)*(1-
alfa)+D(:,ii+1,idf)*alfa+ZE(:);
        else
            idf1=2*idf-1; idf2=2*idf;
            ld1=rand*(kdl-1); ld2=rand*(kdl-1);
            if (ld1+ld2)>(kdl-1)
                ld1=(kdl-1)-ld1; ld2=(kdl-1)-ld2;
            end;
            ii1=floor(ld1)+1; alfa1=ld1-ii1+1;
            ii2=floor(ld2)+1; alfa2=ld2-ii2+1;
            ZD_1(:)=D(:,ii1,idf1)*(1-
alfa1)+D(:,ii1+1,idf1)*alfa1;
            ZD_2(:)=D(:,ii2,idf2)*(1-
alfa2)+D(:,ii2+1,idf2)*alfa2;
            ZD1(:,itd,idf)=ZD_1(:)+ZD_2(:)+ZE(:);
        end
    end
end
rand('state',nr2); randn('state',nr2);
for idf=1:kdf
    for itd=1:ktd
        ZE=randn(size(ZE))/3.0;
        dzn=randn/3.0;
        if idef2==0
            ld=rand*(kdl-1); ii=floor(ld)+1; alfa=ld-ii+1;
            ZD2(:,itd,idf)=D(:,ii,idf)*(1-
alfa)+D(:,ii+1,idf)*alfa+ZE(:);
        else
            idf1=2*idf-1; idf2=2*idf;
            ld1=rand*(kdl-1); ld2=rand*(kdl-1);
            if (ld1+ld2)>(kdl-1)
                ld1=(kdl-1)-ld1; ld2=(kdl-1)-ld2;
            end;
            ii1=floor(ld1)+1; alfa1=ld1-ii1+1;
            ii2=floor(ld2)+1; alfa2=ld2-ii2+1;
            ZD_1(:)=D(:,ii1,idf1)*(1-
alfa1)+D(:,ii1+1,idf1)*alfa1;
            ZD_2(:)=D(:,ii2,idf2)*(1-
alfa2)+D(:,ii2+1,idf2)*alfa2;
            ZD2(:,itd,idf)=ZD_1(:)+ZD_2(:)+ZE(:);
        end;
    end;
end;
% Formación de entradas a la red
for idf=1:kdf
    for itd=1:ktd
        n=(idf-1)*ktd+itd;
        P1(:,n)=ZD1(:,itd,idf);
        T1(idf,n)=1;
    end;
end;
for idf=1:kdf
    for itd=1:ktd
        n=(idf-1)*ktd+itd;
        P2(:,n)=ZD2(:,itd,idf);
        T2(idf,n)=1;
    end
end
****inicio 3****
h_frame= uicontrol(h_fig,'style','text',...
'Units','normalized',...
'Position',[0.6,0.9,0.05,0.02],...
'ForegroundColor',[0,0,1],...
'FontWeight','bold',...
'String','METODO');
h_pop5= uicontrol(h_fig,'style','popupmenu',...
'Units','normalized',...
'Position',[0.6,0.835,0.08,0.06],...
'String','Trainlm|Traingd|Traingdm|Traingda|Traingdx|T
rainrp|Traingcf|Traingcp|Traingcb|Traingcg|Trainbfg|Tr
ainnoss',...
'BackgroundColor',[0.5,0.5,0.5],...
'ForegroundColor',[0,1,1],...
'FontWeight','bold',...
'Callback','pruebared');
Handles.pop5=h_pop5;
set(h_fig,'UserData',Handles);
****Inicio 4****
h_frame= uicontrol(h_fig,'style','text',...
'Units','normalized',...
'Position',[0.8,0.9,0.05,0.02],...
'ForegroundColor',[0,0,1],...

```

```

    'FontWeight','bold',...
    'String','Neuronas');
h_popa=uicontrol(h_fig,'style','popupmenu',...
    'Units','normalized',...
    'Position',[0.8,0.85,0.035,0.045],...
    'String','1|2|3|4|5|6|7|8|9|10|11|12|13|14|15|20|25|30',...
    'BackgroundColor',[0.1,0.5,0.5],...
    'ForegroundColor',[0,0,0],...
    'Callback','radial');
Handles.popa=h_popa;
set(h_fig,'UserData',Handles);
****sigue1****
h_fig=gcbf;
H=get(h_fig,'UserData'); %datos guardados en figura
h_pop5=H.pop5;
%% %% puntos de simulacion
method_select=get(h_push1,'Value');
switch method_select
    case 1
        net
        =newff(MiMa,[12,9],{'tansig','logsig'},'trainlm');
        net.trainParam.epochs=100;
        net.trainParam.show=5;
        net.trainParam.mem_reduc=2;
    case 2
        net
        =newff(MiMa,[12,9],{'tansig','logsig'},'traingd');
        net.trainParam.lr=0.1;
        net.trainParam.epochs=8000;
        net.trainParam.show=50;
    case 3
        net
        =newff(MiMa,[12,9],{'tansig','logsig'},'traingdm');
        net.trainParam.lr=0.1;
        net.trainParam.mc=0.5;
        net.trainParam.epochs=8000;
        net.trainParam.show=50;
    case 4
        net
        =newff(MiMa,[12,9],{'tansig','logsig'},'traingda');
        net.trainParam.lr=0.1;
        net.trainParam.lr_inc=1.05;
        net.trainParam.epochs=1000;
        net.trainParam.show=50;
    case 5
        net
        =newff(MiMa,[12,9],{'tansig','logsig'},'traingdx');
        net.trainParam.lr=0.1;
        net.trainParam.mc=0.5;
        net.trainParam.lr_inc=1.02;
        net.trainParam.epochs=3000;
        net.trainParam.show=50;
    case 6
        net =newff(MiMa,[5,9],{'tansig','logsig'},'trainrp');
        net.trainParam.epochs=300;
        net.trainParam.show=20;
    case 7
        net
        =newff(MiMa,[12,9],{'tansig','logsig'},'traincgf');
        net.trainParam.epochs=200;
        net.trainParam.show=20;
    case 8
        net
        =newff(MiMa,[12,9],{'tansig','logsig'},'traincgp');
        net.trainParam.epochs=200;
        net.trainParam.show=20;
    case 9
        net
        =newff(MiMa,[12,9],{'tansig','logsig'},'traincgb');
        net.trainParam.epochs=200;
        net.trainParam.show=20;
    case 10
        net
        =newff(MiMa,[12,kdf],{'tansig','logsig'},'trainscg');
        net.trainParam.epochs=200;
        net.trainParam.show=20;
    case 11
        net
        =newff(MiMa,[12,9],{'tansig','logsig'},'trainbfg');
        net.trainParam.epochs=200;
        net.trainParam.show=20;
    case 12
        net =newff(MiMa,[12,9],{'tansig','logsig'},'trainoss');
        net.trainParam.epochs=200;
        net.trainParam.show=20;
end
****Radial****
h_fig=gcbf;
H=get(h_fig,'UserData'); %datos guardados en figura
h_popa=H.popa;
%% %% neuronas por capa
neuronas_select=get(h_popa,'Value');
switch neuronas_select
    case 1,
        neurona=1;
    case 2
        neurona=2;
    case 3
        neurona=3;
    case 4
        neurona=4;
    case 5
        neurona=5;
    case 6
        neurona=6;
    case 7
        neurona=7;
    case 8
        neurona=8;
    case 9
        neurona=9;
    case 10
        neurona=10;
    case 11
        neurona=11;
    case 12
        neurona=12;
    case 13
        neurona=13;
    case 14
        neurona=14;
    case 15

```

```

    neurona=15;
case 16
    neurona=20;
case 17
    neurona=25;
case 18
    neurona=30;
end
neu=neurona;
tic;
% Entrenamiento de la red.
net =newrb(P1,T1,0.026,1,neu,2);
A1=sim(net,P1); % Simulación
A2=sim(net,P2); % Simulación
% Cálculo de probabilidades
PD1=zeros(kdf);
for idf=1:kdf
    for itd=1:ktd
        n=(idf-1)*ktd+itd;
        [amax,imax]=max(A1(:,n));
        PD1(imax,idf)=PD1(imax,idf)+1;
    end;
end;
PD1=PD1/ktd;
PDT1=diag(PD1);
psr1=mean(PDT1);
PD1T=PD1';
fw1=fopen('pd1.dat','wt');
if idef2==0
    count=fprintf(fw1,'%6.4f %6.4f %6.4f %6.4f %6.4f %6.4f
%6.4f %6.4f %6.4f %6.4f\n',PD1T,PDT1,psr1);
else
    count=fprintf(fw1,'%6.4f %6.4f %6.4f %6.4f
%6.4f\n',PD1T,PDT1,psr1);
end;
fclose(fw1);
PD2=zeros(kdf);
for idf=1:kdf
    for itd=1:ktd
        n=(idf-1)*ktd+itd;
        [amax,imax]=max(A2(:,n));
        PD2(imax,idf)=PD2(imax,idf)+1;
    end;
end;
PD2=PD2/ktd;
PDT2=diag(PD2);
psr2=mean(PDT2);
PD2T=PD2';
fw2=fopen('pd2.dat','wt');
if idef2==0
    count=fprintf(fw2,'%6.4f %6.4f %6.4f %6.4f %6.4f %6.4f
%6.4f %6.4f %6.4f %6.4f\n',PD2T,PDT2,psr2);
else
    count=fprintf(fw2,'%6.4f %6.4f %6.4f %6.4f
%6.4f\n',PD2T,PDT2,psr2);
end;
fclose(fw2);

figure(1);
subplot(1,2,2);
plot3(ZD1(1, :, 1),ZD1(2, :, 1),ZD1(3, :, 1),'*',ZD1(1, :, 2),Z
D1(2, :, 2),ZD1(3, :, 2),'+',ZD1(1, :, 3),ZD1(2, :, 3),ZD1(3, :, 3
),'x',ZD1(1, :, 4),ZD1(2, :, 4),ZD1(3, :, 4),'x'); grid on;
legend('D1','D2','D3','D4');
xlabel('Z1 - Presion de compresor ');
ylabel('Z2 - Presion de turbina ');
zlabel('Z3 - Temperatura de compresor');
time=toc;
psr2;
h_frame= uicontrol(h_fig,'style','text',...
'Units','normalized',...
'Position',[0.3,0.05,0.05,0.035],...
'ForegroundColor',[0,0,1],...
'FontWeight','bold',...
'FontSize',12,...
'String','PD2');
h_frame= uicontrol(h_fig,'style','text',...
'Units','normalized',...
'Position',[0.35,0.05,0.055,0.035],...
'ForegroundColor',[0,0,1],...
'FontWeight','bold',...
'FontSize',12,...
'String',psr2);
h_frame= uicontrol(h_fig,'style','text',...
'Units','normalized',...
'Position',[0.5,0.05,0.065,0.035],...
'ForegroundColor',[0,0,1],...
'FontWeight','bold',...
'FontSize',12,...
'String','TIEMPO');
h_frame= uicontrol(h_fig,'style','text',...
'Units','normalized',...
'Position',[0.56,0.05,0.065,0.035],...
'ForegroundColor',[0,0,1],...
'FontWeight','bold',...
'FontSize',12,...
'String',time);
***Probabilistics***
% Net training
P=P1;
T=T1;
net =newpnn(P,T,0.026);
****regression_general****
% Net training
P=P1;
T=T1;
net =newgrnn(P,T,0.026);

***Newrb***

% Net training
P=P1;
T=T1;
net =newrb(P,T,0.026,1,13,5);

```



APÉNDICE B: PUBLICACIONES

Comparación de dos tipos de redes neuronales artificiales (retropropagación y base radial), utilizadas para el diagnóstico paramétrico de turbinas de gas

Comparison of two types of artificial neural networks (backpropagation and radial basis), used for parametric gas turbines diagnosis

Eulalio Torres García², Igor Loboda¹, Jesús Martínez López².

¹Profesor en Instituto Politécnico Nacional, SEPI- Culhuacán, ²Estudiante en Instituto Politécnico Nacional, SEPI- Culhuacán. Contacto: ingetorresg@gmail.com

RESUMEN: Este artículo está orientado a la comparación de dos tipos de redes Neuronales Artificiales (RNA), tipo Retropropagación y de Base Radial, aplicadas en el Diagnóstico de Turbinas de Gas. Se hace un análisis del comportamiento con 12 diferentes funciones de adaptación, combinaciones de 1 a 3 en los números llave para la iniciación de las vectores de comportamiento alrededor de clases identificadas, se utilizan 11 regímenes de operación de la turbina, todo ello controlado por una interface usuario que permite reducir el tiempo de procesamiento de datos. El resultado es arrojado a una probabilidad de aserción (calculada por estadística) de un diagnóstico correcto y el tiempo aproximado de entrenamiento. Comparado estos resultados, es posible determinar cuál es el tipo de red y el tipo de entrenamiento que debe tener una RNA, para ser utilizada en esta aplicación.

Palabras clave: Redes Neuronales Artificiales de Retropropagación (perceptrón), Redes de Base Radial, Diagnóstico de turbinas de gas, Clasificación de fallas, Probabilidad de Diagnóstico Correcto.

1. INTRODUCCIÓN.

Las turbinas de gas son sistemas muy complejos, que implica un método de diagnóstico de igual manera complejo. Entre mayor sea el número de componentes del sistema, hay mayor probabilidad de falla. Ello lleva a requerir una técnica que sea capaz de detectar las fallas con prelación en cada componente y así evitar pérdidas mayores. Las fallas y los procesos de deterioración afectan la fiabilidad, disponibilidad del equipo y los costos de operación. La aplicación de sistemas de diagnóstico y monitoreo, los cuales han llegado a ser una práctica estándar, permiten disminuir estos efectos negativos.

Actualmente el proceso de diagnóstico es un tema que implica conocer el motor y su comportamiento en estado normal, de este modo es posible identificar cuando tiene algún síntoma de falla. En el proceso de diagnóstico se aplican algoritmos basados en el estado normal del motor (estado inicial), algunas técnicas de reconocimiento de patrones y modelos avanzados de probabilidad que permiten identificar las desviaciones.

Una vez que se conoce el comportamiento del motor, es muy importante tomar en cuenta que no es correcto utilizar los parámetros directamente medidos, ya que el cambio de régimen puede esconder cambios inducidos por las fallas. En todos los métodos del diagnóstico paramétrico se usan las desviaciones de los parámetros medidos

$$\delta Y = \frac{Y^* - Y_0}{Y_0} \dots \dots \dots (1)$$

en donde: Y^* es el parámetro medido actual del conducto de flujo; Y_0 es el parámetro del estado normal.

Loboda I (2003) menciona que el estado normal de una turbina de gas implica un estado fijo, que cumpla con los requisitos de una turbina, por ejemplo, el estado después de la fabricación. Este estado depende del régimen de funcionamiento de la turbina, por lo tanto, se puede modelar por medio de una función de estado normal, llamada modelo de estado normal.

Así podemos presentar en la forma general, un modelo de estado normal, como:

$$\vec{Y} = f_1(\vec{U}) \dots \dots \dots (2)$$

en donde: \vec{U} vector del régimen de turbina de gas y de las condiciones ambientales (temperatura y presión del aire).

Para formar la descripción completa de la turbina es necesario usar un modelo termodinámico no lineal del conducto de flujo, que se escribe por la siguiente fórmula:

$$\vec{Y} = f_2(\vec{U}, \vec{\theta}) \dots \dots \dots (3)$$

en donde: $\vec{\theta}$ es el vector de los parámetros de estado, los cuales son capaces de corregir las características de los componentes de las turbinas de gas. Al poner en la formula (3) el vector fijo $\vec{\theta}_0$, que corresponde al estado normal transformamos el modelo termodinámico:

$$\vec{Y} = f_2(\vec{U}, \vec{\theta}_0) \dots \dots \dots (4)$$

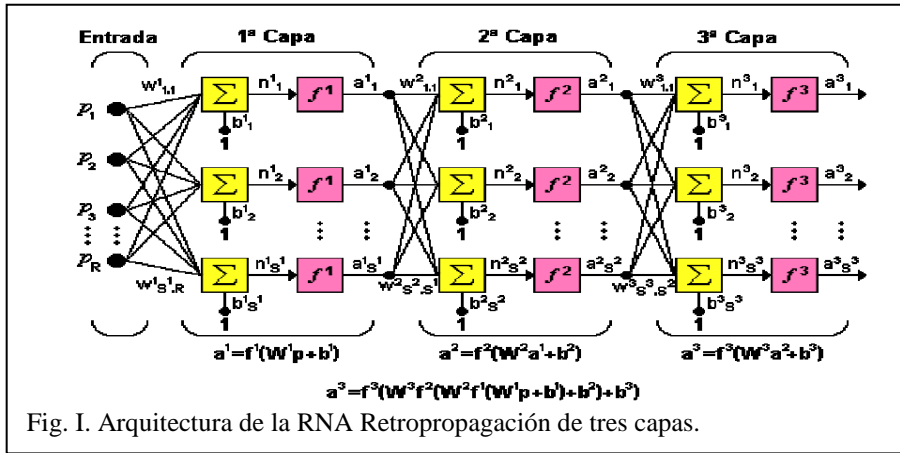
que corresponde al modelo normal (2).

Partiendo del modelo termodinámico, con una serie de datos obtenidos del motor, es posible generar patrones de comportamiento. En este patrón o vector de desviaciones ($\delta\vec{Y}$), cada puto representa una medición en el conducto de flujo. Es necesario considerar que a las desviaciones se deben agregar errores aleatorios ($\delta\vec{Y}^*$), que nos permitan introducir mediciones erróneas (ruido) y a pesar de ello generalizar el comportamiento. Ya que estos patrones que son utilizados para el entrenamiento, nos servirán para reconocer, clasificar y diagnosticar el estado del motor.

2. REDES NEURONALES ARTIFICIALES DE RETROPROPAGACIÓN.

Retropropagación (backpropagation) es la generalización de la regla de aprendizaje Widrow-Hoff para redes neuronales multicapa y funciones de transferencia no diferenciables y no lineales. Los vectores de entrada (input) y los correspondientes vectores de salida (target) son utilizados para entrenar la red hasta que se pueden aproximar a una función. Esta está asociada con los vectores de entrada y los vectores específicos de salida, o clasificación de los vectores de entrada en el apropiado camino definido por el usuario (Howard Demuth 2008).

La arquitectura base de cualquier RNA tipo Retropropagación y que a su vez es la base de nuestro estudio es la que se muestra en la Fig. I.



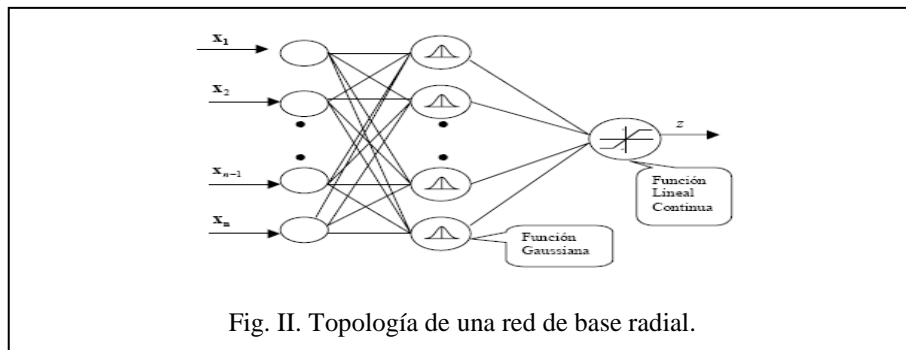
Existen diferentes caminos en los cuales se puede aumentar el desempeño del entrenamiento. Puede ser a través del control de las funciones de activación, que no es más que la forma en cómo se va a calcular los pesos en cada iteración, logaritmo tangencial, logaritmo sigmoidea o lineal, son las básicas. Otra manera de hacer la aproximación más exacta o en menor tiempo es el incremento del número de neuronas por capa o bien el número de capas ocultas. En algunos casos este cambio aumenta la exactitud de la función y/o disminución del tiempo de entrenamiento. Cabe aclarar que esto no es regla, pues bien una vez más es necesario recordar que dependiendo de la aplicación, es la función que darán mejores resultados.

Una función adaptativa es un ente formado por la neurona artificial y un algoritmo de corrección de pesos. La neurona artificial es modelada mediante una combinación lineal, esto es, produce una salida (en cada iteración) $y(n)$ que es el resultado de combinar linealmente las entradas $x(n)$, cada uno con su peso $w(n)$ característico de cada neurona. Los pesos son adaptables, y se modifican tras cada salida mediante un algoritmo de corrección. Dicho algoritmo, calculado como la diferencia entre la salida de la neurona y la salida deseada, que llamamos $d(n)$ y que es conocida. Así el método, no es más que n iteraciones para resolver la optimización. El problema se reduce a encontrar el valor de peso w_0 que minimiza el error y con ello alguna función dependiente del mismo que tome su valor mínimo en el mismo punto (*Manual de Redes neuronales - 2005*).

En el toolbox de MATLAB existen 12 algoritmos de entrenamiento, `trainlm`, `traingd`, `traingdm`, `trainda`, `traingdx`, `trainrp`, `traincgf`, `traincgp`, `traincgb`, `traincsg`, `trainbfg`, `trainoss`. Estos algoritmos han demostrado ser buenos para diferentes aplicaciones, haciendo imposible decir que un algoritmo es mejor que otro. Lo que los hace diferentes es la forma en cómo manejan los pesos a la salida para la convergencia entre la salida deseada y la obtenida, por ejemplo, en dirección del gradiente, utilizando un grado de aprendizaje (lr-learning rate) que determinara el número de veces que se debe incrementar el gradiente. Otra forma es sumar al gradiente la tendencia reciente en la superficie del error. Existen algunos parámetros que permiten a las redes aprender más rápido o bien las hace oscilar y por ende se vuelven inestables, estas variables que permiten a las redes ser más o menos eficientes, son las que se buscan para encontrar las más eficientes en nuestra aplicación.

3. REDES NEURONALES ARTIFICIALES DE BASE RADIAL

Una red de base radial (Radial Basis Function - RBF) está diseñada con neuronas en la capa oculta activadas mediante funciones radiales de carácter no lineal con sus centros gravitacionales propios. En la capa de salida se controla mediante funciones lineales. Aunque se reconoce que en la fase de entrenamiento es una red que presenta un alto desempeño, en general no se utiliza para altos volúmenes de datos, porque con lleva tiempo de entrenamiento muy largo. Lo cual hace que la red sea ineficiente para nuestro contexto, ya que si se desea utilizar para diagnóstico en tiempo real, para el momento que esta alcance el tiempo entrenamiento, posiblemente la turbina ya habrá fallado. El entrenamiento a diferencia de Retropropagación es solo hacia adelante. La salida es influenciada por una transformación no lineal originada en la capa oculta a través de una función radial y una lineal en la capa de salida por una función lineal continua (*Gutiérrez Mojica Eder M. 2005*). En la Fig. II. se puede observar la topología general de una RBF.



4. ALGORITMO

El algoritmo que se ha diseñado consta de 4 partes. La primera es la interface usuario, en ella se introducen todos los valores constantes, como el régimen de operación. Permitiendo tener una base de datos constante para más de un experimento, así disminuimos el tiempo de operación entre un experimento y el siguiente. Por otro lado al ser la misma base de operación los resultados pueden ser objeto de comparación. Además permite una rápida y eficiente manipulación de los datos de inicialización de la red.

La segunda es el ensamble de los datos; estos deben ser adecuados para que puedan ser leídos en forma correcta por el compilador. Esta parte del algoritmo tomara de la interface usuario las constantes, leerá los datos que fueron obtenidos de una turbina, preparara el arreglo multidimensional y proveerá el vector de aprendizaje. Para estos efectos se experimentó utilizando solo 9 clases diferentes de falla. En nuestro caso de estudio aceptamos la hipótesis de que el estado actual del motor pertenece solo a una clase, es decir no hay falla simultánea en más de un componente al mismo tiempo. Después se generó las nubes de puntos ($\delta\vec{Y}^*$) de manera estadística alrededor de nuestras clases identificadas. Esa generación de puntos se hace por medio de una función estadística de cercanía, llamada 3 sigma (3σ), la cual nos asegura que los puntos generados estén dentro de la clase adecuada (probabilidad 99.7%), incluyendo el espacio multidimensional. En la misma etapa se proporcionan las muestras de entrenamiento y validación para la posterior clasificación.

La tercera parte es el entrenamiento de la red, para ello tomamos de los datos previamente preparados. En cada tipo de red se debe especificar el número de épocas, error mínimo permitido, y valores propios del algoritmo de entrenamiento que se está utilizando. El proceso de clasificación toma una muestra de entrenamiento, y con una muestra de validación hace una comparación. Si el valor del error sobrepasa el deseado se regresa al entrenamiento de lo contrario regresa los vectores que forman parte de nuestra función ya entrenada.

La cuarta parte hace el cálculo de probabilidades de forma estadística. Se calcula una matriz de probabilidades y en pantalla solo se muestra el promedio de probabilidades de la segunda matriz, la cual fue de prueba. Se corta el tiempo de operación y también se envía a pantalla. Por último se gráficas las clases, se muestran solo cuatro con el objetivo de hacerlas más visibles.

5. EXPERIMENTOS PRELIMINARES

En primera instancia se definió cuál es el objetivo de la experimentación. Si bien el objetivo es comparación, tenemos que tener un marco de referencia, el objetivo primordial fue medir el tiempo de operación y la eficiencia de la red neuronal, siendo lo más importante la probabilidad de dar un diagnóstico correcto. El algoritmo fue diseñado para calcular dos matrices de probabilidades, como se menciono anteriormente. La primera no es mucho de nuestro interés ya que es la matriz obtenida en el entrenamiento, por ende al hacer el cálculo de probabilidad, el porcentaje de aserción será muy alto. La probabilidad que es calculada en la segunda muestra, es la matriz que se obtiene como resultado de prueba de la red, esta es de más importancia.

Etapas de experimentación; en la primera de ellas simplemente se introdujeron valores estándar dependientes de cada filtro adaptativo, como numero de épocas, grado de aprendizaje, coeficiente de momento, etc., con ello se observó el comportamiento de la red. Como se puede observar en la Fig. III., estos valores son definitorios del tiempo y el resultado obtenido, no debemos iniciar una red que necesita más de 2000 épocas

con 100, pues ello dará probabilidades de aserción muy bajas, o viceversa, ello llevara al sobre entrenamiento, extensión del tiempo y resultados no satisfactorios.

metodo	1) trainlm	2) Traingd	3)Traingdn	4)traingda	5)
Tiempo	10.53	15.41	15.20	1.59	
performance	0.0359	0.05	0.05	0.02814	
epocas	100	8000	8000	1000	
PD1	0.7451	0.715	0.7142	0.8274	
PD2	0.745	0.714	0.7136	0.8178	

Fig. III. Tabla inicial comparativa de datos sin optimización. Demuestran baja probabilidad (PD) y errores hasta del 5 % y tiempo de hasta 15 minutos.

Con estos resultados se inicializa la experimentación, buscando en primera instancia aumentar la probabilidad obtenida para la muestra 2. Como se puede observar en la Fig. IV. en un principio la probabilidad era muy baja. La probabilidad 2 (PD2) está íntimamente ligada con la gráfica de desempeño (performance), esta gráfica es la suma de los errores. Si se reduce el valor en la gráfica de performance indica que estamos reduciendo el error y por ende aumentando la probabilidad de acierto.

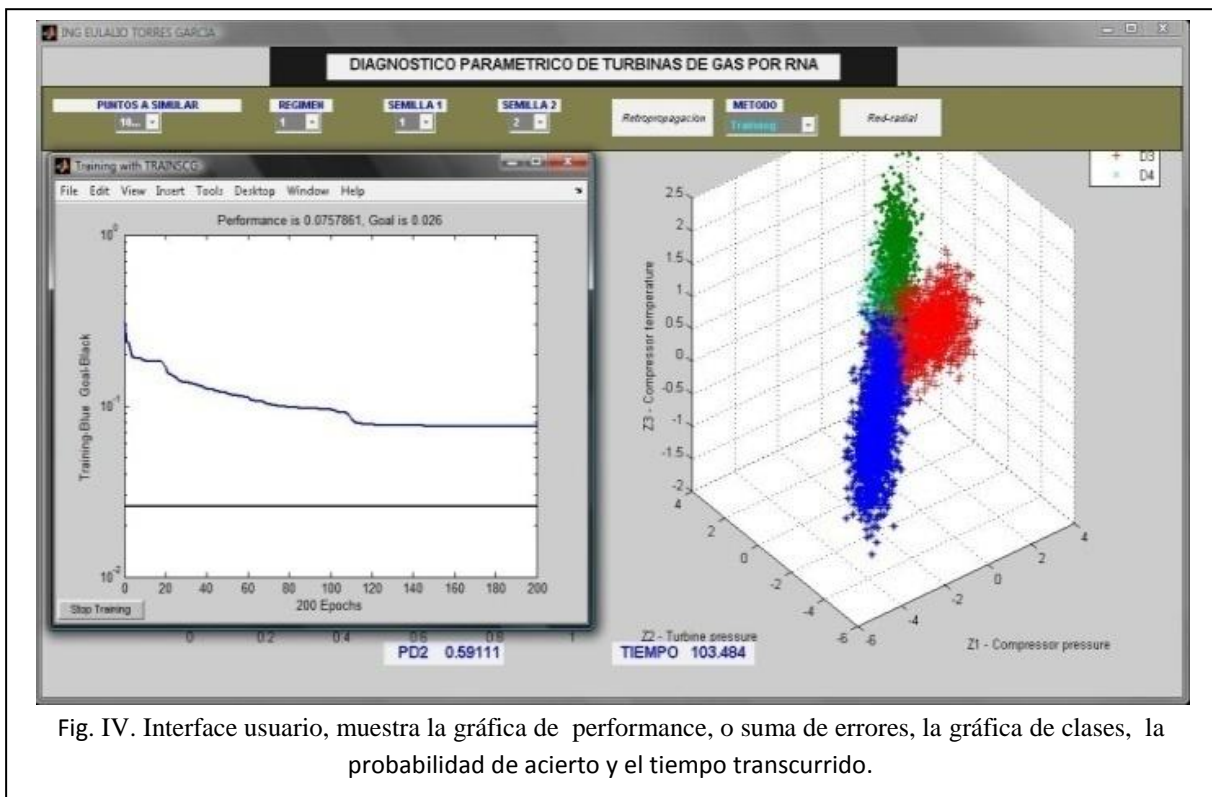


Fig. IV. Interface usuario, muestra la gráfica de performance, o suma de errores, la gráfica de clases, la probabilidad de acierto y el tiempo transcurrido.

En la siguiente etapa, el parámetro a controlar es el tiempo de entrenamiento, pues bien de nada sirve tener una RNA que sea capaz de identificar el patrón de comportamiento después de horas. Si el diagnóstico se quiere hacer en tiempo real, pues después de horas es posible que el motor ya haya fallado, entonces el diagnóstico se vuelve inútil.

En la siguiente etapa se cambió el número de épocas, disminuimos o aumentamos dependiendo del comportamiento de la gráfica de performance, pues si la gráfica permanecía estable por demasiadas épocas, no es necesario tuviese tantas épocas, entonces se reduce el valor hasta el mínimo para dar un valor de probabilidad aceptable.

Las nubes de puntos se generan aleatoriamente, cada una de estas nubes conforman una clase de falla, en cada una de ellas contendrá n puntos (número que nosotros especificamos en el volumen al iniciar el cálculo),

siendo estos puntos vectores de comportamiento del motor. Estas deben ser inicializadas con los mismos valores, de otra manera es imposible compara los resultados. Para la inicialización se utilizan números llave (semillas), entre 1 y 3. Así la siguiente etapa es inicializar el programa con diferentes números llave para observar la generalización del entrenamiento.

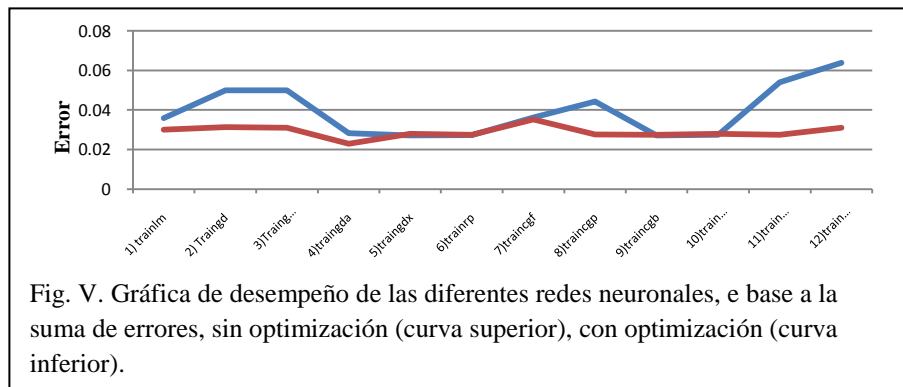
Los datos provenientes del motor pertenecen a 11 regímenes de operación diferentes. Una vez que hemos logrado reducir el número de épocas, y hemos cambiando los otros valores propios de cada algoritmo de entrenamiento, se disminuye el tiempo de convergencia de la función, en general se ha optimizado la red. Se procede a cambiar el régimen de operación, para observar que estos valores sean validos para diferentes regímenes de operación.

La siguiente etapa es referente a el número de puntos de simulación (volumen de datos), en primera instancia se utilizaron 500 puntos por clase de simulación, en donde cada punto representa un vector (patrón $\delta\bar{Y}^*$), se hace el mismo procedimiento pero ahora para 1000, 1500 y 2000 puntos de simulación, para cada clase de cada muestra, con ello observamos si la red sigue arrojando los mismos resultados. Por ende es estable para la utilización con diferentes volúmenes de datos.

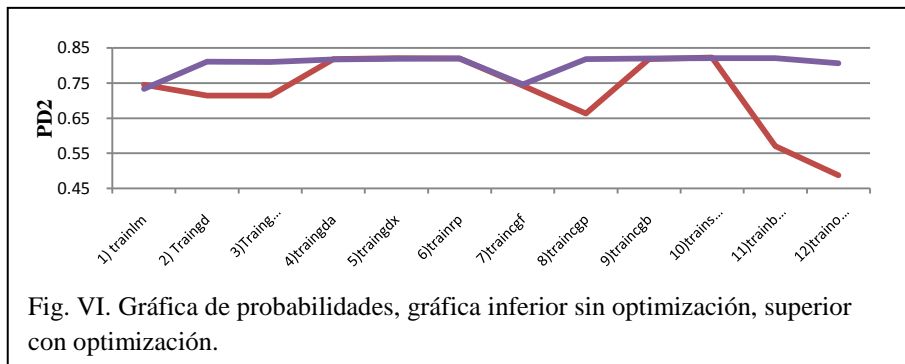
La última etapa es hacer el mismo procedimiento para la comparación con redes de tipo radial, haciendo diferentes cambios en los valores propios de la función para lograr un resultado comparable con retropropagación. También se aumento el número de capas ocultas y el número de neuronas por capa para observar el comportamiento de la red, con ello de igual manera redundamos en el objetivo. Por último se compara con los resultados obtenidos de la experimentación con redes de retropropagación.

RESULTADOS

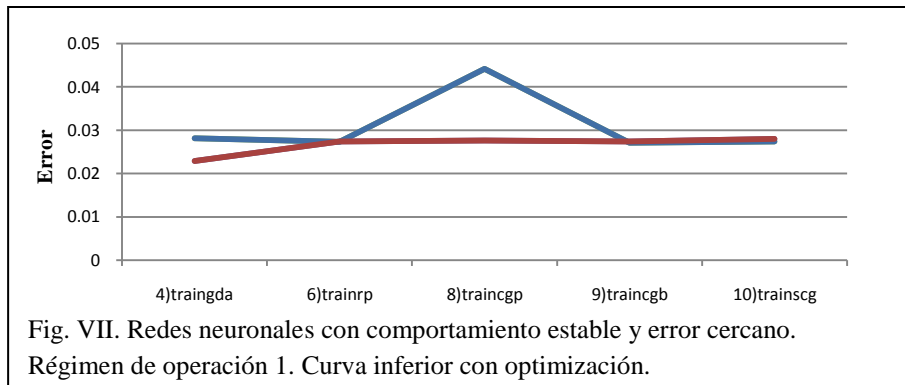
El primer paso, fue analizar las redes sin ningún tipo de optimización eligiendo el número de neuronas al azar y observando el comportamiento, en la Fig. V., curva superior, se puede apreciar la dispersión en los resultados, aunque algunos se acercaban al resultado deseado en cuanto a performance. Ajustando algunos parámetros, como número de neuronas o funciones de activación, se obtuvieron errores aproximadamente iguales, como se puede observar la dispersión de los datos al iniciar la experimentación es demasiado grande, al hacerla optimización se logran resultados aproximadamente iguales (curva inferior Fig. V.), por lo menos en lo la referencia de error. En referencia a tiempo los resultados siguen siendo dispersos.



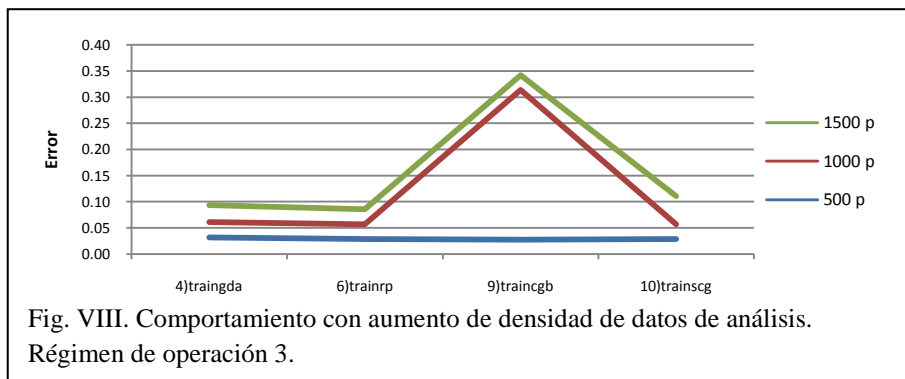
Al comparar las gráficas de desempeño (Fig. V) y probabilidad de aserción (Fig. VI) es obvio la relación inversamente proporcional que tiene error calculado en la red con la probabilidad de aserción en el diagnóstico. Durante la primera etapa de experimentación el volumen de datos fue con solo 500 puntos de operación. Y el primer régimen de operación. Una vez que se ha hecho la optimización de cada procedimiento es posible decir que todas las redes, son capaces de alcanzar la misma probabilidad de aserción (curva superior Fig. VI.).



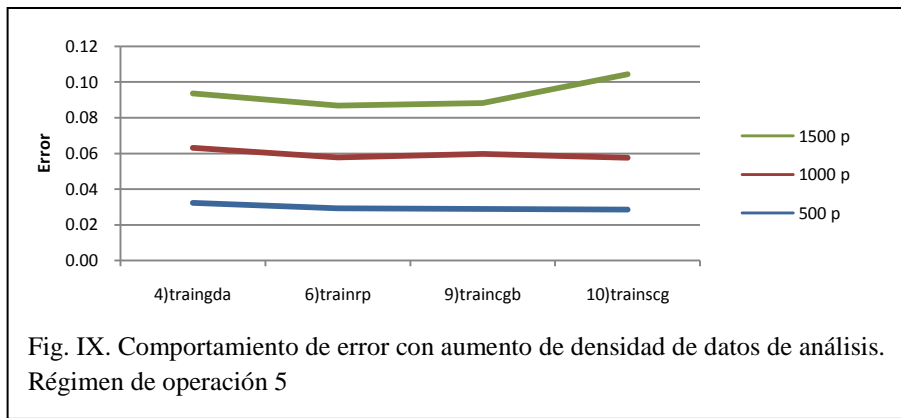
Sin embargo, cuando nos referimos a el tiempo de operación, siendo las funciones de clasificación que se muestran en Fig. VII. las que fueron capaces de entrenar la red en menor tiempo esto es en un rango de 30 a 150 segundos. Y como se puede observar el error esta en el rango de 2.5 y 4 %. Hubo algunas redes que tomaban hasta 30 minutos en el entrenamiento solo con 500 puntos por clase.



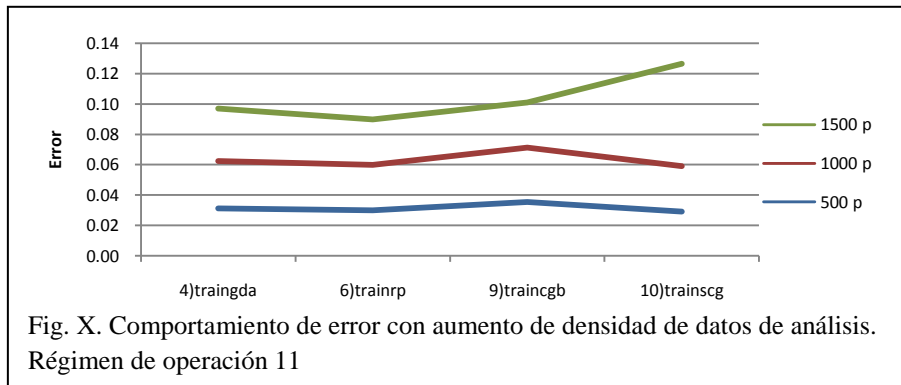
Al cambiar el régimen de operación es posible observar y eliminar las redes que tiene comportamiento inestable, como en el caso de traincgb, la cual se comporto excelente con 500 puntos de operación en el primer régimen (Fig. VIII.), y no presenta problemás en los regímenes subsecuentes. Pero sin embargo, presenta problemás al hacer el aumento del volumen de datos, aumenta el error hasta el 35%, el resto de ellos fueron eliminados por mostrar errores de hasta el 40%.



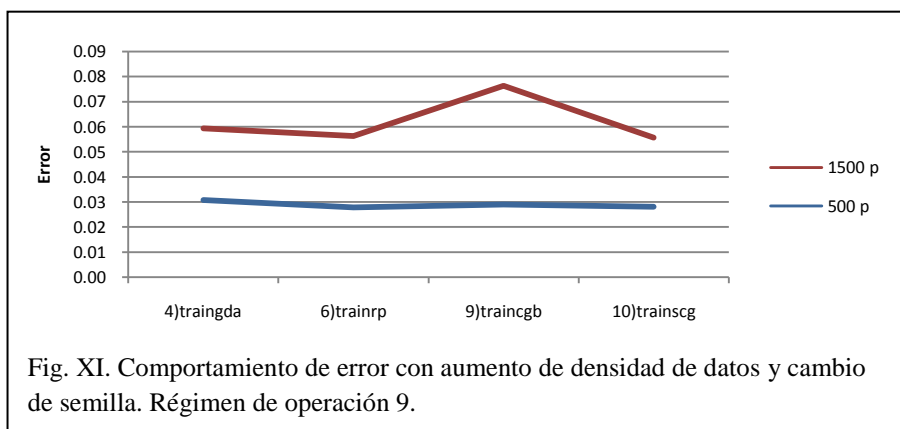
Al hacer el aumento del volumen de datos, como se puede apreciar en la Fig. IX., se observo que el error aumenta directamente proporcional al número de datos analizados, aun así no sobrepasando en las redes seleccionadas el 12%. Una vez más es interesante ver que con el cambio de régimen la función traincgb se comporta de manera estable.



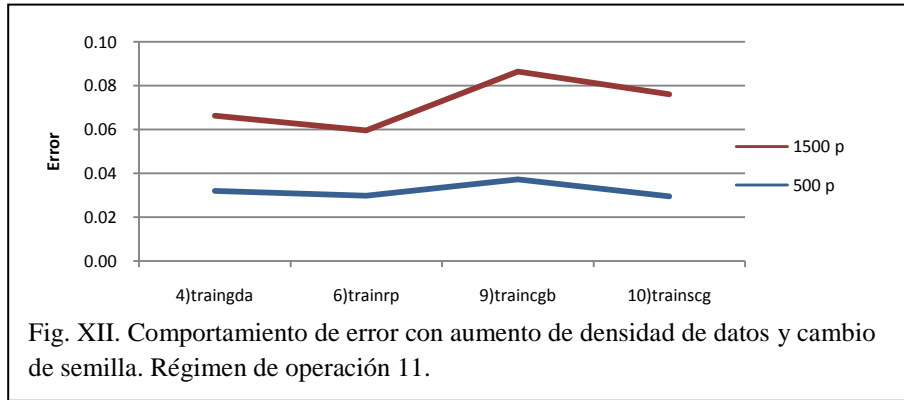
En los regimenes subsecuentes el comportamiento de las redes es muy parecido. La red entrenada se mantiene estable y aumenta el error directamente proporcional al numero de datos. El tiempo de calculo tambien incrementa en forma considerable. Las redes entrenadas con funciones traincgb y trainscg en algunos regimenes de operación son muy inestables, aumentando el error. También presenta un incremento del error de manera no lineal en relacion con el numero de puntos utilizados para la simulacion. Como se puede observar en la Fig. X el error no necesariamente esta dado por una funcion lineal, por ejemplo en la función trainscg el error es proporcional en el primer aumentode datos, pero al hacer el experimento con 1500 puntos el error aumenta practicamente el doble del calculo anterior.



Como se muestra en la Fig. XI. al analizar el cambio de llave de inicialización de datos es posible observar, que la mayoría de redes siguen teniendo la misma tendencia. El resultado no depende del número de inicialización (semilla), sino del tipo de entrenamiento que esté recibiendo la red, siendo una vez más traincgb la que presenta una fluctuación mayor en comparación con los otros tipos de entrenamiento.



En la Fig. XII, demostramos que el cambio de regimen no tiene impacto en el entrenamiento de la red, una vez más es posible observar que el comportamiento del error depende enteramente del numeros de puntos de operación.



Las redes neuronales de tipo retropropagacion demostraron un comportamiento bastante confiable, y en cuanto a tiempo demostraron ser capaces de dar un diagnóstico en tiempo muy corto, pues el tiempo en la fase de entrenamiento no va más allá de los 2 minutos.

Con las redes del tipo radial, al hacer la configuracion basica demostraron dos tipos de comportamiento. Primero, si no se especifica el numero de neuronas de la red, esta toma el tamaño de la matriz y utiliza el numero de columnas, asi pues, esta no tiene la capacidad de hacer calculos con grandes volumenes de datos. Segundo, si se reducen el numero de puntos es posible que esta reduzca de igual manera el numero de neuronas, y aunque es capaz de realizar el calculo, no tiene una probabilidad adecuada. Como se muestra en la Fig. XIII., se reduce el error a aproximadamente 7 % (SSE), pero se presenta el fenomeno de sobre entrenamiento, al hacer la prueba esta solo tiene una probabilidad de dar un diagnóstico correcto del 30%, y el tiempo de entrenamiento se ve severamente castigado, tomando aproximadamente 78 horas para el entrenamiento.

```

NEWRB, neurons = 4425, SSE = 1.04526
NEWRB, neurons = 4450, SSE = 0.466058
NEWRB, neurons = 4475, SSE = 0.0719039

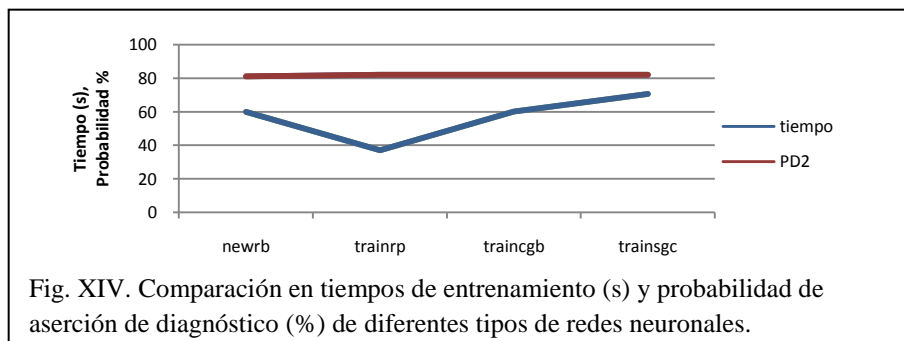
t =
  4.9411e+005

psr2 =
  0.3098

```

Fig. XIII. Ejemplo de sobre-entrenamiento de red base radial.

Más sin embargo, es posible disminuir el tiempo de operación y de igual manera aumentar ella probabilidad de aserción, incluso al aumentar el número de puntos de simulación la red neuronal es capaz de dar una probabilidad de acierto considerable, la cual ya es comparable con los resultados arrojados por las redes de retropropagación. Trabajando con número de neuronas en los rangos de 15 y 20 se pudo alcanzar el tiempo y la probabilidad deseada para ser comparada con retropropagación. Como se puede observar en la Fig. XIV. La red de base radial tiene pequeñas diferencias en cuanto a probabilidad, aún así ya es capaz de acertar en el diagnóstico y el tiempo de entrenamiento de la red es comparable con el de las redes de tipo retropropagación.



CONCLUSIONES

En este artículo hemos descrito la experimentación de un algoritmo utilizado para el diagnóstico paramétrico de turbinas de gas basado en redes neuronales artificiales. Se utilizó un modelo termodinámico no lineal para simular los modos de degradación de la turbina de gas y las fallas de clasificación. Obtenemos y verificamos los índices de diagnóstico adecuados que permiten decir si nuestra turbina puede ser diagnosticada correctamente. Es importante recordar que las mediciones que fueron tomadas no se utilizan directamente sino que utilizamos desviaciones para simular fallas reales, con ello introducimos al modelo diferentes fallas y se considera el ruido en patrones simulados. Se genera la nube de puntos de cada clase, en donde cada punto representa un vector de desviaciones inducidas por fallas simuladas (patrón).

La experimentación en este tipo de algoritmos puede llegar a ser muy larga, dependiendo del tipo de procesador se puede aumentar o disminuir los tiempos de operación. Siendo estos experimentos los que conforman la base de investigaciones posteriores, pues ahora somos capaces de decir cuál es la red neuronal que tiene mejor comportamiento para el tipo de datos que se manejan, y el volumen de las bases de datos, sabemos que son datos que no pueden tener mucha variación, también sabemos que este tipo de redes neuronales son capaces de eliminar el ruido dentro de los datos medidos, así podemos utilizar estas como interface para poder diagnosticar en tiempo real una turbina en operación.

Hay cuatro funciones (*trainrp*, *traiscgb*, *traiscg*, *newrb*) que presentan un comportamiento similar en el entrenamiento. De ellas, tres fueron funciones de entrenamiento de perceptrón y una es la red con RBF por base radial en su forma básica. La diferencia principal es en el tiempo de entrenamiento (Fig. XIV), pues mientras una fue capaz de analizar, clasificar y diagnosticar en solo 38 segundos, otra tomo 70 segundos, siendo esta diferencia un tanto indiferente (considerando solo 500 puntos de operación). En lo referente a la probabilidad de diagnóstico correcto, estas mostraron un comportamiento estable, tanto en el cambio de régimen, como en el aumento de número de puntos de simulación. Siendo esta probabilidad entre 81 y 83%.

AGRADECIMIENTOS

- Al Instituto Politécnico Nacional, por el apoyo económico a través del proyecto de investigación No. 20091273

REFERENCIAS

- Gutiérrez Mojica Eder Martin, 2005.-Tesis: “Diagnóstico y monitoreo de los patrones de una turbina de gas mediante redes neuronales artificiales”, Instituto Politécnico Nacional, México.
- Howard Demuth 2008, Martin Hagan, Mark Beale, *Neuronal Network Toolbox 6, Mathworks 2008.*
- Loboda I. 2001, E. L. Santiago, “Problems of gas turbine diagnostic model identification on maintenance data”, Memorias del 6° Congreso Nacional de Ingeniería Electromecánica y de Sistemas, ESIME-Zacatenco, México, 2001.
- Loboda I. 2003, Trustworthiness of gas turbine diagnosing on transient regimes. Memorias del XVIII Congreso de Instrumentación, SOMI, D.F, México, 2003.
- Manual de redes neuronales, 2005, Universidad tecnológica de Pereira, <http://ohm.utp.edu.co/neuronales>, consultado el 25 agosto 2005.