Universidad Tecnológica de la Mixteca

EVALUACIÓN DE DOS TÉCNICAS DE RECONOCIMIENTO DE PATRONES PARA SU IMPLEMENTACIÓN EN EL SIMULADOR DE PILOTAJE AUTOMÁTICO (PA-135, NM-79 CHOPPER) DE TALLER DEL STC METRO DE LA CD. DE MÉXICO

TESIS

Para obtener el grado de

Maestro en Electrónica, opción:
Sistemas Inteligentes Aplicados

Presenta:

Ignacio Arroyo Fernández

Director de Tesis:

Dr. Rosebet Miranda Luna

Huajuapan de León, Oax., Noviembre de 2013

Tesis presentada en Noviembre de 2013
ante el siguiente jurado:

Dr. Antonio Orantes Molina (Asesor)
Dr. Ricardo Pérez Águila (Sinodal)
M. S. R. C. José Antonio Moreno Espinosa (Sinodal)
Dr. José Anibal Arias Aguilar (Sinodal)
Dr. Rosebet Miranda Luna (Director)

Dedicado a:

El Inabominable No-hombre de las Nubes, ese de las estrellas fugaces, señales y demás...
A mi hijo Alan.

Agradecimientos

Mi más sincero y principal agradecimiento al no-hombre por traerme aquí y protegerme durante toda mi vida; así mismo, por guiarme hacia este nuevo camino.

Agradezco muy especialmente a mi mamá Leticia por su grandeza y cariño, por lo profundo de sus palabras cuando el final de una curva en el camino parecía cubierto de niebla o que no llegaría. Por su alegría para hacer resonar positivamente todo aquello que parece normal, pero que sólo es monótono. Un gran esfuerzo que no cuesta, pero que por él ha dado su vida. Muchas gracias por todo su apoyo.

De la misma forma, agradezco a mi papá Daniel por su valentía, su gran fuerza y su coraje para enseñarme el valor de las cosas, por sus consejos siempre sinceros e inagotables; muchas veces implícitos y muchas veces explícitos. Muchas gracias por enseñarme el verdarero valor de la verdad. Sin duda alguna, sin todo eso y más, no estaría aquí. Muchas gracias.

Agradezco con toda mi alma también a mis hermanos Marcos, Daniel y Heraclio por ser quienes son y por su apoyo total e incondicional. Son parte de mi pensamiento diario, cuando eventualmente me transporto a épocas de nuestra infancia, ya un poco lejanas en este tiempo relativo. Son parte de mí. Muchas gracias.

Gracias a mi director y sinodales de tesis por el tiempo invertido en la revisión de este trabajo, por su gran conocimineto y amabilidad así como por ser unas grandes personas. A la Dra. Tatiana Kempowsky por contestar dudas cruciales durante el desarrollo, a algunos otros investigadores externos a este trabajo (Daniel L. Chester, Lotfi A. Zadeh, *et. al.*) y que desinteresadamente compartieron sus grandes conocimineto. Muchas gracias.

Agradezco también a mi amigo el Gau (‘‘*Gordolo*’’) por recibirme en su madriguera cuando era soltero, al ‘‘*Negro*’’ (Edel) por ser mi amigo en las buenas y en las malas, al ‘‘*Wilo*’’ (José Luis) por ser también uno de mis amigos y una gran persona, a Tania por ser mi amiga sincera, hermana y una grandísima persona, a Carlos por ser un gran amigo, al ‘‘*Chambra*’’ (JG Zambrano, como es conocido ahora en el mundo indexado) por ser un gran jugador y revisor (Gracias a todos los que alguna vez fueron Pikeros), un gran padre y sobre todo un gran amigo, al Dr. Enrique Guzmán Ramírez por sus consejos siempre oportunos

e imparciales y por ser un gran amigo, a Ivonne (‘‘*Bob esponja*’’) y Carlos como amigos y compañeros de la nueva aventura, a Naye por su amistad, interés y hospedaje, a Celes, Carmen y Oly, Doña Mary (todo el equipo de limpieza del IEM) y Doña Nancy (todo el equipo de limpieza del IP). Gracias a todas esas personas que hicieron posible recordar a la UTM y a Huajuapan como unos grandes lugares para pasar la vida.

Al CONACyT ya que sin el apoyo otorgado, este trabajo no se hubiera concluido.

Abstract

The main motivation for this work arises from the need of Metro's Automatic Piloting (PA in spanish) system to be diagnosed for failures. At present, the PA is diagnosed by using a very obsolete equipment that generates high maintenance costs and low time efficiency. Due to above, it was proposed to design and to implement an Automatic Piloting System Simulator (SPAT-135 in spanish) which is a pattern-recognition-based automatic diagnosis system, programmed in a portable computer. It uses acquiring hardware to obtain the input patterns from the PA in real time. This proposed system gives working mobility required from the Metro's technical staff. The pattern recognition-based software was implemented according to decision making based on temporal computational complexity analysis and generalization tests, made for comparing two pattern recognition algorithms: *Learning Algorithm for Multivariable Data Analysis* (LAMDA, fuzzy clustering technique) and *Feedforward Backpropagation Artificial Neural Networks* (ANNs). In this work, we analyzed theoretical details about learning and, mostly, classification features of mentioned algorithms for establishing an objective evaluation procedure. Namely, two global experiments were made to evaluate plasticity: firstly, it was carried out the training one ANN for each PA module, in such way that we ensured that plasticity obtained was the best possible. It was possible by preparing an experiment in which it was a training set for both algorithms and also some hard-validation sets were obtained from the training set. The mentioned hard-validation sets were created adding different white-noise intensities to the training set. Each hard-validation set was presented to a neural network in order to detect in a progressive way the plasticity trend in such way that we ensured that plasticity obtained was the best possible. The above was not necessary for LAMDA due to its evolutive self-organizing feature. Secondly, we tested both algorithms with the major noise intensity in pattern recognition mode and then we observed that LAMDA was better classifier than ANNs in 66.66% of whole experimental pattern recognition processes. Note that differences are not too significant in this matter. Also, we calculated the temporal computational complexity for each trained algorithm and we found that ANNs was faster in a proportion of 3:1. LAMDA becomes more temporary complex as the number of learned classes grows. Nevertheless, since that in our application we considered that certainty of diagnosis is more important than speed, we have implemented a software tool based on LAMDA. It was tested *in-situ* and we obtained statistical data which are too useful to delivery a reliable diagnosis.

Resumen

La principal motivación de este trabajo surge de la necesidad de diagnosticar fallas del sistema de Pilotaje Automático del STC Metro de la Cd. de México, ya que actualmente cuenta con procesos y equipo completamente obsoletos; que desde luego, no ofrecen la flexibilidad necesaria para aumentar la calidad del servicio, tal como lo requiere el STC. Se ha propuesto diseñar un sistema de diagnóstico llamado SPAT-135, el cual tiene la principal característica de ser portátil y automatizado; montado con una computadora y hardware de tratamiento y adquisición de señales. Dicha computadora ejecutará un módulo de software para diagnosis cuyo núcleo constituye una técnica de reconocimiento de patrones. El objetivo principal de este trabajo de tesis, fue evaluar dos de ellas a efecto de implementar la que mejor se desempeñe: *Learning Algorithm for Multivariable Data Analysis* (LAMDA) o Redes Neuronales Artificiales (Entrenadas mediante retropropación). Se analizó detalladamente la base teórica sobre las fases de entrenamiento y, sobre todo, sobre la fase de clasificación de ambos algoritmos a fin de exponer claramente las diferencias entre ellos así como fundamentar los experimentos que después se llevarían a cabo. Con la finalidad de hacer una evaluación objetiva y que permitiera establecer un criterio de preferencia entre uno y otro, se hicieron dos experimentos globales en fase de clasificación: el primero fue llevado acabo realizando pruebas que garantizaran que el entrenamiento de cada clasificador fuese el mejor posible; después, se agregó ruido blanco de distinta intensidad a los conjuntos de entrada, ésto permitió llevar a ambos clasificadores a situaciones críticas, en las cuales se pudo obtener información valiosa sobre su capacidad de generalización. El segundo experimento consistió en realizar un análisis de la complejidad computacional temporal en fase de clasificación. Al término del primer experimento, se concluyó, por una diferencia no muy sustancial, que LAMDA tiene un mejor desempeño, mostrándose más acertado que las RNAs en un 66.66 % de las ocasiones. En el segundo experimento, se calcularon las funciones temporales de ambos clasificadores y se mostró tanto gráfica como analíticamente que las RNAs, son un algoritmo más eficiente en una proporción de 3:1. A partir de los resultados obtenidos, se implementó un núcleo de software de diagnóstico basado en LAMDA, el cual también ha sido sometido a pruebas finales de entrenamiento y clasificación en tiempo real. Dichas pruebas, arrojan resultados satisfactorios sobre la clasificación de patrones y generación de modelos de comportamiento del sistema de pilotaje automático. Además, esta clasificación fue analizada estadísticamente, mostrando ser lo suficientemente útil para emitir diagnósticos contundentes.

Índice general

Índice de figuras III

Índice de tablas V

1. Introducción 1

1.1. Estado del arte 2

1.1.1. Antecedentes 2

1.1.2. Detección y diagnóstico de fallas 5

1.1.3. Detección de fallas mediante LAMDA 7

1.1.4. Detección de fallas mediante redes neuronales 8

1.2. Planteamiento del problema 9

1.3. Justificación 11

1.4. Objetivos 13

1.5. Hipótesis 14

1.6. Metas 14

1.6.1. Metas generales y específicas 14

1.7. Metodología 17

2. Marco teórico **19**

2.1. Estructura de procesos supervisados 19

2.2. Funcionamiento del Metro y diagnóstico actual del Piloto Automático 21

2.3. Aprendizaje automático *(Machine Learning)* 25

2.3.1. Aprendizaje supervisado 25

2.3.2. Aprendizaje no supervisado 26

2.4. Clasificación de patrones 26

2.5. Agrupamiento de datos *(Data Clustering)* 30

2.5.1. Definición de ‘‘Clustering’’ 30

2.5.2. Objetos y atributos 31

2.5.3. Funciones discriminantes 31

2.5.4. Clusters, centros y modos 32

2.6. Algoritmos de clustering particional 32

2.7. Generalidades de clustering difuso 35

2.7.1. Conjuntos difusos 37

2.7.2. Estructuración de un algoritmo de clustering difuso 42

2.8. Redes neuronales artificiales 46

2.8.1. Modelo neuronal orgánico 47

2.8.2. Modelo neuronal artificial 48

2.8.3. Funciones de activación 50

2.8.4. Estructura de las redes neuronales 51

2.8.5. El perceptrón 53

2.9. Extracción de información 57

2.9.1. Análisis espectral por transformada de Fourier para señales no estacionarias 58

3. Consideraciones de implementación para redes neuronales de retro-propagación *(Back-Propagation neural networks)* 61

3.1. Entrenamiento de la red 62

3.2. Fundamentos de la regla delta generalizada 64

3.3. Inconvenientes de entrenamiento y arquitectura 67

3.3.1. Superficie de error 67

3.3.2. Sobre-ajuste (*overfitting*) y generalización (*pasticity*) 68

3.4. Consideraciones valiosas para el diseño (Minimización de inconvenientes) . . 69

3.4.1. Paso 1: Selección de variables 69

3.4.2. Paso 2: Recolección de datos 70

3.4.3. Paso 3: Preprocesamiento de datos 70

3.4.4. Paso 4: Construcción de los conjuntos de entrenamiento, prueba y validación 70

3.4.5. Paso 5: Dimensionamiento de la red 71

3.4.6. Paso 6: Establecer un criterio de evaluación 76

3.4.7. Paso 7: Entrenar la red 76

3.4.8. Paso 8: Implementación 78

4. Clustering difuso LAMDA 79

4.1. Particionamiento difuso 80

4.1.1. T-normas y T-conormas (o S-normas) 83

4.1.2. Conectivos híbridos linealmente compensados 86

4.1.3. Funciones de adecuación y densidades de probabilidad 88

4.2. Ausencia de información 92

4.3. Aprendizaje 93

4.4. Grados de Adecuación 94

4.4.1. Grado de Adecuación Marginal (MAD) 95

4.4.2. Grado de Adecuación Global (GAD) 96

4.5. Esquema general del algoritmo de clustering difuso LAMDA 96

4.6. Propiedades de LAMDA 98

5. Implementación de redes neuronales y resultados de la Comparación con LAMDA 101

5.1. Implementación experimental de Redes neuronales de retro-propagación . . . 101

5.2. Cajón `I` - PA/CMC 105

5.2.1. Redes neuronales artificiales 105

5.2.2. Entrenamiento y validación de LAMDA (a través de SALSA) 111

5.3. Cajón `II` - PA/CMC 115

5.3.1. Redes neuronales artificiales 115

5.3.2. Entrenamiento y validación de LAMDA (a través de SALSA) 117

5.4. Cajón `III` - PA/CMC 120

5.4.1. Redes neuronales artificiales 120

5.4.2. Entrenamiento y validación de LAMDA (a través de SALSA) 121

5.5. Cajón IV - PA/CMC 123
5.5.1. Redes neuronales artificiales 123
5.5.2. Entrenamiento y validación de LAMDA (a través de SALSA) 124
5.6. Cajón I - CML/CMR 127
5.6.1. Redes neuronales artificiales 127
5.6.2. Entrenamiento y validación de LAMDA (a través de SALSA) 129
5.7. Cajón II - CML/CMR 130
5.7.1. Redes neuronales artificiales 130
5.7.2. Entrenamiento y validación de LAMDA (a través de SALSA) 133
5.8. Complejidad computacional 133
5.8.1. Redes neuronales Artificiales 134
5.8.2. LAMDA 135
5.9. Resultados de la comparación entre los algoritmos 136
5.10. Conclusión de la comparación 139

6. Implementación y resultados **141**

6.1. Pre-procesamiento 142
6.2. Acceso/manejo de archivos CSV 144
6.3. Supervisión (LAMDA) 146
6.4. Resultados 150

7. Conclusiones y trabajo futuro **169**

A. Tabla de componentes de la GUI del NC-SPAT135 **173**

B. Código fuente para curvas de tendencia **179**

C. Código fuente para manejo de archivos **183**

D. Código fuente para procesamiento de señales **191**

E. Código fuente LAMDA **199**

Bibliografía **207**

Índice de figuras

1.1. Sistema electrónico de Pilotaje Automático de un tren del Metro. 2

1.2. Sistema de diagnóstico SPY para el sistema de PA SACEM 3

1.3. Nuevo sistema de diagnóstico SPY. 4

1.4. Interfaz gráfica del nuevo SPY. 4

1.5. Tarjeta lógica del PA 135. 5

1.6. Esquema general del Simulador PA de Taller ($SPAT$) 135kHz. 10

1.7. Contexto del software NC-SPAT 135. 11

1.8. Evolución del transporte público en México. 11

1.9. Ciclo de vida del desarrollo de software en espiral. 17

1.10. Esquema general del NC-SPAT 135. 18

2.1. Contexto de un sistema de supervisión automática. Figura tomada de [73]. . 21

2.2. Banco de pruebas que actualmente se usa en el laboratorio. 23

2.3. (a) Panel delantero y (b) panel trasero del chasis de montaje del Sistema de Pilotaje Automático. 24

2.4. Esquema de un clasificador. 27

2.5. Escala de grises mapeada al concepto de conjuntos difusos. 37

2.6. Ilustración de dos conjuntos difusos A y B 39

2.7. Ilustración de un conjunto difuso convexo en $\mathbb{R}^1$. 41

2.8. Conjunto convexo A, y su respectiva función de pertenencia $f_A(x)$. 42

2.9. Ilustración del teorema de separabilidad para conjuntos difusos en $\mathbb{R}^1$. 42

2.10. Mezcla de dos distribuciones gaussianas de probabilidad que representarían a un par de clusters. 43

2.11. Ejemplo de una superficie de error para $J_q(\Theta, U)$ en términos de $q \in \{1, ..., 6\}$. 44

2.12. Disposición simplificada de una neurona orgánica. 47

2.13. Proceso sináptico de las neuronas orgánicas. 48

2.14. Modelo artificial de una neurona. 50

2.15. Función de activación tangente hiperbólica. 51

2.16. Estructura de una red neuronal multicapa. 52

2.17. Estructura de una red neuronal feedback. 53

2.18. Estructura de un perceptrón simple. 54

2.19. Estructura de una red con n perceptrones simples. 54

2.20. Espacio de entrada bidimensional para el perceptrón simple. 55

2.21. Ejemplo de la construcción de una señal cuadrada mediante una suma de señales seno. 59

2.22. Ejemplo de la aplicación de la DFT a una señal coseno. 59

3.1. Flujo de señales a través de la red en alimentación directa. 62

3.2. Flujo del vector de error a través de la red en propagación hacia atrás (*backward*). Figura sacada de [46]. 63

3.3. Forma hipotética de la superficie de error y el problema de los mínimos locales 68

4.1. Ilustración del paradigma parmenídeo. 81

4.2. Ilustración de los elementos principales de una partición C sobre X. 83

4.3. Gráficos de las t-normas (a) Gödel o máx, (b) producto y (c)-Lukasiewicz. . 85

4.4. Ilustración de la acción básica de parametrización y manipulación de una partición. 86

4.5. Ilustración de la función de (a) densidad Gaussiana para $X \subset \mathbb{R}^1$ y (b) para $X \subset \mathbb{R}^2$. 90

4.6. Ilustración de la familia de curvas de la función de pertenencia binomial y binomial distancia. 91

4.7. Partición difusa mediante la función de adecuación Binomial 95

4.8. Fases del *Learning Algorithm for Multivariable Data Analysis* (LAMDA). . . 98

5.1. Captura de pantalla del ‘‘Neural Network Toolbox’’ de MATLAB. 102

5.2. Conjunto de entrada 28-dimensional para el entrenamiento de las trece redes neuronales de la Tabla 5.2. 107

5.3. Rendimiento durante la validación para la red neuronal del experimento 6 en la Tabla 5.2. 108

5.4. Rendimiento durante la validación para la red neuronal del quinto experimento en la Tabla 5.2. 109

5.5. Tendencia de generalización por arquitectura de red. 110

5.6. Evolución del entrenamiento de la redes 5 y 13 Cajón I-PA/CMC. 111

5.7. Desempeño de la red 13, Cajón I-PA/CMC, al tratar de seguir la función descrita por el vector de objetivos. 112

5.8. Desempeño de la red 5 al tratar de seguir la función descrita por el vector de objetivos. 113

5.9. Interfaz gráfica de usuario S. A. L. S. A. 114

5.10. Ilustración de Salida de clasificación automática y la salida de clasificación de patrones. 114

5.11. Tendencia de generalización por arquitectura de red. 116

5.12. Evolución del entrenamiento de la red 4 ($n_I = 12, n_h = 4, n_O = 1$). 117

5.13. Ilustración de (a) salida de reconocimiento de patrones de la red 4 con $a = 90$. 118

5.14. Evolución del entrenamiento de la red 13. 118

5.15. Ilustración de conjunto de entrada modificado por ruido para el Cajón II - PA/CMC ($a = 1$). 119

5.16. Ilustración de resultado de clasificación de LAMDA para el Cajón II - PA/CMC. 119

5.17. Patrones de entrada para entrenamiento, prueba y validación. 121

5.18. Curvas de tendencia de generalización. 122

5.19. Evolución del entrenamiento y reconocimiento de patrones. 122

5.20. Ilustración de resultado de clasificación de LAMDA para el Cajón III - PA/CMC. 123

5.21. Ilustración del (a) conjunto de entrada usado para entrenamiento. 124

5.22. Curvas de tendencia de generalización. 125

5.23. Evolución del entrenamiento y desempeño en el reconocimiento de patrones. 126

5.24. (a) Resultado de clasificación y (b) desempeño de reconocimiento de patrones de LAMDA. 126

5.25. Ilustración del conjunto de entrada usado para entrenamiento, prueba y validación. 127

5.26. Curvas de tendencia de generalización. 128

5.27. Evolución del entrenamiento y desempeño en el reconocimiento de patrones. 129

5.28. Ilustración del (a) resultado de clasificación y (b) desempeño de reconocimiento de patrones de LAMDA. 129

5.29. Patrones de entrada para el Cajón II CML/CMR. 131

5.30. Curvas de tendencia de generalización. 132

5.31. Evolución del entrenamiento y desempeño en el reconocimiento de patrones. 132

5.32. Ilustración del (a) resultado de clasificación y (b) desempeño de reconocimiento de patrones de LAMDA. 133

5.33. Funciones temporales máximas de LAMDA y RNAs. 137

6.1. Diseño modular del software de diagnóstico (elementos marcados con líneas continuas). 142

6.2. Diseño de la estructura de un archivo `*.csv`. 146

6.3. Ejemplo de un archivo de estados (`estados.csv`). 147

6.4. GUI experimental del NC-SPAT135. Véase cada elemento de manera detallada en el Apéndice A. 147

6.5. (a) Montaje de elementos para realizar pruebas de cajón, entrenamiento y pruebas finales del NC-SPAT 135 y (b) tarjetas electrónicas dañadas para obtener modelos de comportamiento en falla. El STC Metro, ha proporcionado una tarjeta para cada cajón, con la finalidad de caracterizar la falla más frecuente. 153

6.6. Para el Cajón I PA/CMC, prueba 3 (`tracPA-AV`): (a) Modelo de comportamiento en funcionamiento correcto, (b) modelo de comportamiento en falla. Prueba 5 (`tracCMC-AV`): (c) Modelo de comportamiento en funcionamiento correcto, (d) modelo de comportamiento en falla. Prueba 7 (`lampPNA-AV`): (e) Modelo de comportamiento en funcionamiento correcto, (f) modelo de comportamiento en falla. 162

6.7. Para el Cajón II PA/CMC, prueba 38 (`umbr255AV_OD`): (a) Modelo de comportamiento en funcionamiento correcto, (b) modelo de comportamiento en falla. Prueba 40 (`umbr197AV_OD`): (c) Modelo de comportamiento en funcionamiento correcto, (d) modelo de comportamiento en falla. Prueba 49 (`CMC_AV`): (e) Modelo de comportamiento en funcionamiento correcto, (f) modelo de comportamiento en falla. 163

6.8. Para el Cajón III PA/CMC, prueba 52 (`iniPA-AV`): (a) Modelo de comportamiento en funcionamiento correcto, (b) modelo de comportamiento en falla. Prueba 57 (`83Q-AVOD`): (c) Modelo de comportamiento en funcionamiento correcto, (d) modelo de comportamiento en falla. 164

6.9. Para el Cajón IV PA/CMC, prueba 63 (`captrsVal`): (a) Modelo de comportamiento en funcionamiento correcto, (b) modelo de comportamiento en falla. 165

6.10. Para el Cajón I CML/CMR, prueba 95 (`CML-50-AV`): (a) Modelo de comportamiento en funcionamiento correcto, (b) modelo de comportamiento en falla. Prueba 96 (`CML-50-AR`): (a) Modelo de comportamiento en funcionamiento correcto, (b) modelo de comportamiento en falla. 166

6.11. Para el Cajón II CML/CMR, prueba 79 (`CML25-AV`): (a) Modelo de comportamiento en funcionamiento correcto, (b) modelo de comportamiento en falla. Prueba 81 (`CML50-AV`): (a) Modelo de comportamiento en funcionamiento correcto, (b) modelo de comportamiento en falla. Prueba 83 (`CMR`): (a) Modelo de comportamiento en funcionamiento correcto, (b) modelo de comportamiento en falla. 167

7.1. Patrón estimado de capacidad de generalización para las redes neuronales estudiadas. 170

Índice de tablas

4.1. Operadores de t-normas y t-conormas empleados usualmente en LAMDA. . . 85

5.1. Cantidad de datos recolectados por cajón del PA. 104

5.2. Datos resultantes del entrenamiento de trece redes neuronales con diferente enfoque de dimensionamiento, para el Cajón I PA/CMC. 106

5.3. Segmentación de patrones del conjunto de entrada, Cajón I PA/CMC. . . . 107

5.4. Datos resultantes del entrenamiento de trece redes neuronales con diferente enfoque de dimensionamiento, para el Cajón II PA/CMC. 116

5.5. Datos resultantes del entrenamiento de 12 redes neuronales con diferente enfoque de dimensionamiento, para el Cajón III PA/CMC. 120

5.6. Datos resultantes del entrenamiento de 12 redes neuronales con diferente enfoque de dimensionamiento, para el Cajón IV PA/CMC. 125

5.7. Datos resultantes del entrenamiento de 12 redes neuronales con diferente enfoque de dimensionamiento, para el Cajón I CML/CMR. 128

5.8. Datos resultantes del entrenamiento de 12 redes neuronales con diferente enfoque de dimensionamiento, para el Cajón II CML/CMR. 131

5.9. Complejidad computacional para cada red neurononal. 135

5.10. Complejidad computacional para LAMDA, según N_{clases}. 138

5.11. Porcentajes de reconocimiento para LAMDA y RNAs. 138

6.1. Código de las componentes de los vectores mapa. 144

6.2. Pruebas de cajón, sus etiquetas y nombres. 155

6.3. Registro de comportamiento obtenido para el Cajón `I` - PA/CMC, con $\lambda = 0.60$. 156

6.4. Registro de comportamiento obtenido para el Cajón `II` - PA/CMC, con $\lambda = 0.60$. 157

6.5. Registro de comportamiento para el Cajón III - PA/CMC, con $\lambda = 0.60$. . . 158

6.6. Registro de comportamiento para el Cajón `IV` - PA/CMC, con $\lambda = 0.60$. . . 159

6.7. Registro de comportamiento para el Cajón `I` - CML/CMR, con $\lambda = 0.53$. . . 160

6.8. Registro de comportamiento para el Cajón `II` - CML/CMR, con $\lambda = 0.60$. . 161

Capítulo 1

Introducción

En este trabajo de tesis, se ha propuesto el diseño de un núcleo de software basado en una técnica de reconocimiento de patrones, cuya finalidad es la captura y supervisión en tiempo real de señales muestreadas de voltaje, provenientes de un sistema electrónico de Pilotaje Automático (o sólo *PA*), que opera en algún tren del parque vehicular del STC Metro de la Ciudad de México. Cada tren cuenta con su propio PA. Dicho núcleo de software será parte de una solución completa llamada *Simulador de Pilotaje Automático de Taller*, que tendrá la misión de simular eléctricamente, para el sistema de PA, el comportamiento de un tren. El resultado de dicha simulación, tiene la finalidad de proporcionar información muy importante, en términos del diagnóstico de fallas, que ayude al personal técnico de mantenimiento de los trenes a tomar decisiones que ahorren tiempo y recursos.

El PA está constituido por seis secciones[1], cada una es conocida como *cajón* (Cajón I-PA/CMC, Cajón II-PA/CMC, Cajón III-PA/CMC, Cajón IV-PA/CMC, Cajón I-CML/CMR y Cajón II-CML/CMR), de manera que cuando hay problemas en la conducción del tren, es posible que alguno de ellos haya sufrido un daño y esté fallando. Véase la Figura 1.1. Lo anterior es muy difícil de determinar en las fosas[2], por lo que fue necesario el desarrollo de un sistema con las prestaciones necesarias para integrarse al conjunto de herramientas que ahí se utilizan diariamente.

[1]Existen dos modelos de PA cuya presencia es dominante (Véase en el parque vehicular del STC[3]): el *PA-135 Chopper* y el *PA-135 JH*. Este trabajo de tesis, se concentra exclusivamente en el PA-135 Chopper (Material NM-79 Chopper). Los dos modelos están diseñados de la misma manera.

[2]Sector del taller del STC donde salen de la línea los trenes para recibir mantenimiento.

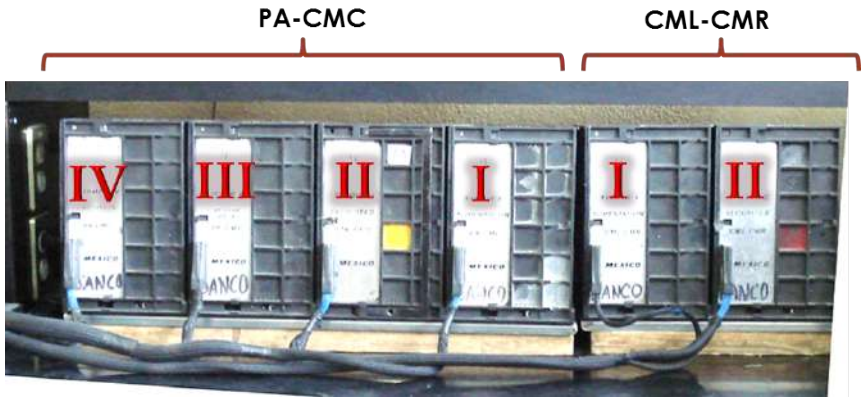


Figura 1.1: Sistema electrónico de Pilotaje Automático de un tren del Metro.

1.1. Estado del arte

1.1.1. Antecedentes

La detección de fallas en procesos industriales es actualmente un campo propenso a recibir mucha atención por lo que conlleva a un gran esfuerzo por realizar trabajos de investigación que simplifiquen la tarea de *conservación* de los elementos susceptibles de desgaste en un proceso industrial. En este ámbito, el proceso de detección de fallas es una parte de lo que en general es el *mantenimiento*, término usado después de la revolución industrial, a finales del siglo XIX y que se define como[47]:

Definición **1** *El conjunto de técnicas usadas para conservar equipos e instalaciones durante el mayor tiempo posible, buscando su máximo de disponibilidad y rendimiento.*

Lo que se ve en la Definición 1 nos conduce a que la intención de quienes están dedicados al mantenimiento de dispositivos, buscan ahorrar costos disminuyendo riesgos laborales y evitando pérdidas por paro de producción, previniendo el desgaste prematuro o fallas que aparezcan de forma inesperada, sin embargo, existen casos en los que el mantenimiento preventivo no excluye la ocurrencia de fallas, de modo que la detección eficiente (en el menor tiempo posible) de las mismas es indispensable para que el equipo observado regrese de inmediato a su puesto de producción. A esto último se le conoce como mantenimiento *correctivo* y el sistema desarrollado en este trabajo tendrá la misión de ser una herramienta para llevarlo a cabo en el Pilotaje Automático del Metro de la Cd. de México.

Nunca se ha desarrollado un trabajo que pretenda llevar a cabo la detección automática de fallas o simulación del P. A. del Metro en sus versiones PA 135 (Chopper y JH); sin embargo, es

Figura 1.2: Sistema de diagnóstico SPY para el sistema de PA SACEM

posible mencionar sistemas similares para otras versiones de pilotaje automático, tal es el caso del sistema SACEM (*Système d'Aide à la Conduite, à l'Exploitation et à la Maintenance*), que al igual que PA 135, es de diseño y fabricación en Francia por *INTERELEC, société anonyme* (absorbida hoy por SIEMENS Transportation Systems), en 1983. El SACEM fue puesto en marcha en México con los trenes de la línea A, línea 8 y línea B (en los años 1991, 1994 y 2000 respectivamente)[6]. Está constituido de un sistema de cómputo empotrado (basado en FPGAs y microprocesadores) y está provisto de un sistema de diagnóstico básico llamado *SPY* el cual se ejecuta en una computadora portátil Intel 486, con sistema operativo MS-DOS de 16 bits[7].

Según [7], SPY genera información en tiempo real sobre el estado en que se halla operando el sistema, dicha información es usada por un técnico capacitado con el fin de determinar el origen de una falla, para posteriormente determinar el procedimiento que conduzca a la corrección de la misma; así mismo, para realizar el mantenimiento preventivo correspondiente. El software se comunica con el SACEM por un medio guiado, un puerto y protocolo especializados para diagnóstico. Como se muestra en la figura 1.2, SPY permitía almacenar (en formato digital) y mostrar en pantalla 74 parámetros tales como: tipo de marcha, tipos de marcha disponibles, velocidad del tren, defectos de frenado, localización, vibradores, permiso de apertura de puertas, entre otros; todos ya sea en estado de pruebas dinámicas, estáticas o de simple monitoreo.

Posteriormente se intentó, en diferentes ocasiones, sin éxito migrar el SPY a arquitecturas de sistemas operativos más recientes (Windows XP y Vista) dado que el equipo en que residía se volvía extremadamente obsoleto. No fue sino hasta el 2010 cuando el área de proyectos

Figura 1.3: Nuevo sistema de diagnóstico SPY.

Figura 1.4: Interfaz gráfica del nuevo SPY y sus variables para diagnóstico de fallas.

del STC propone crear un software mediante la reingeniería que Sergio R. Guevara H. y Anabel Gaytán M. llevarían a cabo sobre el SPY original[7]. Este nuevo SPY (que se ve en la Figura 1.3), a diferencia del original, corre actualmente sobre Windows XP y Vista, cuenta con una interfaz gráfica basada en LabView, opcionalmente puede comunicarse de forma inalámbrica con el SACEM (aprovechando que éste ya cuenta con funciones adquisidoras de datos incorporadas) y realiza diagnósticos de fallos básicos tomando en cuenta diferentes combinaciones de los valores de las variables que adquiere (ver Figura 1.4).

Surge también la necesidad de tener facilidades similares para el mantenimiento de los sistemas PA 135 (Chopper y JH), puesto que se cuenta con ellos en la gran mayoría del STC (líneas 1 a 7), por lo que el área de proyectos del STC (en el 2011), después del proceso de adjudicación respectivo, asigna a la Universidad Tecnológica de la Mixteca la tarea de diseñar y construir un sistema de simulación y detección de fallas para el PA 135 (Véanse los detalles en la sección 1.2). Sin embargo, esto no resulta tan fácil, como en el caso del SACEM, dado que el PA 135 fue diseñado en 1975 y no cuenta con capacidad para proporcionar las facilidades mínimas necesarias. El único software presente se remonta a una memoria EPROM que alimenta código máquina a un microprocesador Motorola M68000, soldados en una placa fenólica mono-capa montada en el cajón de seguridades del PA 135 (véase la Figura

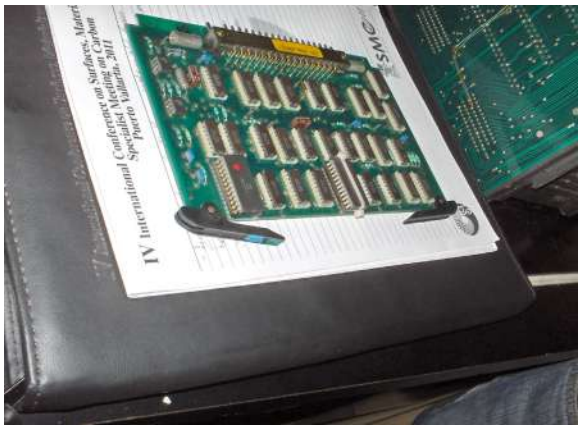


Figura 1.5: Tarjeta lógica del PA 135.

1.5). Consecuentemente se debe recurrir a técnicas de análisis mucho más complejas (ya mencionadas en la sección 1) y que aparte requieran una intervención casi nula del usuario.

1.1.2. Detección y diagnóstico de fallas

Los primeros esquemas formales para la detección de fallas se remontan a la década de los 70's (metodologías documentadas por Alan S. Willsky en 1976[21]). El método más antiguo se llama *redundancia de hardware*[20], que, como su nombre lo indica, consiste en instalar determinado número de clones redundantes, en línea, de los elementos que eran susceptibles de falla, de modo que si alguno de ellos se comportaba diferente a sus redundantes, se sabía que el elemento en cuestión se encontraba en estado de falla, y por lo tanto se tenía el tiempo necesario para hacer el mantenimiento correctivo correspondiente; este método también se conoce como *sistema de votación*. Sin embargo, era muy caro construirlos y mantenerlos, además de que los conceptos de detección y diagnóstico no se hallaban lo suficientemente definidos.

La *detección de fallas* es simplemente determinar si el sistema observado se encuentra funcionando en estado correcto o incorrecto.

El *diagnóstico de fallas* significa que una vez detectada la presencia de una falla, se procede a especificar qué tipo de falla es y cómo repararla.

Aquellos sistemas de detección de fallas estaban diseñados únicamente para tratar procesos de los cuales se tuviera un modelo matemático que emergiera de una "*formulación del problema de detección de fallas*", de modo que la detección de un fallo se basara en la discrepancia que hubiese en el funcionamiento del proceso con respecto al modelo formulado[21].

Otro enfoque de la época pionera se basaba también en modelos pero usando *filtros sensibles a fallas*[21] (conocido también como *Redundancia analítica*[24]), propuesto en 1971 por R. V. Bread del Tecnológico de Massachusetts. Estos filtros estaban diseñados y construidos para mostrar un comportamiento diferente, en caso de existir en el sistema observado un comportamiento distinto al normal; por ejemplo: si existiera un cambio abrupto en la frecuencia de vibración de un motor, el filtro reflejaba, con su comportamiento, instantáneamente dicho cambio.

Existe otra técnica que es muy usada hasta nuestros días en la detección de fallas, sobre todo en sistemas de control (sistemas dinámicos), también está basada en modelos, fue propuesta en 1984 por Isermann mediante el concepto de *espacio de paridad* que tiene sus bases en los filtros propuestos por R. V. Bread. No obstante de la limitación de estar basada en modelos, su hegemonía permanece debido a su gran flexibilidad; estamos hablando de las *Jump Process Formulations* o también llamadas *Devising System Design Methodologies*, que básicamente refieren el uso de observadores para estimar variables que ayuden a monitorear el funcionamiento del sistema. La flexibilidad que hemos mencionado se debe simplemente a que un observador se puede programar en cualquier dispositivo computacional. Una aplicación de esto la podemos hallar en [20][23][24].

Los métodos mencionados hasta el momento tienen en común la gran desventaja de ser aptos únicamente para sistemas lineales, por lo que para sistemas no lineales, es necesario hacer aproximaciones que no proporcionan la flexibilidad adecuada.

Otra metodología muy usada y muy simple es la clasificación de vectores de falla obtenidos mediante la extracción de características a través de la transformada Wavelet, midiendo únicamente la distancia euclidiana que existe entre vectores de falla (cuyas componentes son los coeficientes Wavelet) conocidos y los que se obtienen del sistema observado[22].

Otros muchos enfoques pueden hallarse en la literatura; [23] nos ofrece una revisión sintetizada de varias referencias pioneras, junto con otras que presentan propuestas que van más allá de los métodos clásicos que se han mencionado, además de superar la limitante lineal. Por mencionar las más destacadas: Redes neuronales [18][27][28], sistemas difusos, sistemas neuro-difusos[36], clustering, máquinas de soporte vectorial (SVMs)[14][37], sistemas expertos (o basados en conocimiento)[28], clasificadores Bayesianos (redes bayesianas)[38] y métodos híbridos ([38] ofrece un gran aporte experimental combinando varios enfoques, proponiendo como técnica óptima el razonamiento difuso y probabilístico). Este último es el caso de LAMDA (clustering difuso-probabilístico)[19][14].

1.1.3. Detección de fallas mediante LAMDA

LAMDA *(Learning Algorithm for Multivariable Data Analysis)* es un método de clasificación desarrollado en 1982[8] por Joseph Aguilar-Martín, Ramon Lopéz-Mántaras, Núria Piera, *et al.* Fue empleado por primera vez en [8] para reconocimiento y aglomeración de patrones de voz; estos eran relacionados con un significado propio para el algoritmo. Rápidamente sus desarrolladores dieron cuenta de la gran variedad de aplicaciones que se le podían dar al algoritmo. Así que para 1989 comienza a usarse para la extracción de características en imágenes médicas (tomografías cerebrales)[15]. Por primera vez el trabajo más destacado, en cuanto a la detección automática de fallas, usando LAMDA se publica en 1999 [9]. En éste se monitorea y diagnostica en tiempo real el funcionamiento de un sistema de tanques que contienen sustancias químicas líquidas. Este sistema ya cuenta con un sistema de control que registra y regula los niveles en los tanques accionando una bomba y un conjunto de electroválvulas. LAMDA es implementado en un sistema de cómputo que se halla interconectado con los sensores del sistema, reportando en todo momento si el funcionamiento del mismo es correcto, incorrecto o incluso si se halla propenso a fallar. A partir de aquí el uso del algoritmo en procesos industriales complejos ha sido muy variado y efectivo; tanto que, en el mismo año es usado ahora para la *supervisión* y diagnóstico del funcionamiento de una turbina de propulsión, monitorizando magnitudes tales como presión, temperatura y flujo de masa de aire[10].

Un avance muy destacado se lleva a cabo en 2003 cuando Tatiana Kempowsky[11], con ayuda de Joseph Aguilar-Martín y el financiamiento del proyecto europeo CHEM, desarrollan una herramienta de Software llamada S. A. L. S. A. (*Sitiation Assessment using LAMDA claSsification Algorithm*) que implementa LAMDA. Dicha herramienta fue usada por primera vez para supervisar y diagnosticar el funcionamiento de un reactor químico (de la Universidad Politécnica de Cataluña), el cual produce hidrógeno a partir de carbón, o a partir de residuos de madera y agua. La herramienta de software es generalizada dado que se puede usar para supervisar y diagnosticar cualquier proceso complejo, ya que únicamente lee datos desde un archivo de texto que cualquier otra aplicación externa de adquisición de datos pueda escribir (la dimensión de estos datos es ajustable en la herramienta).

Es posible diagnosticar también el estado mental de un paciente psiquiátrico, ya que LAMDA puede procesar descriptores cualitativos. Una aplicación de este tipo es la que ha propuesto en el 2004 Martha Galindo de Ariza, asesorada por Aguilar-Martín, en donde se plantea la clasificación de los trastornos de personalidad o psicopatologías (si existen o no) de una muestra de 151 personas residentes de Toulouse, Francia[12].

Actualmente LAMDA es un algoritmo muy efectivo pero poco usado aún (en comparación con las redes neuronales) debido a la escasez de información que imperaba al respecto hasta hace poco. En la Universidad Tecnológica de la Mixteca existen trabajos que emplean este algoritmo para llevar a cabo tareas de clasificación. Es el caso de A. Orantes M., quien ha

colaborado directamente con el creador de LAMDA para desarrollar una aplicación [13] que centra su atención en el acondicionamiento del sistema observado para optimizar (mediante el concepto de entropía) el monitoreo de variables, usando solamente las más significativas.

1.1.4. Detección de fallas mediante redes neuronales

Redes neuronales, cuyos antecedentes datan de los años 50's (F. Rosenblatt en 1958 con el Perceptrón[45]), es una metodología mucho más usada que LAMDA (propuesta inicialmente por J. Aguilar-Martín en 1982); así mismo, existe una amplia gama de aplicaciones que la comunidad científica ha propuesto y sigue proponiendo. Para este trabajo nos centramos en aplicaciones que tengan que ver con la detección de fallas, que por lo general se aplican a sistemas no lineales.

El primer registro que se tiene (o por lo menos, de los primeros, el más destacado) es la propuesta de L. H. Ungar y B. A. Powell en 1988[25], donde se propone el uso de RNAs (llamadas aquí "Redes adaptativas no lineales", *Non-linear Adaptive Networks*) para la detección de fallas en el sistema de control de una pequeña planta química. Aquí se argumenta de manera categórica, simple y sin lugar a dudas, la viabilidad del uso de estas redes no lineales puesto que no hay la necesidad de formular un modelo; por otro lado, no es posible aplicar una técnica de análisis lineal a la solución del problema (de hecho la red neuronal se convierte en un modelo adaptativo, de aquí el nombre que el autor refiere). Posteriormente L. H. Ungar *et al.* aplica su idea para, aparte de detectar fallas, modelar un sistema de control no lineal para la planta química[26] (no quiere esto decir que el enfoque del sistema sea basado en modelos), de modo que el monitoreo y control de dicha planta, queda completamente replanteado, con un enfoque totalmente distinto e innovador respecto del original (basado en modelos de sistemas de control clásico). Estos dos trabajos abrieron brecha en la gran autopista actual de la detección de fallas usando redes neuronales.

Vendrían después otros trabajos inspirados en [25][26] (la mayoría de ellos haciendo la cita correspondiente de manera directa o indirecta), en los cuales se resolvieron problemas de detección de fallas en sistemas dinámicos no lineales[27], detección de fallas en plantas nucleares[28] y procesos químicos[25], validación de sensores[34], detección de intrusiones (tráfico anormal) en redes de cómputo [32][31], detección de fallas en sensores[30][33][35], detección de fallas en sistemas de distribución de energía eléctrica[29][18]. Esta última aplicación es instanciada en varias partes del mundo, entregando resultados excelentes que dotan a las redes neuronales de gran robustez comprobada, haciendo que se puedan considerar como una opción de alto grado de viabilidad en la supervisión y diagnóstico de sistemas críticos. Muchos de estos experimentos han hecho un gran aporte también en el sentido de que incluso se ha comparado el desempeño de las RNAs con otras técnicas más convencionales[30] (como árboles de decisión y sistemas expertos), mostrando resultados muy satisfactorios que generan confianza para que más investigadores trabajen bajo este paradigma.

1.2. Planteamiento del problema

Como ya se ha mencionado en el capítulo 1 actualmente el laboratorio del SCT cuenta con un sistema de simulación no portátil, conocido como Banco Simulador de Pruebas, que requiere conexión con todos los cajones al mismo tiempo, de modo que se puedan recrear situaciones tales como: Parada en estación, tracción, frenado, entre otras. Lo anterior se logra activando diferentes combinaciones de botones que posee el banco de pruebas, estos estímulos generan reacciones por parte de los cajones del PA, las cuales se pueden observar en las lámparas del panel vertical de dicho banco de pruebas, esto con el fin de determinar si existe alguna falla.

A menudo el departamento de *equipo embarcado* recibe reportes de fallas referentes al Piloto Automático (PA) de los trenes, regularmente dichos fallos se presentan en uno o más cajones específicos de los seis que forman parte del PA. En muchas ocasiones, no existe falla y, sin embargo, el sistema completo debe ser trasladado al laboratorio para su diagnóstico.

La complejidad y cantidad de las señales que se requiere analizar en cada prueba son variadas, existen simples voltajes de CD con valores fijos para alimentación o mandos, voltajes con forma de onda seno, hasta señales cuadradas de amplitud variable por periodo. Según sea la naturaleza de la prueba, será necesario medir y generar diferentes combinaciones de estas señales.

Por lo anterior y dada la solicitud del Sistema de Transporte Colectivo Metro, se propone aumentar la eficiencia de las operaciones de mantenimiento mediante el diseño e implementación de una herramienta portátil de diagnóstico de fallas del piloto automático. Dicha herramienta, es el *Simulador PA de Taller 135kHz (JH y CHOPPER)*, el cual deberá ser capaz de verificar las principales funciones y parámetros de cada uno de los cajones que lo conforman, en los diferentes modos de conducción (PA, CMC, CML y CMR), así como emitir un diagnóstico de su estado de funcionamiento.

El Simulador PA de Taller 135kHz (*SPAT 135kHz*) será un sistema totalmente automatizado, tendrá la finalidad de probar un cajón a la vez, requerirá una mínima intervención del personal y deberá constar de un equipo de cómputo y dispositivos periféricos capaces en conjunto de medir corriente, voltaje y frecuencia(Ver Figura 1.6).

El SPAT 135kHz deberá generar las señales necesarias (como lo hace actualmente el banco de pruebas) que emularán, ante los módulos, el comportamiento del tren. Las señales principales son:

1. Programa B2.
2. Señal RPH (de la rueda fónica).

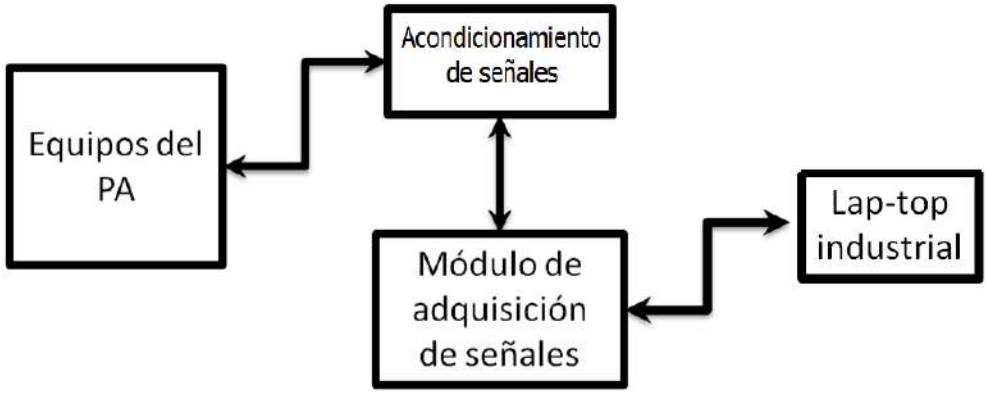


Figura 1.6: Esquema general del Simulador PA de Taller (*SPAT*) 135kHz.

3. Señal TMH (del transmisor del cronotaquígrafo).
4. Los tiempos cronometrados (de los modos PA, CMC, CML y CMR).
5. Alimentación de 72VDC, +/-24VDC, +/-12VDC y 5VDC.
6. Corriente de la señal P.
7. Cruzamientos del Chevron lógico.

Se ha expuesto de manera sintetizada en qué consiste el sistema completo (el *SPAT 135kHz*), de modo que se pueda familiarizar al lector con el contexto general de la problemática que se busca resolver con su desarrollo. Por otro lado (lo más importante), este trabajo de tesis limita su alcance al desarrollo de una parte específica de dicho sistema, misma que consiste de un módulo de software cuya implementación estará condicionada por el estudio y evaluación de dos técnicas de reconocimiento de patrones: *LAMDA* y *Back Propagation Artificial Neural Networks*.

Como se ve en la Figura 1.7, el software desarrollado en este trabajo de tesis captura desde el cajón del PA que se desea diagnosticar (a través de una tarjeta de adquisición) un conjunto de datos normalizados que son usados para llevar a cabo las pruebas de diagnóstico requeridas. El personal puede verificar en la pantalla del equipo de cómputo[3], ya sea que el funcionamiento del cajón conectado es correcto o que sea incorrecto.

Este núcleo de software proveerá la información necesaria a la interfaz del sistema de modo que sea posible que esta última esté habilitada para proporcionar un resumen impreso

[3]La verificación se hará formalmente a través de una interfaz gráfica cuyo desarrollo será llevado a cabo independientemente de este trabajo de tesis.

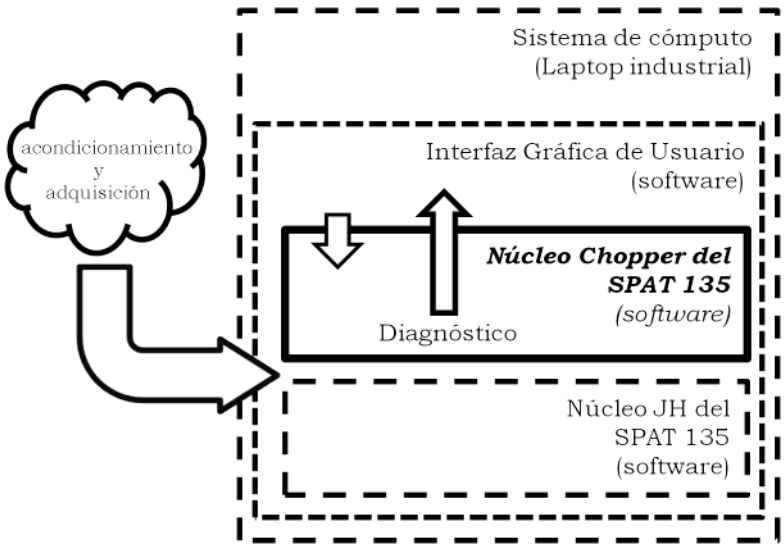


Figura 1.7: Contexto del software NC-SPAT 135.

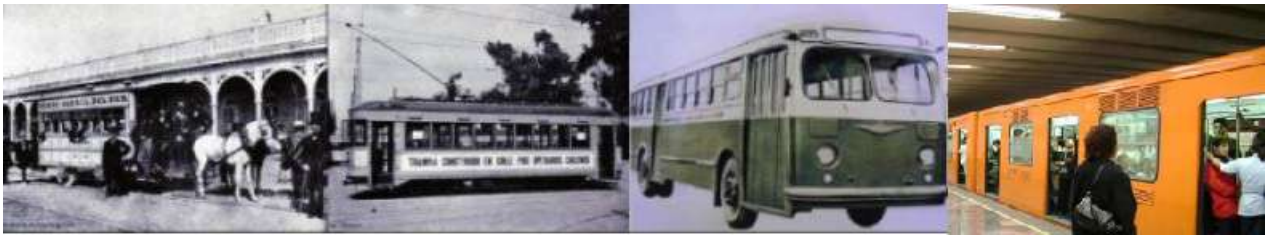


Figura 1.8: Evolución del transporte público en México.

del diagnóstico correspondiente. Dicho resumen tendrá la posibilidad de ser almacenado de manera electrónica (USB, CD, etc.) para su consulta posterior.

1.3. Justificación

Uno de los principales medios de comunicación para el ser humano es el transporte. La mayoría de las personas usan diferentes tipos de transporte público para trasladarse desde su hogar a su trabajo o a cualquier destino y viceversa. A lo largo de la historia del país han existido diversas formas de cumplir con estos objetivos: desde un caballo, una carreta, el tren hasta un microbús, un taxi o el metro. A medida que la población crece con el paso del tiempo, las ciudades también lo hacen y el transporte público toma cada vez mayor importancia, así mismo, evoluciona. Véase Figura 1.8.

Actualmente en la Cd. de México existe una población de casi 9 millones de habitantes[1], de los cuales sólo el 5.5 % se consideran como de clase alta (muy probablemente no usen el

transporte público) y el 94.5 % son de clase media y clase baja[5]. De los 8.5 millones (94.5 % de la población) que se considera que podrían usar el transporte público, el 52.9 % usan el sistema de transporte colectivo metro, lo cual significa una cantidad de 4.5 millones de habitantes[2] (el 50 % de la población total del D. F.) que este sistema de transporte público moviliza diariamente.

Nótese por lo tanto, el impacto que este sistema de transporte público tiene en la mayoría de los aspectos de la Cd. de México. Si se piensa simplemente en qué pasaría si el metro de la ciudad de México deja de funcionar repentinamente en el corto plazo, la respuesta precisa sería difícil de exponer y no está en el alcance de este trabajo averiguarlo; sin embargo, se puede suponer rápidamente y con gran certeza que no sólo el 52.9 % de la población de la Cd. de México no estaría trabajando mínimamente por un par de días, sino que la actividad económica que depende de estas personas que no se pueden transportar, también tendría pérdidas incuantificables.

Lo anterior aunado con el tipo de tecnología que se usa en el PA del Metro (de hace más de 40 años), coloca al Sistema de Transporte Colectivo Metro en una situación crítica dado que si bien se sabe que las piezas de reemplazo se vuelven cada vez más escasas para el PA, así también sucede con los propios bancos de trabajo (sujetos a mayor manipulación y desgaste) que se usan para probar presuntos equipos averiados.

Es aún más crítica la situación de un sistema de PA en particular cuando se hace un diagnóstico equivocado acerca del estado de uno o todos los cajones que lo conforman y debido a esto, es necesario desmontar el sistema completo para trasladarlo al laboratorio con el fin de que sea reparado. Muchas de las veces (según los expertos) el PA no es el causante de la falla que se manifiesta en el tren; así que se ha hecho mucho trabajo en vano que a su vez implica una gran pérdida de tiempo, pues desde que un técnico de campo determina erróneamente que el PA está fallando, hasta que éste se revisa en el laboratorio (habiendo pasado ya por una cola de espera) y se manda de regreso sin falla localizada, se han consumido un par de semanas.

Por otro lado, el equipo que se ha retirado de su puesto de trabajo en el tren ya sea con fines de prueba o por un mal diagnóstico se debe reconectar, lo cual desgasta o puede llegar causar una avería en los puertos de conexión. Estos últimos son un caso muy particular, ya que se encuentran por demás escasos, al grado de que hay puertos de conexión averiados que se han tenido que reutilizar con adaptaciones prácticamente artesanales. En otros casos menos fortuitos, el equipo queda inutilizable aunque funcione.

Las técnicas de reconocimiento de patrones que se proponen para este trabajo de tesis, son susceptibles de considerarse cuando se trata de volver eficiente o eficaz algún proceso operativo como el mantenimiento preventivo, pues pueden darle al usuario argumentos para diagnosticar una situación de falla, incluso antes de que eśta ocurra, previniendo situaciones que vulneren la seguridad de los pasajeros. Además, cuentan con un gran soporte teórico y

pruebas de validación [18] [19] [20] en el rubro del diagnóstico de fallas, lo que las hace muy adecuadas como propuestas de análisis.

Expuesto lo anterior, damos cuenta de que resulta imperativo migrar a un sistema de diagnóstico robusto y de tecnología moderna cuyas partes de reemplazo sean requeridas con muy poca frecuencia y que aún así, sean de fácil acceso y de alta disponibilidad en el mercado nacional.

La anterior justificación ha sido planteada desde un punto de vista de impacto social y operativo; sin embargo, puesto que este trabajo de tesis es un documento de investigación científica, la propuesta sobre la evaluación de dos técnicas de reconocimiento de patrones plantea una gran espectativa, y se justifica en el sentido de que no existe un estudio previo en el que se compare, en particular, a RNAs con LAMDA. De manera que los resultados obtenidos, serán de mucha utilidad en caso de que sea requerida información al respecto. Por otro lado, tampoco existen trabajos en los que sean aplicados este tipo de algoritmos en el contexto del Sistema de Transporte Colectivo Metro de la Cd. de México, lo que en efecto, aporta información muy importante, dado que fue obtenida durante la experimentación en un contexto (como ya se vio) de alto impacto social y de altos requerimientos; en el cual se encuentra como pieza fundamental el sistema de Pilotaje Automático.

1.4. Objetivos

Objetivo general

El objetivo general de este trabajo consiste en implementar un sistema de software de detección y predicción de fallas, tales que pudieran estarse presentando en el PA 135 Chopper (NM-79), basado en técnicas de reconocimiento de patrones; así mismo integrarlo a un sistema de adquisición (Hardware), de manera que permita llevar a cabo un diagnóstico (mediante pruebas sucesivas) del funcionamiento del cajón que sea conectado al sistema, mostrando de manera clara, según la información proporcionada por el experto, los defectos (si existen) que presenta cada prueba y, consecuentemente, cada cajón.

Para alcanzar el objetivo general del trabajo, es necesario dividirlo; así que se plantean los objetivos específicos que se describen a continuación:

- **Objetivo específico 1.** Realizar la implementación de la herramienta de diagnóstico con fines de prueba, basada en Redes Neuronales Artificiales usando MATLAB.
- **Objetivo específico 2.** Realizar una adaptación y prueba de diagnóstico con la

herramienta SALSA, basada en LAMDA, la cual fue desarrollada por el grupo DISCO del Laboratoire d'Analyse et d'Architecture des Systèmes de Toulouse, Francia.

- **Objetivo específico 3.** Realizar el entrenamiento de la herramienta sujeto de pruebas con los cajones de un sistema de PA cuyo funcionamiento sea correcto, de modo que se obtenga un modelo de comportamiento de estado normal (sin fallas).
- **Objetivo específico 4.** Realizar el entrenamiento de la herramienta sujeto de pruebas conectando cajones de un sistema de PA cuyo funcionamiento sea incorrecto, y cuyas fallas sean conocidas, o bien simulando la ausencia o presencia de señales que induzcan estas fallas conocidas, de modo que se obtengan diferentes modelos de comportamiento en estado de avería.
- **Objetivo específico 5.** Decidir cuál es la técnica que mejor se ajusta a nuestra aplicación (con mejor rendimiento) e implementarla en un programa de computadora escrito en la plataforma de desarrollo NI Developer Suite (LabWindows/CVI).
- **Objetivo específico 6.** Realizar el entrenamiento del software implementado, de forma que se obtengan los diferentes modelos de comportamiento definitivos.
- **Objetivo específico 7.** Realizar las pruebas en sitio del sistema desarrollado para verificar su funcionamiento según los requerimientos del usuario.

1.5. Hipótesis

1. Es posible diagnosticar de manera eficiente la operación del Sistema de Pilotaje Automático del Metro mediante el diseño e integración de un sistema de detección y predicción de fallas, basado en algoritmos de reconocimiento de patrones.
2. En comparación con las RNAs, el algoritmo LAMDA posee una capacidad ilimitada (limitada únicamente por el hardware) de aprendizaje y adaptabilidad, así mismo la cantidad y complejidad de las operaciones que realiza es menor; por lo tanto, se considera que figura como una mejor opción para implementarse.

1.6. Metas

1.6.1. Metas generales y específicas

El desarrollo del proyecto que respalda este trabajo fue dividido en 8 etapas generales (mismas que veremos en los apartados siguientes), que a su vez estarían conformadas cada una

por metas específicas muy claras que, mediante su puntual conclusión, llevaron al objetivo general planteado.

Estudio de los aspectos generales del P. A.

1. Estudiar los aspectos generales de operación del PA en el tren.

2. Estudio de trabajos relacionados que contribuyan a familiarizarse con el sistema.

3. Conocer las características de las señales que circulan por las interfaces del sistema de PA; así mismo, determinar qué relevancia podría tener cada una al formar parte de diferentes modelos de comportamiento que contribuyan con el desarrollo del NC-SPAT 135.

Aspectos generales sobre diagnóstico automático de fallas

1. Revisar el estado del arte de las técnicas que se usan para el diagnóstico automático de fallas.

2. Revisar el estado del arte de las aplicaciones más relevantes para la detección y/o predicción de fallas, basadas en técnicas de reconocimiento de patrones.

3. Plantear la estructura de un procedimiento destinado para el diagnóstico de fallas en el sistema de PA del Metro.

Implementación de una herramienta experimental de dianóstico

1. Realizar la implementación en MATLAB de una herramienta experimental de diagnóstico, basada en Redes Neuronales Artificiales entrenadas por retro-propagación.

2. Entrenar a esta herramienta con el conjunto de datos que represente a diferentes modelos de comportamiento en estado de funcionamiento correcto.

3. Entrenar a esta herramienta con el conjunto de datos que represente a diferentes modelos de comportamiento en estado de funcionamiento incorrecto; esto mediante la inducción de fallas conocidas.

Uso experimental de una herramienta existente que implemente LAMDA (SALSA)

1. Adaptar los datos adquiridos de un sistema de PA, de modo que puedan ser procesados por la herramienta SALSA, basada en LAMDA.

2. Entrenar a esta herramienta con el conjunto de datos que represente a diferentes modelos de comportamiento en estado de funcionamiento correcto.

3. Entrenar a esta herramienta con el conjunto de datos que represente a diferentes modelos de comportamiento en estado de funcionamiento incorrecto; esto mediante la inducción de fallas conocidas.

La decisión: Comparación de resultados experimentales

1. Obtener datos sobre el desempeño de las herramientas sujetos de experimentación, sobre los cuales se puedan aplicar medidas de rendimiento.

2. Realizar comparativos que determinen a una de estas herramientas como la que posee la técnica más adecuada para ser integrada en el SPAT 135kHz.

Implementación del núcleo de software NC-SPAT 135

1. Llevar a cabo la implementación de la técnica más adecuada (LAMDA o RNAs) sobre la plataforma de desarrollo NI Developer suite 2012 DS2 (LabWindows/CVI).

2. Definir un formato en que el software adquirirá los datos que va a procesar.

3. Desarrollo de una interfaz (a modo de prototipo de pruebas) con la funcionalidad suficiente, esto último con el fin de que, posteriormente, el núcleo de software desarrollado pueda recibir la inserción de una interfaz final por parte del laboratorio de usabilidad.

Entrenamiento y pruebas de laboratorio del NC-SPAT 135

1. Entrenar a esta herramienta con el conjunto de datos que represente a diferentes modelos de comportamiento en estado de funcionamiento correcto.

2. Llevar a cabo las pruebas respectivas de verificación del entrenamiento.

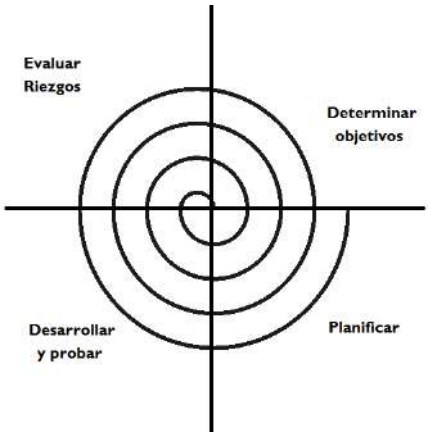


Figura 1.9: Ciclo de vida del desarrollo de software en espiral.

3. Entrenar a esta herramienta con el conjunto de datos que represente a diferentes modelos de comportamiento en estado de funcionamiento incorrecto; esto mediante la inducción de fallas conocidas.

4. Llevar a cabo las pruebas respectivas de verificación del entrenamiento.

Pruebas finales en sitio

1. Probar y validar el comportamiento del sistema de sotware desarrollado en diferentes situaciones reales para las cuales fue diseñado.

1.7. Metodología

En base a lo planteado por el objetivo general, se llevará a cabo un diagnóstico mediante una prueba del funcionamiento (validada por el NC-SPAT 135) del cajón que sea conectado al SPAT 135kHz, dicha prueba consistirá de una serie de rutinas bien definidas de generación y medición de señales, que desde el punto de vista del cajón en cuestión, cumplirán el papel de estímulos y reacciones respectivamente. Estas funcionalidades estarán dirigidas por el software que se trata en este trabajo, por lo que se propone seguir la metodología en espiral para su desarrollo y ciclo de vida.

En la Figura 1.9 se muestran las etapas del ciclo de vida de un software desarrollado mediante la metodología de espiral.

Una vez definida la metodología de desarrollo, veamos cómo estaría constituido el sistema

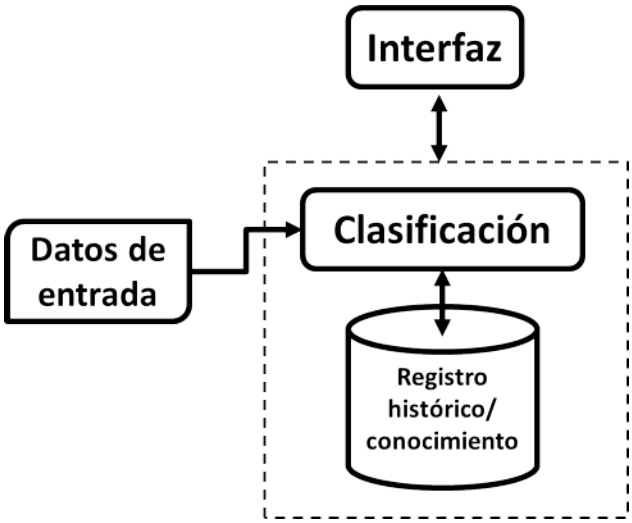


Figura 1.10: Esquema general del NC-SPAT 135.

de software a desarrollar. En la Figura 1.10 se observan claramente un esquema general y los módulos que lo componen, mismos que a continuación describimos de manera individual:

- **Reconocimiento.** Para un sistema automatizado, la mejor forma de entender cómo funciona un sujeto de tratamiento es clasificar los diferentes conjuntos de señales que caracterizan su comportamiento. La sección de clasificación del sistema tratado en este trabajo llevará a cabo la funcionalidad de clasificar conjuntos de muestras de cada señal relevantes para una rutina de simulación en particular. Dichas muestras se hallarán disponibles en un archivo escrito por el software de una tarjeta de adquisición de datos conectada al equipo de cómputo (que ejecuta el sistema). De este modo se pueden obtener, y por lo tanto almacenar, modelos de comportamiento; decimos entonces que el sistema aprende o que ganó conocimiento. Este conocimiento nuevo se almacena en una sección de registro histórico de clases y podrá ser usado posteriormente para asociarse con entradas de datos desconocidos y emitir un diagnóstico.
- **Registro histórico.** Esta sección es la unidad de almacenamiento del sistema y consiste en diferentes archivos (localizados en el disco duro del equipo que ejecuta el sistema) que serán cargados en RAM según la demanda que la sección de clasificación indique, lo cual dependerá de las rutinas de simulación que se estén ejecutando y si éstas se están llevando a cabo para un cajón del PA en particular.

Capítulo 2

Marco teórico

2.1. Estructura de procesos supervisados

En esta sección vamos a considerar de manera general cómo estaría dispuesto un proceso industrial y qué características tiene un proceso que es factible de supervisar mediante la implementación de un programa de computadora, basado en reconocimiento de patrones.

En muchos casos se es posible utilizar técnicas de control clásico para efectuar la monitorización de variables que están presentes en un proceso industrial; por ejemplo, niveles de tanques, velocidad de motores, posición angular de una electroválvula, temperatura o flujo; no obstante, el control clásico es eficiente cuando se cuenta con referencias de estas variables a las cuales hay que seguir de manera independiente. Sin embargo, el escenario deja de ser tan sencillo cuando se desea supervisar un proceso completo y éste tiene además niveles de complejidad que provocan que no sea posible aplicar una ley de control dada la cantidad, variabilidad y lo impredecible que pudieran llegar a ser tales variables. Para casos de este tipo es sumamente complicado establecer un modelo matemático que describa fielmente el comportamiento de las señales que se producen en el proceso industrial, de modo que se han diseñado aplicaciones basadas en reconocimiento de patrones que son capaces de tratar al proceso industrial como una caja negra.

Para fines de estar en condiciones de aplicar cualquiera de estos algoritmos (LAMDA y Redes Neuronales para el caso de este trabajo de tesis), es necesario contar con conocimiento específico sobre el proceso que se desea supervisar. Por lo pronto, el proceso se puede decidir en dos partes sobre las que se busca obtener medidas: el proceso y el producto. Habrá que darse a la tarea de interactuar de manera estrecha con el experto humano encargado de que se lleve a cabo la transformación exitosa de materiales de entrada a productos de calidad aceptable. Lo cual ayudará al desarrollador de la aplicación de *supervisión automática* basada

en reconocimiento de patrones a entender el problema que deberá resolver.

Como resultado de dicha interacción, es necesario tomar en cuenta la recolección de información para obtener el conocimiento necesario que ayude al desarrollador a comprender el contexto del problema a resolver[73]:

1. Conocimiento de la calidad del producto.
2. Conocimiento sobre el estado de funcionamiento del proceso.
3. Conocimiento sobre las posibles acciones a efectuar para obtener la calidad deseada.
4. Conocimiento sobre las causas de malos funcionamientos y diagnóstico.

Se tienen además, dos tipos de conocimiento de carácter más particular que el desarrollador deberá adquirir sobre un proceso industrial, dicho conocimiento fue útil (en complemento con los tipos de conocimiento más generales previamente mencionados) para estructurar el diseño del programa de computadora que se encargará de la supervisión del proceso; en este caso, el proceso es la interacción del Piloto Automático con el tren que conduce.

1. Conocimiento estructural.
    - Experimental: conocimiento sobre la estructura funcional del sistema y su descomposición en unidades elementales así como sus interconexiones.
    - Comportamental: conocimiento sobre los eventos que describen las fases del proceso y la calidad del producto.
2. Conocimiento operacional.
    - Causal: conocimiento sobre las relaciones causales entre variables y/o eventos producidos en el proceso.
    - Experimental: conocimiento sobre las acciones de un operador experimentado para solventar problemas.

En la Sección 2.2, se tiene detalladamente el resumen de la información que comprende al conocimiento necesario para el diseño del sistema de supervisión automática respectivo. Se considera que un sistema de supervisión comprende tanto la medición de variables, el reconocimiento de patrones para el diagnóstico y las acciones necesarias para mantener una referencia de calidad dada para el producto de salida (véase la Figura 2.1); sin embargo, puesto que el proceso que se va a supervisar no se encontrará en producción, las acciones requeridas para el mantenimiento correctivo se dejan como responsabilidad del personal experto; es decir, no se tiene a cargo diseñar un robot que cambie un componente dañado en

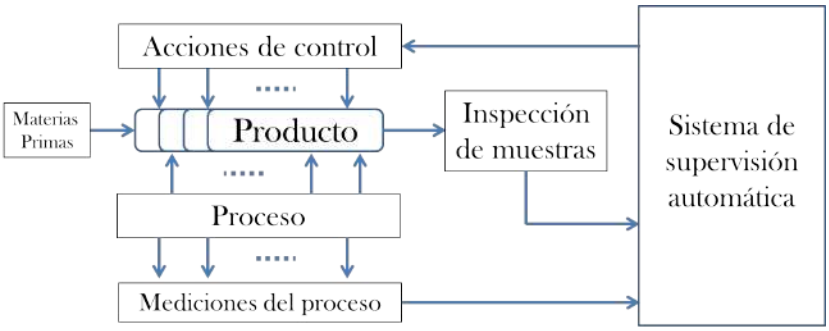


Figura 2.1: Contexto de un sistema de supervisión automática. Figura tomada de [73].

el PA, sino que deberá diseñarse un programa de computadora que le indique al experto en reparaciones qué falla se ha producido, para que éste tome las acciones necesarias. Obsérvese que en las formas de conocimiento anteriores, uno se puede imaginar infinidad de situaciones que no es posible modelar mediante una ecuación matemática, que sea procesada por una ley de control a efecto de corregir posibles desvíos durante el proceso, que pudieran ser perniciosos para la calidad en la salida del proceso.

2.2. Funcionamiento del Metro y diagnóstico actual del Piloto Automático

El transporte colectivo metro está constituido por trenes equipados con un sistema de pilotaje automático (PA) diseñado modularmente, el cual se encarga de ejecutar diferentes programas de navegación, mismos que hacen operar de maneras variadas al tren según la situación en la que éste se encuentre; por ejemplo: regular la velocidad del tren a un valor constante cuando la vía está seca y a otro valor diferente cuando está mojada, acelerar, desacelerar, abrir, cerrar puertas sólo en casos estrictamente definidos, realizar cabios de velocidad para mantener distancias prudentes entre trenes que circulan en una dirección de la línea, etc. El PA recibe un conjunto de instrucciones que se denominan *programa* a través de antenas de radio frecuencia (RF) que se encuentran colocadas en forma de cableado a lo largo de la vía, dicho cableado se denomina *tapiz* y emite portadoras en frecuencias cercanas a los 135KHz. La señal de radio es recibida en el tren, también mediante antenas de RF llamadas *captores* que envían la RF a través de cable coaxial hacia los módulos del PA encargados de desmodular, decodificar e interpretar el programa.

Existen cuatro *modos de conducción* que ejecuta el sistema de PA en base a la interpretación que se realice del programa: Piloto Automático (PA), Conducción Manual Controlada (CMC), Conducción Manual Limitada (CML) y Conducción Manual Restringida (CMR). Cada modo, en el orden en que se mencionan, tiene mayores o menores restricciones y se

activan en distintas situaciones. El modo PA se encuentra seleccionado de manera habitual, significa que el sistema de pilotaje automático se encarga totalmente de la conducción del tren, lo cual indica que no existen situaciones anormales tanto en el tren como en su entorno. El modo CMC se activa cuando existe una falla en el sistema (en el tren, en su entorno o en el modo de PA) impidiendo que el modo PA funcione de manera adecuada, así el tren puede ser conducido por el operador de manera controlada usando el programa. En el modo CML se agregan restricciones a la conducción asistida con respecto al modo CMC. Estando seleccionado el modo CML las condiciones en las que el operador debe conducir el tren no son tan favorables como para usar el modo CMC; por ejemplo, durante operaciones de mantenimiento dentro de un taller o una caída parcial del sistema de PA. Aquí las restricciones tienen que ver, principalmente, con la limitación de la velocidad, que por seguridad se establece, además de que el operador deberá pedir autorización antes de tomar cualquier acción. En el modo CMR las restricciones que tiene el sistema de PA y el operador son casi totales, no opera ninguna función automática del sistema de PA (el control se transfiere totalmente y de manera restringida al operador del tren), la velocidad del tren se restringe a la tracción mínima posible y el operador deberá llevar a cabo de manera manual cualquier operación; tal vez ante una caída total del sistema de PA. El modo PA es el único que puede ser usado si el tren está en servicio normal.

El STC (*Sistema de Transporte Colectivo*) metro cuenta con diferentes modelos de sistemas de pilotaje automático que han sido adquiridos en etapas subsecuentes del crecimiento de la red de transporte. La finalidad que tienen estos diferentes modelos en el tren es la misma que ya hemos mencionado al principio, sin embargo tienen diferencias en cuanto a que están diseñados para funcionar en trenes de distintas series: *JH* (control de velocidad electromecánico), *Chopper* (control de velocidad electrónico con tiristores), *SACEM* (control de velocidad electrónico con IGBTs); siendo el último el más reciente. La atención de este trabajo se enfoca en el piloto automático que funciona en trenes modelo Chopper NM-79 cuya población en la red es de 58 trenes (de 9 carros) mismos que representan un 22 % de los 258 trenes que deben estar de servicio en la red (el inventario total es de 355). Estos trenes circulan por las líneas 1, 3, 7, 8 y 9. A este sistema de PA se le conoce como *material NM-79 Chopper* (al sistema JH se le conoce como *material MP-68 JH*) que por comodidad sólo llamaremos "PA 135 versión Chopper'', haciendo lo mismo para el JH. Puede consultarse [3] para más detalles sobre el parque vehicular del STC metro.

El PA 135 versión Chopper (también la versión JH) está constituido por dos secciones que se denominan *módulos de interpretación*, mismos que a su vez están conformados cada uno por *cajones* que tienen diferentes finalidades específicas. A saber:

- **Módulo de interpretación PA-CMC.**

  1. Cajón I de Alimentación.
  2. Cajón II de Seguridades.

3. Cajón III de Cable de energía.
4. Cajón IV de Captación.

- **Módulo de interpretación CML-CMR.**

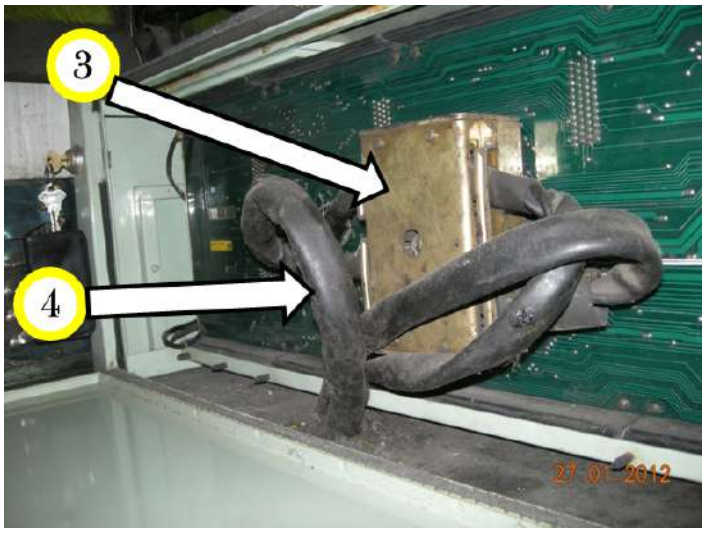
1. Cajón I de Alimentación.
2. Cajón II de Seguridades.

Actualmente el pilotaje automático recibe mantenimiento correctivo usando un equipo llamado *banco simulador de pruebas*, cuya característica principal es la obsolescencia (diseñado en los años 70's) puesto que es demasiado pesado y voluminoso. Véase la Figura 2.2. El banco de pruebas intercambia señales con el sistema de PA sujeto de pruebas, usando su puerto principal (conocido como conector ''*meduza*'', ver Figura 2.3a) y sus puertos de pruebas (conectores DB-25 en la parte frontal de cada cajón, ver Figura 2.3b), además todo el PA debe estar conectado (los 6 cajones) para que sea posible realizar las pruebas necesarias (rutinas de prueba descritas en el manual del banco simulador[4]), que consisten en simular señales que el tren genera y recibe mientras circula y se presentan situaciones específicas provistas en el programa. En dicha simulación, se asume que si alguna rutina está diseñada para verificar el funcionamiento de un cajón en particular, los demás cajones están funcionando correctamente, lo cual permite aislar una falla a nivel cajón.

Figura 2.2: Banco de pruebas que actualmente se usa en el laboratorio.

Como se observa, el PA es un sistema complejo que controla operaciones cruciales para la seguridad de los pasajeros y por lo tanto, es muy importante que se mantenga en buen estado operativo proporcionándole servicio de mantenimiento preventivo y correctivo de calidad en el mínimo tiempo posible, lo cual requiere gran flexibilidad. El sistema con el que se cuenta actualmente (Banco Simulador) ha demostrado ser a lo largo de algunas de décadas de servicio, su gran robustez; sin embargo, no es trasportable en lo absoluto, lo que lo hace sumamente limitado, confinando su uso para situaciones poco diversas.

Para lograr la flexibilidad deseada y dada la naturaleza del sistema se ha planteado usar técnicas de reconocimiento de patrones programadas en una computadora, para monitorizar



(a)

(b)

Figura 2.3: (a) Panel delatero y (b) panel trasero del chasis de montaje del Sistema de Pilotaje Automático. (1) Conector de puntos de prueba DB-25, (2) bus de cables de puntos de prueba provenientes del Banco Simulador de pruebas, (3) conector meduza y (4) bus de cableado proveniente del sistema eléctrico del tren (o del Banco Simulador en caso de que el PA se encuentre en el taller y sea objeto de pruebas). Los conectores de puntos de pruebas, sólo son usados cuando el PA se halla como sujeto de pruebas con el Banco Simulador. En el tren no se usan.

señales específicas que el técnico experto en la reparación del PA ha mencionado como de especial importancia, con el fin de detectar una o más fallas que presente el sistema en alguno o algunos cajones, teniendo la certeza de que no se trasladará al laboratorio equipo que no está fallando, evitando así daños[1] innecesarios que pudieran generarse durante el traslado desde un taller no equipado.

La decisión sobre el algoritmo de reconocimiento de patrones que habrá de usarse fue tomada a partir de la evaluación y comparación de dos de ellos:

1. LAMDA *(Learning Algorithm for Multivariable Data Analysis).*
2. Redes Neuronales Artificiales de retro-propagación (''*Back Propagation Neural Networks*'' o Perceptrón multicapa: MLP).

2.3. Aprendizaje automático *(Machine Learning)*

El concepto de aprendizaje automático se encuentra siempre inherente como fase de las diferentes categorías de algoritmos de reconocimiento de patrones que existen. Se tiene a continuación su definición[50]:

Aprendizaje automático (*machine learning*):

Definición **2** *Se dice que una máquina aprende de una experiencia E con respecto de una clase de tarea T y una medida de desempeño P, si su desempeño en tales tareas T mediante la experiencia E, es mejorado según la medida P.*

2.3.1. Aprendizaje supervisado

Este tipo de aprendizaje se deriva de una idea simple: preguntarnos qué sucede cuando un ser humano está aprendiendo a leer. Simplemente relaciona los símbolos (letras), las combinaciones de ellos y hace combinaciones de mayor nivel para formase una idea, un significado, una abstracción. Pero alguien (un profesor) le debe ayudar a hacer asociaciones,

[1]El personal del laboratorio de mantenimiento, señala que el personal que hace los traslados no se encuentra capacitado debidamente en cuanto a los cuidados que debe recibir un equipo electrónico delicado, por lo que de manera frecuente, durante el traslado, el PA recibe golpes y es tratado sin las medidas de precaución necesarias, ocasionando fallas que muy probablemente no tenía el equipo antes de ser sustraido del tren.

corrigiendo errores para que después el alumno haga dichas asociaciones sin ayuda, puesto que ya aprendió. Entonces en el *aprendizaje por corrección de error*, por ejemplo en una red neuronal (y muy similarmente para otro tipo de clasificadores donde la organización del conocimiento no es mediante pesos sinápticos), se presenta un patrón de entrada $\boldsymbol{x}^{(i)}$, la red calcula una salida $\boldsymbol{y}_i$ en función de una matriz de pesos sinápticos iniciales W_i (desajustados), dicha salida se compara con una salida deseada $\boldsymbol{t}_i$, la comparación da como resultado un vector de error $\boldsymbol{e}_i$, el cual se usa para ajustar la matriz de pesos de modo que se obtengan sus valores reajustados: W_{i+1}. El proceso se repite de manera iterativa con objeto de minimizar el error $\boldsymbol{e}_i$. Existen otras variantes del aprendizaje supervisado, pero todas toman en cuenta el mismo principio. Tal es el caso del *aprendizaje por reforzamiento*, el cual aumenta la magnitud de los pesos sinápticos hasta que se le indica a al clasificador que su comportamiento es el deseado. También se tiene el *aprendizaje estocástico*, mismo que se basa en modelar al clasificador como un sistema dinámico de mínima energía, en el cual *la función de energía de Lyapunov* se minimiza mediante una función de distribución de probabilidad. El estado de mínima energía es alcanzado cuando las trayectorias de la función de error (deducida a partir de la función de energía) convergen[43][44].

2.3.2. Aprendizaje no supervisado

Este paradigma de entrenamiento no requiere de vectores profesor (salida deseada) para reajustar los parámetros del clasificador, sólo se le presentan los vectores de entrada para que los clasifique según su parecido con las categorías que tiene a su disposición (como sucede en aprendizaje Hebbiano[42]). A medida que se le presenta un mayor número de entradas nuevas al clasificador, este las va agrupando y genera sus propias clases, tal como sucede en los *self-organized maps (SOM)*[43] y en el *custering*.

2.4. Clasificación de patrones

Clasificación (o reconocimiento) de patrones es un área de estudio que actualmente toma mucha importancia a medida que los sistemas computacionales se vuelven más potentes y, por ende, adecuados para implementar algoritmos que ejecuten estas tareas taxonómicas. Dichas tareas consisten de manera general en *clasificar objetos nuevos (mediciones) dentro de un conjunto de elementos (categorías) ya conocidos*[51]. Como se puede observar en la Figura 3, se tiene un espacio de entrada X y un conjunto de categorías $\Omega = \{\omega_1, \omega_2, ..., \omega_k, ...\} \subseteq X$, representado por $\{\boldsymbol{\rho}_1, \boldsymbol{\rho}_2, ..., \boldsymbol{\rho}_k, ...\}$ respectivamente. El clasificador D recibe como estímulo a cada objeto $\mathbf{x}^{(i)} = (x_1, x_2, ..., x_j, ..., x_m) \in X$; usando los *centros paramétricos* como *clases* conocidas $\{\boldsymbol{\rho}_1, \boldsymbol{\rho}_2, ..., \boldsymbol{\rho}_k, ...\}$ (la información que tiene a su disposición), representativos de Ω, D analiza los pares $\{\mathbf{x}^{(i)}, \boldsymbol{\rho}_k\}$ componente a componente mediante un criterio de asociación

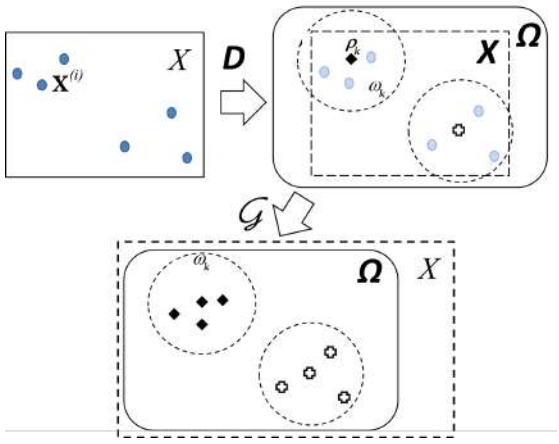


Figura 2.4: Esquema de un clasificador.

$g(\cdot,\cdot)$ que le indica al clasificador cuánto se parece cada par, de modo que el resultado sea una decisión: a qué clase se asigna el objeto nuevo $\mathbf{x}^{(i)}$. Se tiene a continuación la forma de definir de manera formal al ente ejecutor de tales tareas[52]:

DEFINICIÓN **3** *Sea $\mathbf{x}^{(i)} \in X$ una medición (sujeto de entrada), con $X \subseteq \mathbb{R}^m$ como espacio de entrada y además, sea $\omega_k \subset \Omega \subseteq X$ una categoría (clase conocida) representada por su centro paramétrico $\boldsymbol{\rho}_k \in \mathbb{R}^m$, entonces un **clasificador** es una transformación no necesariamente lineal D, tal que:*

$$D : X \longrightarrow X$$

la cual establece que $\mathbf{x}^{(i)}$ será asignado a ω_k según la función de correspondencia:

$$I = \mathcal{G}\big(\max_k\{g(\mathbf{x}^{(i)}, \boldsymbol{\rho}_1), g(\mathbf{x}^{(i)}, \boldsymbol{\rho}_2), ..., g(\mathbf{x}^{(i)}, \boldsymbol{\rho}_k), ..., g(\mathbf{x}^{(i)}, \boldsymbol{\rho}_N\}\big); \{\{i\}_1^n, \{k\}_1^N\} \subset \mathbb{Z}^+$$

donde n es el número de objetos de entrada, N el número de categorías conocidas, $g(\cdot) \in \mathbb{R}$ representa un criterio de asociación (o función discriminante) y $\mathcal{G}(\cdot) \in \mathbb{Z}^+$ asigna a I el valor del índice k para el cual sea máximo el valor de $g(\cdot)$, para el sujeto de entrada i.

Existen variadas formas de apreciación en la literatura para establecer enfoques de clasificadores, de modo que se observen diferencias bien marcadas y objetivas entre ellos . Se verán a continuación tres criterios que sobresalen al categorizar a los clasificadores.

En el Instituto Politécnico Nacional (*IPN*) se han desarrollado mejoras un tanto sencillas pero efectivas al carácter asociativo de los SOM, propuestos originalmente por Kohonen; además, agregan de manera conveniente el ‘‘enfoque asociativo’’ a su categorización[52]:

- ***Enfoque de mínima distancia.*** Emplea aprendizaje supervisado y su criterio discriminante son *métricas*, tales como pueden ser la distancia Euclideana o de Manhattan. Un elemento nuevo es asignado a una clase existente si la métrica es mínima.

- ***Enfoque Neuronal.*** Este enfoque (objeto de muchos eufemismos claramente infundados que reflejan poco conocimiento[52]) está basado en la conectividad de sus nodos de procesamiento y el ponderamiento de tales conexiones. Usan aprendizaje supervisado y no supervisado para reajustar su base de conocimiento (pesos sinápticos). En este tipo de clasificadores no es tan trivial visualizar lo establecido en la definición 3, dado que las clases existentes a las cuales les sería asignada una observación nueva no están expresadas explícitamente.

- ***Enfoque Probabilístico-Estadístico.*** Se basa en modelos de probabilidad y estadística, asociando los vectores de entrada con vectores de distribuciones de probabilidad que son producto de experiencias previas.

- ***Enfoque Sintáctico-Estructural.*** Opera con información simbólica, con una analogía entre la estructura de algún patrón y la sintáxis de un lenguaje, que estará determinado por una gramática. Al introducir la teoría de los lenguajes formales se tienen clasificadores sintácticos llamados *parsers* a los cuales les será presentada una secuencia ordenada de símbolos, orden que será supervisado por un *autómata*, o *máquina de estados finitos (FSM)*, que ayudará a decidir si tal secuencia pertenece o no a un lenguaje.

- ***Enfoque Asociativo.*** Este enfoque se encuentra convenientemente construido sobre la bondad del enfoque neuronal, existen versiones originales de operación muy sencilla como es el caso de los *SOM* (neuronal-asociativo) donde se utilizan criterios de mínima distancia y aprendizaje no supervisado; así mismo, versiones mejoradas como las llamadas memorias asociativas que funcionan básicamente de la misma forma, con excepción de que usan diferentes criterios discriminatorios y funciones de correspondencia.

En lo posterior, las siguientes apreciaciones consideran algunos de los enfoques de la anterior, de modo que se expondrán sólo los casos que no hayamos mencionado. Para los ya mencionados, se hará la aclaración correspondiente.

S. Theodoridis y K. Koutroumbas[51] establecen una categorización muy buena en el sentido de que generaliza la definición de los enfoques en un sentido más matemático:

- ***Clasificadores Basados en la teoría de decisión de Bayes*** Expuesto anteriormente como ‘‘Enfoque Probabilístico-Estadístico’’.

- ***Clasificadores lineales***. Este tipo de clasificadores están categorizados en base a su función discriminante, la cual debe ser de orden lineal, además de centrar su aplicación para problemas de dos clases linealmente separables. Más específicamente se trata de una hipersuperficie (hiperplano) $h(\bullet)$, que en un espacio n-dimensional de características estaría descrito por

$$h(\mathbf{x}) = \mathbf{w}^T\mathbf{x} + b,$$

donde $\mathbf{w}^T = (w_1, ..., w_n)^T$ es el vector de pesos, b es el umbral de activación y $\mathbf{x} = (x_1, ..., x_n)$ es el patrón de entrada al clasificador. El ejemplo más demostrativo de este tipo de clasificadores el *Perceptrón simple*.

- ***Clasificadores no-lineales***. Este tipo de clasificadores son capaces de resolver problemas de clasificación con separabilidad lineal, como lo hacen los clasificadores lineales. Pero además son capaces de resolver problemas cuyo espacio de características no sea linealmente separable; capacidad de la que prescinde el clasificador lineal. Basta con agregar una capa oculta al perceptrón simple para obtener un clasificador no-lineal, de modo que se obtiene al ejemplo más representativo de un clasificador no-lineal: *El Perceptrón Multicapa*.

- ***Comparación de plantillas***. Expuesto anteriormente como ''Enfoque de mínima distancia''.

- ***Clasificador dependiente del contexto***. Expuesto anteriormente como ''Enfoque Sintáctico-Estructural''.

- ***Clustering***. La característica principal de estos ''clasificadores'' es que son entrenados de manera no-supervisada; no es requerido un conjunto de ejemplos durante el entrenamiento para definir el espacio de características. Los algoritmos de clustering son capaces de hacer auto-clasificación (*self-organizing*) de patrones que comparten características, formando grupos de ellos que son sensibles a similitudes.

Seema Asht y Rajeshwar Dass [53] publican de manera objetiva una revisión actualizada de enfoques y exponen la siguiente categorización:

- ***Modelo estadístico*** Expuesto anteriormente como ''Enfoque Probabilístico-Estadístico''.

- ***Modelo estructural*** Expuesto anteriormente como ''Enfoque Sintáctico-Estructural''.

- ***Modelo de comparación de plantillas*** Expuesto anteriormente como ''Enfoque de mínima distancia''.

- ***Modelo basado en redes neuronales*** Expuesto anteriormente como ''Enfoque Neuronal''.

- ***Modelo basado en borrosidad (o difuso)*** Este enfoque está basado en la teoría de los conjuntos difusos, los cuales tienen como característica principal el concepto de *grado de pertenencia* $\mu \in [0, 1]$, con lo que se mide de manera cuantitativa que tanto un objeto pertenece a un conjunto o varios conjuntos. Según esta descripción, la relación entre la lógica difusa y la clasificación es íntima y se apega más al mundo real. LAMDA, por ejemplo, pertenece a esta categoría.

- ***Modelo híbrido*** Bajo este enfoque se encuentra una amplia variedad de clasificadores, ya que se estudian aquí combinaciones (o más bien contribuciones múltiples) de todos los enfoques. Las investigaciones en esta área se basan en el hecho de que inherentemente cualquier algoritmo tiene desventajas que se busca disminuir haciendo propuestas complementarias. Por ejemplo, se pueden hallar contribuciones de algoritmos estadísticos con estructurales para comparar decisiones y aumentar la certidumbre[53].

2.5. Agrupamiento de datos *(Data Clustering)*

En esta sección se verán los conceptos generales que tienen que ver con las metodologías de clasificación no supervisada conocidas como *''Data Clustering''* (también llamadas *''Cluster Analysis''*, *''Segmentation Analysis''*, *''Taxonomy Analysis''*, *''Unsupervised Classification''*), lo cual incluye una revisión de estos tipos de algoritmos así como la descripción general de algunos de los más sobresalientes. Cabe señalar que no se considera a un algoritmo de clustering como clasificador; sino que más bien, un clasificador se encuentra incluido como elemento de un algoritmo de clustering. Veremos también durante el desarrollo de este tema las diferencias principales.

2.5.1. Definición de ''Clustering''

En la literatura se puede hallar una gran variedad de definiciones de este concepto, se tiene a continuación la más importante para el autor[69]:

Definición **4** *Es un método para crear grupos de objetos, de tal forma que los objetos pertenecientes a un grupo son muy similares y, por otro lado, son muy distintos a objetos de otros grupos.*

Implícitamente, cualquier definición formal lleva a una conclusión más general que puede ser el origen de la inspiración que creó este concepto tan útil y de naturaleza muy descriptiva[67]:

''Para entender nuestro mundo es necesario conceptualizar las similitudes y diferencias que hay entre las entidades que lo componen.''

Formalmente se tiene una definición matemática muy general de un sistema de clusters (sistema de grupamiento de datos) es[51, 67]:

DEFINICIÓN **5** *Sea un conjunto* $C = \{C_1, ..., C_k, ..., C_n\}$ *y sea* X *el espacio de entrada, entonces un sistema de clusters* S *está dado por:*

$$S = \left\{ \bigcup_{k=1}^{n} C_k \mid \bigcap_{k=1}^{n} C_k = \varnothing \right\} \subseteq X,$$

de modo que algún objeto $\boldsymbol{x} \in X$ *es asignado por un clasificador a la clase* C_k *y sólo a ella.*

Véase ahora la terminología usada en clustering, así como los conceptos que ésta infiere. En esta sección se tratará primero una descripción general de los elementos principales de un algoritmo de clustering, qué herramientas matemáticas se pueden incluir en éstos, qué tipos de ellos existen en el nivel más alto de la clasificación (*rígidos y difusos*) y después, se expondrán de manera muy simple algunos de los algoritmos de clustering más destacados en la literatura.

2.5.2. Objetos y atributos

En la literatura se define a un vector de variables de entrada como un objeto de entrada o como un sujeto de entrada, de modo que cada vector de entrada se interpreta físicamente como la i-ésima observación $\mathbf{x}^{(i)} \in X = \{\mathbf{x}^{(1)}, ..., \mathbf{x}^{(i)}, ...\} \subset \mathbb{R}^n$. Las componentes de un $\mathbf{x}^{(i)}$, se conocen como descriptores o caraterísticas o atributos (i. e. $\mathbf{x}^{(i)} = (x_1, ..., x_j, ..., x_n)^T$); estos a su vez, pueden ser cualitativos (variables numéricas) o cuantitativos (*variables lingüísticas*).

2.5.3. Funciones discriminantes

En clustering, sobre todo, se usa el concepto de similitud para medir de manera cuantitativa el grado de parecido o semejanza que hay entre dos objetos u observaciones, calculando una *función de similitud*. Por otro lado, la mayoría de los algoritmos de clustering, usan *métricas* para discriminar clusters que no se parezcan a un objeto de entrada, revelando qué tan *próximos* se hallan dos vectores que sean sujetos de comparación. Algunas de las métricas más usadas son[69]: *Distancia euclidiana*, *Distancia de Manhattan*, *Distancia de Chebyshev*, *Distancia de Minkowski*, *Distancia de Mahalanobis* y *Distancia promedio*. También se tienen en la literatura algunas funciones de similitud usuales[69]: *Coeficiente del coseno*, *Coeficiente de correlación de Pearson* y *Coeficiente extendido de Jaccard*.

En este trabajo de tesis, el algoritmo de clustering estudiado es LAMDA y su función discriminante no entra en ninguna de las dos categorías que se han mecionado; aunque es muy similar a la distancia de Mahalanobis. Se ha de hablar de ello en el Capítulo 4, cuando se haga una descripción detallada del clustering difuso y de dicho algoritmo.

2.5.4. Clusters, centros y modos

Los clusters son regiones continuas en un espacio l-dimensional, contienen una densidad alta de puntos que a su vez se encuentran rodeadas por otras regiones de baja densidad (relativamente vacías[69]), las cuales se consideran como regiones de separación[51]. Además, [69] muestra una lista de características que deben compartir una colección de puntos a los cuales se les considera como elementos de un cluster:

1. Compartir propiedades iguales o fuertemente relacionadas.
2. Mostrar mutuamente distancias pequeñas o altos coeficientes de similitud.
3. Un cluster debe ser claramente distinguible de su complemento (el resto de clusters).

Por otro lado, un algoritmo de clustering no necesariamente poseería almacenados a todos los elementos de un cluster. Sólo se ha de recurrir a una representación de todos ellos. A tal representación se le denomina *centro paramétrico* (para clusters cuantitativos), cuyo valor será una función de todos los elementos asignados al cluster que representa. Los valores de un centro paramétrico, son actualizados a medida que nuevos objetos de entrada son asignados durante el agrupamiento al cluster representado. En el caso de clusters cualitativos, la representación de un cluster se llama *modalidad*[69].

Cabe aclarar que en varios textos (e. g. [51, 67, 70]) no se propone un nombre especial para distinguir entre representaciones cualitativas y cuantitativas; se les llama de manera indiscriminada y más general ‘‘centro’’, pues es posible, dependiendo del algoritmo, tratar con espacios de entrada caracterizados por descriptores cualitativos y cuantitativos. Este último es el caso de LAMDA.

2.6. Algoritmos de clustering particional

Ya se ha expuesto que existen dos tipos de algoritmos en el nivel más alto de la clasificación (Sección 2.7). Dentro de ella, los algoritmos de clustering rígido de dividen en dos subcategorías: *clustering jerárquico y clustering particional*. Los dos particionan el espacio de entrada; sin embargo, el clustering particional hace una división en la cual todos los clusters pertenecen al mismo (único) nivel jerárquico (*single partition*); mientras el clustering jerárquico divide el espacio de clasificación en particiones anidadas[69]. El autor de este trabajo de tesis, considera que ésta es la categorización más general entre algoritmos rígidos; sin embargo, existen otras categorizaciones más selectivas que pueden consultarse en [51, 70]: *algoritmos secuenciales, basados en optimización, probabilísticos, basados en detección de fronteras, branch and*

bound, genéticos, de relajación estocástica, basados en búsqueda de valles, de aprendizaje competitivo, basados en transformaciones morfológicas, basados en sub-espacios (análisis de componentes), evolutivos, graph-theoretic y kernel-based.

En esta sección se mostrarán los aspectos fundamentales de los algoritmos de clusteing particional y en la siguiente sección se abordará lo referente a los fundamentos de clustering difuso (en este trabajo de tesis se emplea un algoritmo de clustering difuso). No se considera necesario exponer una sub-clasificación para los algoritmos difusos puesto que ya se ha aclarado que las leyes que rigen a los algoritmos rígidos son casos particulares de las leyes que rigen la lógica difusa. Por lo tanto, se expondrá de manera muy general lo más importante que debe caracterizar a un algoritmo difuso, debido a que, en teoría, cualquier algoritmo rígido puede generalizarse para volverse difuso. En efecto, existen ya versiones difusas de algunos de los algoritmos rígidos más populares, tales son los casos de (por citar algunos): *Fuzzy k-means, Fuzzy c-modes* y el *Fuzzy c-means.*

Puesto que el término *''particional''* es demasiado general, no es posible hablar de algún esquema único y global, para describir el funcionamiento de todos los algoritmos que pueden tipificarse como tales. La subcategoría más importante, dentro de los algoritmos particionales, son los algoritmos basados en la **optimización de la función de costo**[2] y se expone en este trabajo de tesis debido a que LAMDA tiene una estructura bastante similar, lo que contribuye en gran medida a su comprensión. *''Center-based algorithms''* es un término menos preciso usado por [69] para denominar a la misma categorización. Estos algoritmos están diseñados de tal manera que cuentan con su propia función objetivo, la cual se puede usar como criterio de parada del algoritmo, buscando su minimización. Estos algoritmos son muy populares por su gran eficiencia para trabajar sobre bases de datos de gran tamaño[51] (millones de registros); tal funcionalidad se conoce como *data mining*[67].

Se define en principio el contexto del algoritmo. En este caso, la función de costo es una función de error que indica qué tan compacto es un cluster. Mientras más compactos sean los clusters, menor será el valor de la función de error en cada pasada del algoritmo. Dado el valor de la función de error como parámetro de compacidad y un umbral de éste, el algoritmo podrá determinar que el proceso de agrupamiento ha terminado. Supóngase que se desea obtener una partición $C = \{C_1, ..., C_i, ..., C_j, ..., C_m\}$ de m clusters, que cada cluster a su vez estará representado por su valor medio respectivo $\{\boldsymbol{\rho}_1, ..., \boldsymbol{\rho}_i, ..., \boldsymbol{\rho}_j, ..., \boldsymbol{\rho}_m\}$, previamente el algoritmo ha asignado a cada C_i una cantidad de objetos de entrada igual a n_i. Si se sabe que cada C_i se encuentra representado en la base de conocimiento del algoritmo por $\boldsymbol{\rho}_i$, supóngase que se tiene un objeto de entrada $\mathbf{x}_k$ que por ahora es elemento de (o está asignado a) C_i, entonces el valor J_e de la función criterio (función costo), llamada para este caso *Suma de Errores Cuadráticos*, está dado por[70]:

$$J_e = \sum_{i=1}^{m} J_i, \tag{2.1}$$

[2] A veces se usan los términos ''función paricuinal''[51] en lugar de función costo.

donde el *error efectivo* (error cuadrático) por cluster es:

$$J_i = \sum_{k=1}^{n_i} ||\mathbf{x}_k - \boldsymbol{\rho}_i||^2,$$

donde a su vez, la media (centro paramétrico o representativo) del i-ésimo cluster está dada por:

$$\boldsymbol{\rho}_i = \frac{1}{n_i} \sum_{k=1}^{n_i} \mathbf{x}_k.$$

Una vez definido lo anterior, supóngase que para el objeto $\tilde{\mathbf{x}}_k \in C_i$ se desea verificar si es mejor que permanezca asignado a C_i o es mejor que sea reasignado a C_j, a efecto de optimizar el agrupamiento (minimizando J_e). Para resolver tal planteamiento, es necesario tomar en consideración lo siguiente: si $\tilde{\mathbf{x}}_k$ efectivamente fuese trasladado a C_j, correspondientemente habrá que actualizar el centro $\boldsymbol{\rho}_j$ mediante[70]:

$$\hat{\boldsymbol{\rho}}_j = \boldsymbol{\rho}_j + \frac{\tilde{\mathbf{x}}_k - \boldsymbol{\rho}_j}{n_j + 1}; \tag{2.2}$$

además habría que actualizar J_j mediante:

$$\hat{J}_j = J_j + \frac{n_j}{n_j + 1} ||\tilde{\mathbf{x}}_k - \boldsymbol{\rho}_j||^2. \tag{2.3}$$

De forma análoga, habrá que actualizar tanto el error cuadrático J_i como el centro $\boldsymbol{\rho}_i$ del cluster C_i del cual se ha sustraído[70] $\tilde{\mathbf{x}}_k$:

$$\hat{\boldsymbol{\rho}}_i = \boldsymbol{\rho}_i - \frac{\tilde{\mathbf{x}}_k - \boldsymbol{\rho}_i}{n_i - 1}; \tag{2.4a}$$

$$\hat{J}_i = J_i - \frac{n_i}{n_i - 1} ||\tilde{\mathbf{x}}_k - \boldsymbol{\rho}_i||^2. \tag{2.4b}$$

Las ecuaciones (2.2) y (2.4a) se hallan implícitas en el cálculo de los errores, lo realmente importante para resolver el planteamiento anterior, es centrar atención en los segundos términos del segundo miembro tanto de (2.3) como de (2.4b), debido a que tales expresiones representan el cambio en la función de error de cada cluster, debido a la transferencia de $\tilde{\mathbf{x}}_k$ desde C_i hacia C_j. Por lo tanto, para determinar si la transferencia del objeto es viable se debe cumplir que[70]:

$$\frac{n_j}{n_j + 1} ||\tilde{\mathbf{x}}_k - \boldsymbol{\rho}_j||^2 \;<\; \frac{n_i}{n_i - 1} ||\tilde{\mathbf{x}}_k - \boldsymbol{\rho}_i||^2.$$

En caso contrario, la transferencia no es factible.

Supongamos ahora que en otro escenario se busca verificar si C_i es óptimo para algún $\tilde{\mathbf{x}}_k \in C_i$; basta (por observación) calcular[70]:

$$l = \arg\min_{l} ||\tilde{\mathbf{x}}_k - \boldsymbol{\rho}_l||^2 \quad \forall l \in 1, 2, ..., m, \tag{2.5}$$

donde l es el índice del cluster para el cual $||\tilde{\mathbf{x}}_k - \boldsymbol{\rho}_l||^2$ es mínima. Si $l = i$ a lo largo de la partición actual, el cluster C_i es óptimo. En cualquier otro caso, habrá que transferir $\tilde{\mathbf{x}}_k$ desde C_i hasta C_l.

Lo anterior significa, que se puede buscar el cluster en la partición C que más se apegue a los valores de algún objeto de entrada; es decir que para algún $\tilde{\mathbf{x}}_k$ que se adquiera en línea, éste sería clasificado en C_l, de acuerdo con lo especificado por (2.5). Como consecuencia del último escenario planteado, habrá que reparametrizar la nueva partición recalculando (2.1). Cabe destacar que cada vez que se recalcula el centro de un cluster mediante (2.2) ó (2.4a), el algorimo aprende; actualiza su base de conocimiento.

Básicamente se acaba de describir de manera general el comportamiento de la mayoría de los algoritmos de clustering particional rígido. Este principio es fundamental, ya que es empleado en algoritmos muy comunes y usados conocidos como el *ISODATA (Iterative Self-Organizing Data Analysis Technique Algorithm), k-means, c-means, k-modes*, cuya principal característica en versiones extendidas es que el número inicial de clusters es desconocido, lo cual aporta ventajas considerablemente sustanciales para su uso en la extracción de características subyacentes en colecciones de datos de grandes dimensiones (*data mining*). Enfoques más generales como *clustering probabilístico, clustering posibilístico, clustering por detección de fronteras e incluso el clustering difuso* también están basados en la optimización de una función objetivo[51].

2.7. Generalidades de clustering difuso

Todo lo que se ha expuesto hasta ahora define herramientas matemáticas para tratar, en general, con un sólo tipo de algoritmos de clustering: *clustering rígido*; sin embargo, las técnicas empleadas en estos algoritmos no son más que casos especiales de las que se usan en algoritmos de *clustering difuso*.

A efecto de soportar la afirmación anterior, se tiene a continuación la definición formal del resultado de agrupamiento, la *matriz de particionamiento*, para algún algoritmo dado:

$$U = \begin{pmatrix} u_{11} & \dots & u_{1n} \\ \vdots & u_{ji} & \vdots \\ u_{k1} & \dots & u_{kn} \end{pmatrix} \tag{2.6a}$$

$$\sum_{j=1}^{k} u_{ji} = 1, \tag{2.6b}$$

$$\sum_{i=1}^{n} u_{ji} > 0, \tag{2.6c}$$

donde n es el número de objetos de entrada y k es el número de clusters que *"descubrió"* el algoritmo, de tal manera que se dice que el espacio de características está $k - particionado$. Para el caso rígido, cada u_{ji} deberá ajustarse a lo siguiente (en adición a las ecuaciones (2.6)):

$$u_{ji} \in \{0, 1\}. \tag{2.7}$$

Lo que significa la restricción (2.7) es que para algún $u_{ji} = 1$ el objeto pertenece a la clase j pero a ninguna otra; ningún otro elemento de esta columna i puede ser igual a 1, todos son cero, a efecto de que se cumpla (2.6b). Por otra parte, con mucha más flexibilidad y también en complemento con (2.6), el caso difuso ajusta los u_{ji} a lo siguiente:

$$u_{ji} \in [0, 1]. \tag{2.8}$$

La restricción (2.8) implica que no es necesario que sólo un elemento de la columna i de U deba tener valores diferentes de cero; permitiendo que en alguna medida, el patrón de entrada i sea miembro de todas las clases simultáneamente[51]; claro, ajustándose a lo dispuesto por (2.6b), poniendo así de manifiesto la naturaleza difusa de la clasificación y confirmando su mayor generalidad.

La asignación de un objeto de entrada a una clase, universalmente, es el estado terminal que adquiere dicho objeto al ser procesado por cualquier algoritmo de clasificación; por lo tanto, a fin de cuentas ha de decidir el algoritmo de clustering difuso a que clase pertenece dicho objeto de entrada, de modo que una partición rígida reside subyacente a una partición difusa[69]. Con base en el argumento anterior, se plantea la correspondiente partición rígida adyacente mediante:

$$\omega_{ri} = \begin{cases} 1 \quad si \quad r = \arg \ \max_{1 \leq j \leq k} \ u_{ji}, \\ \\ 0 \qquad \text{en cualquier otro caso}, \end{cases} \tag{2.9}$$

de modo que (2.9) se halle sujeta a (2.6), u_{ji} estén sujetos a (2.8) y $\omega_{ri} \in \{1, 0\}$, tal como se cumple en (2.7).

Para los dos casos (rígido y difuso), la inecuación (2.6c) establece que no existen clases vacías en el espacio de clases. Para mayor minuciosidad y apuntalamiento de lo que se va a mostrar en el Capítulo 4, vamos a explanar a continuación los fundamentos sobre los que descansa el clustering difuso, de tal manera que la terminología que se use en adelante resulte más familiar.

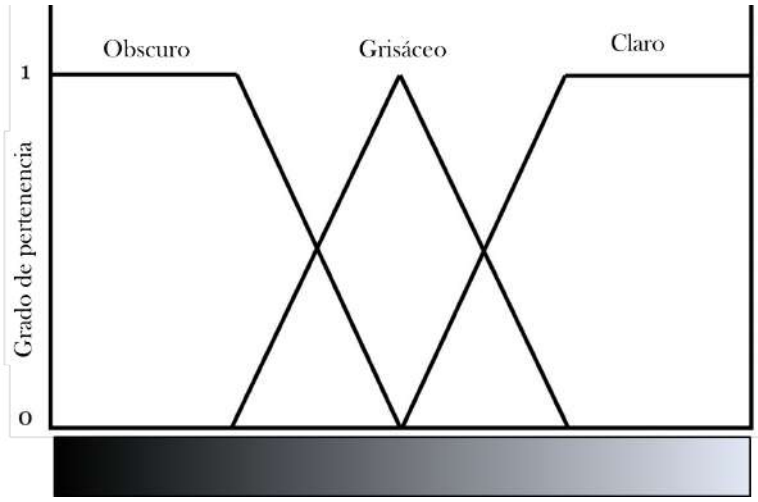


Figura 2.5: Escala de grises mapeada al concepto de conjuntos difusos; tales conjuntos son *Obscuro, Grisáceo y claro*. Obsérvese la imprecisión de los términos lingüísticos que refieren a los conjuntos y la forma en que tal impresición es manejada de manera consistente, de tal manera que se abarca totalmente el universo de discurso ‘‘escala de grises’’.

2.7.1. Conjuntos difusos

Los conjuntos difusos son hoy en día un concepto interdisciplinario, idea original de Lotfi A. Zadeh[72] (ingeniero en eléctrico por la Universidad de Teherán), una herramienta muy útil que se aplica principalmente a la clasificación de objetos. Desde el punto de vista de un objeto, en lógica difusa tal objeto está caracterizado por funciones de pertenencia, que le otorgan un grado de pertenencia diferente por cada conjunto (o clase) con el que éste es asociado.

En un contexto más general, desde el punto de vista de un conjunto A, a un objeto $\mathbf{x}$ se le asocia una función de pertenencia $f_A(\mathbf{x})$ (léase: *función de pertenencia de* $\mathbf{x}$ *a* A; o solamente: *pertenencia de* $\mathbf{x}$ *a* A). Dicha función está definida en el intervalo [0, 1], siendo 0 el grado mínimo de pertenencia (no pertenencia de $\mathbf{x}$ a A) y 1 el grado máximo de pertenencia (pertenencia total $\mathbf{x}$ a A). Pero no sólo eso, además, dado que el grado de pertenencia es un número real, $\mathbf{x}$ puede pertenecer de manera parcial a A; por ejemplo, $f_A(\mathbf{x}) = 0.7$ se podría considerar un valor *cercano al más alto* (pero no lo suficiente para decir que $\mathbf{x}$ sólo pertenece a A), o $f_A(\mathbf{x}) = 0.2$ se consideraría un valor *muy bajo* (pero no nulo, como para decir que $\mathbf{x}$ no pertenece a A en lo absoluto). Es esta conjunción de términos lingüísticos los que le dan mucho sentido a la lógica difusa (véase la figura 2.5), ya que en el mundo real la mayoría de las cosas no están definidas de manera tan precisa que se puedan caracterizar adecuadamente mediante la lógica convencional, la cual sólo considera valores de pertenencia en $\{0, 1\}$; es decir: $\mathbf{x}$ NO pertenece a A, o bien $\mathbf{x}$ SÓLO pertenece a A; FALSO o VERDADERO, sin lugar a imprecisiones. La naturaleza, es inherentemente imprecisa. Se ha postulado que la lógica convencional es un caso particular de la lógica difusa, por lo tanto la mayoría de las

operaciones convencionales se extienden a los conjuntos difusos[72, 69]. De manera que se definen las operaciones básicas generalizadas a la lógica difusa :

Sean el universo de discurso X, tres conjuntos difusos A, B y C (véanse a A y B en la Figura 2.6a), cada uno asociado a una función de pertenencia $f_A(\mathbf{x}), f_B(\mathbf{x})$ y $f_C(\mathbf{x})$ $\forall \mathbf{x} \in X$ (por simplicidad en lugar de $f_A(\mathbf{x}), f_B(\mathbf{x})$ y $f_C(\mathbf{x})$ sólo se escribe f_A, f_B y f_C) entonces:

- **La igualdad** entre dos conjuntos difusos A y B, escrita como $A = B$, existe si y sólo si

$$f_A = f_B$$

.

- **El complemento** de un conjunto difuso A se denota como A' y se define por:

$$f_{A'} = 1 - f_A$$

- El conjunto difuso A está **contenido** en (o es *subconjunto*, o *es menor o igual que*) el conjunto difuso B, si y sólo si $f_A \leqq f_B$; expresado de otra forma:

$$A \subset B \Longleftrightarrow f_A \leqq f_B$$

- **La unión** (véase la Figura 2.6b) de dos conjuntos difusos A y B es un conjunto difuso C tal que $C = A \cup B$, cuyas funciones de pertenencia se encuentran relacionadas de la siguiente manera:

$$f_C = \text{máx}\{f_A, f_B\},$$

escrito de otra forma:

$$f_C = f_A \vee f_B.$$

El operador $\cup$ cuenta con la propiedad asociativa.

- **La intersección** (véase la Figura 2.6b) de dos conjuntos difusos A y B, asociados respectivamente a las funciones de pertenencia f_A y f_B, es un conjunto difuso C tal que $C = A \cap B$ cuyas funciones de pertenencia se encuentran relacionadas de la siguiente manera:

$$f_C = \text{mín}\{f_A, f_B\},$$

escrito de otra forma:

$$f_C = f_A \wedge f_B.$$

El operador $\cap$ cuenta con la propiedad asociativa. Al igual que en los conjuntos convencionales, dos conjuntos difusos A y B son disjuntos si $A \cap B = \emptyset$. Pueden verificarse en [72] algunas otras propiedades importantes tales como las leyes de De Morgan, aritmética y álgebra para conjuntos difusos.

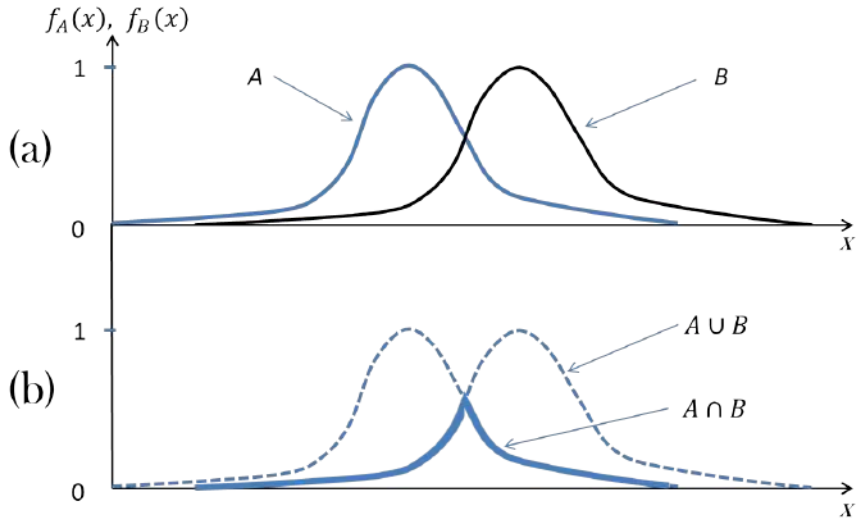


Figura 2.6: Ilustración de (a) dos conjuntos difusos A y B, así como (b) la unión (línea punteada) y la intersección (línea remarcada) entre ellos.

Definiciones útiles

Veremos en este apartado algunas relaciones básicas entre conjuntos difusos, las cuales nos serán útiles para entender cuestiones relacionadas con el clustering difuso que se implementaría en este trabajo de tesis. En algunas ocasiones, se hablará del caso particular de lógica convencional a partir del cual se plantea alguna definición.

En general si se tienen funciones (o vectores) $g_1, g_2, ..., g_n$, una *combinación convexa*, desde el punto de vista convencional, está definida por:

$$\{\lambda_1 g_1 + \lambda_2 g_2 + ... + \lambda_n g_n \ \mid \ \sum_{i=1}^{n} \lambda_i = 1\}. \tag{2.10}$$

Escrito en un modo más conveniente[3], para dos funciones g_1 y g_2, se puede construir una combinación convexa de la forma[72]:

$$\{\lambda g_1 + (1-\lambda) g_2 \ \mid \ 0 \leq \lambda \leq 1\},$$

lo cual es fácil probar que cumple con (2.10). Generalizando el concepto anterior para conjuntos difusos; sean tres conjuntos difusos A, B y Λ entonces existe una combinación convexa que los relaciona y está definida por:

$$(A, B; \Lambda) = \Lambda A + \Lambda' B$$

[3]Es más conveniente dado que en reconocimiento de patrones se busca verificar pertenencia a un conjunto, delimitado por $[x_0, x_1]$.

Lo cual es equivalente a escribir en términos de las funciones de pertenencia de cada conjunto:

$$f_{(A,B;\Lambda)} = f_\Lambda f_A + (1 - f_\Lambda) f_B.$$

Un uso extremadamente importante de la combinación convexa de conjuntos difusos, se lleva a cabo en LAMDA para obtener el Grado de Adecuación Global (concepto que será abordado con posterioridad) y está plasmado en la siguiente

DEFINICIÓN **6** *Sea $M_A = \text{máx}_x\, f_A(x)\ \forall x$ el máximo grado de A, esencialmente alcanzado por dos puntos x_0 y x_1 tales que $x_0 \neq x_1$, entonces también M es esencialmente alcanzado por todo x tal que:*

$$x = \lambda x_0 + (1 - \lambda) x_1; \quad 0 \leq \lambda \leq 1.$$

Esta definición podría ser generalizada para alguna vecindad esférica de M, en caso de que $M = \sup_x f_A(x)$ para algún $x = x_0$ que sea único.

En otro respecto, tómese primero como consideración un espacio euclidiano $X \in \mathbb{R}^n$, dos conjuntos difusos A y B así como sus respectivas funciones de pertenencia $f_A(x)$ y $f_B(x)$.

Un conjunto A *es convexo* si existe un conjunto Γ_α tal que esté definido como:

$$\Gamma_\alpha = \{x \mid f_A(x) \leqq \alpha\} \quad \forall \alpha \in (0, 1]. \tag{2.11}$$

Para entender la definición anterior en términos geométricos y de la trayectoria f_A, según lo expresado por la Ecuación (2.11), existe una combinación convexa para dos puntos x_1 y x_2 tales que se puede afirmar que un conjunto difuso A es convexo si se cumple que:

$$f_A[\lambda x_1 + (1 - \lambda) x_2] \geqq \text{mín}\{f_A(x_1), f_A(x_2)\} \quad \forall x_1, x_2 \in X; \quad \lambda \in [0, 1],$$

tal como se muestra en la Figura 2.7.

Teorema 1 *Si A y B son convexos, también lo es $A \cap B$.*

Un conjunto difuso A es *acotado* en $\Gamma_\epsilon = \{x \mid f_A(x) \leqq \epsilon\}$ si y sólo si existe un $\epsilon > 0$ tal que se pueda construir una región $\mathcal{B}_{a_0}(\epsilon)$ de modo que se cumpla:

$$||x|| \leq \mathcal{B}_{a_0}(\epsilon),$$

donde $\mathcal{B}_{a_0}(\epsilon)$ está centrada en a_0 (usualmente $a_0 = \sup f_A(x);\ \forall x \in X$) y puede ser un círculo en $\mathbb{R}^2$, una esfera en $\mathbb{R}^3$ o una *bola*[4] en $\mathbb{R}^n$; cuyo radio es ϵ. Un resultado especialmente útil en reconocimiento de patrones, es que:

[4]Una bola (cerrada) es la región $\mathcal{B}_a(\epsilon) = \{x | d(a, x) < \epsilon\}$ con centro en a definida en algún espacio inducido por alguna métrica d (incluyendo la usual d_2).

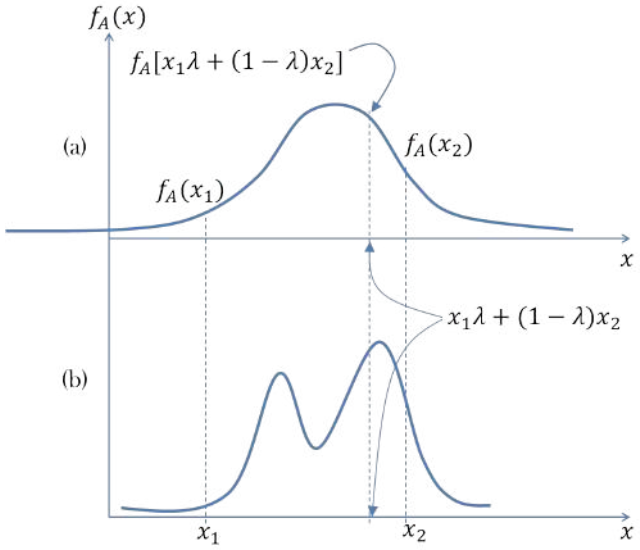


Figura 2.7: Ilustración de (a) un conjunto difuso convexo en $\mathbb{R}^1$ así como sus propiedades y (b) de un conjunto difuso NO convexo.

Definición **7** *Si se garantiza mediante las definiciones anteriores que A es convexo y acotado se puede construir un hiperplano H, por ejemplo del lado que no contiene al origen en $\mathbb{R}^2$, que divida a X de tal forma que se cumpla:*

$$f_A(x) \leqq \epsilon,$$

para toda x en el lado de H que no contiene al origen (véase la Figura 2.8).

Lo mismo se podría hacer de manera iterativa para varios puntos sobre $\mathcal{B}_{a_0}(\epsilon)$, de tal forma que quede circunscrito un cluster.

Teorema 2 *Sean A y B conjuntos difusos convexos en $\mathbb{R}^n$, con máximos grados M_A y M_B, respectivamente. Sea también $M_{A\cap B}$ el máximo grado de la intersección $A \cap B$, entonces:*

$$D = 1 - M_{A\cap B}$$

es el máximo grado de separabilidad entre A y B. Véase que $M_A = \sup_x f_A(x), M_B = \sup_x f_B(x)$ y $M_{A\cap B} = \sup_x \text{mín}\{f_A(x), f_B(x)\}$.

EL Teorema 2 difiere de su homónimo para conjuntos convencionales en que para cuales quiera conjuntos convencionales A y B, se dice que son separables si y sólo si son mutuamente excluyentes (disjuntos); es decir, $A \cap B = \emptyset$. Ésto no es un requerimiento para conjuntos difusos.

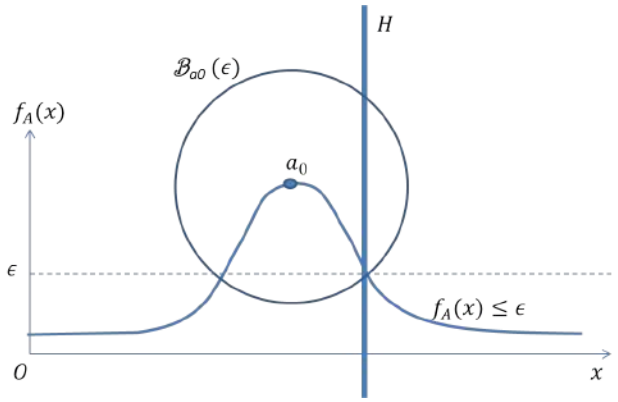


Figura 2.8: Conjunto convexo A, y su respectiva función de pertenencia $f_A(x)$, acotados por el hiperplano H.

El Teorema 2 se conoce como *Teorema de separabilidad para conjuntos difusos* y según la Definición 7, es posible construir un hiperplano H que pase por $M_{A\cap B}$. Por lo tanto, obsérvese en la Figura 2.9 que tanto la Definición 7 como el Teorema 2 están íntimamente relacionados y que además las ideas que aportan son de mucha utilidad si se requiere hacer un análisis detallado de la partición establecida para algún algoritmo de clasificación difusa.

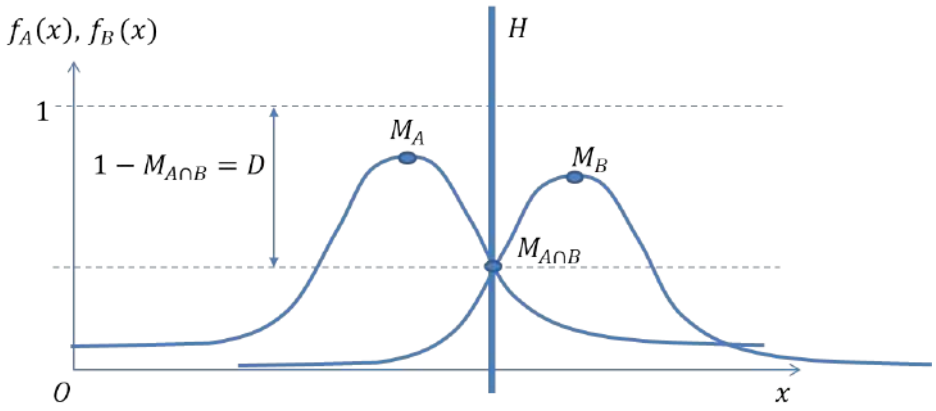


Figura 2.9: Ilustración del teorema de separabilidad para conjuntos difusos en $\mathbb{R}^1$.

2.7.2. Estructuración de un algoritmo de clustering difuso

Tal como se ha mencionado en la Sección 2.6, los algoritmos difusos fundamentan su articulación en una función objetivo que habrá que optimizar; esencialmente, minimizar la función costo. Se ha expuesto también, de forma general, la forma que toma la función costo para un algoritmo rígido basado en optimización. Véase la Ecuación (2.1). Supóngase que se tiene un conjunto de *centros paramétricos* $\Theta = \{\boldsymbol{\rho}_1, ..., \boldsymbol{\rho}_j, ..., \boldsymbol{\rho}_k\}$ (representando a los clusters C_j), particularmente para el caso de un algoritmo difuso, tal función costo es de la

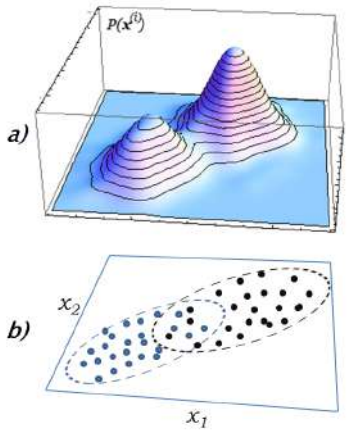


Figura 2.10: Mezcla de dos distribuciones gaussianas de probabilidad que representarían a un par de clusters (b) en $\mathbb{R}^2$, cada uno con un centro paramétrico que genera la respectiva vista tridimensional de la distribución de los grados de pertenencia (a), mediante una función de la forma: $P(\boldsymbol{x}^{(i)}) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left[\frac{1}{2}\left(\frac{\boldsymbol{x}^{(i)}-\boldsymbol{\mu_j}}{\sigma}\right)^2\right]$.

forma:

$$J_q(\Theta, U) = \sum_{i=1}^{n}\sum_{j=1}^{k} u_{ij}^q f(\mathbf{x}^{(i)}, \boldsymbol{\rho}_j), \tag{2.12}$$

donde $\Theta \in M_{n\times k}$ es la matriz[5] paramétrica del espacio m-dimensional en el que se hallan los clusters, cada $\boldsymbol{\rho}_j = (\rho_1, \rho_2, ..., \rho_m, t_1, ..., t_l, ...)$ es el centro paramétrico del cluster C_j (siendo los t_l parámetros de las funciones de pertenencia de los clusters), la función $f(\bullet, \bullet)$ es la función de disimilitud entre el objeto de entrada $\mathbf{x}^{(i)}$ y el centro $\boldsymbol{\rho}_j$, además $q \in \mathbb{R}^+ - \{0\}$ es un parámetro de la función costo llamado *difusor*, u_{ij} es el grado de membresía del objeto $\mathbf{x}^{(i)}$ al cluster C_j y n es el número de objetos de entrada. Se dice entonces que habrá que minimizar J_q con respecto a Θ y U, sujeta a las restricciones (2.6) y (2.8). A la ecuación (2.12) se le conoce como ecuación probabilística de clustering difuso, este nombre, en términos de probabilidad, se debe a que J_q describe una densidad de probabilidad llamada *mezcla de densidades (density mixure)*.

Se puede observar en la Figura 2.10 que si los clusters estuvieran parametrizados, mediante cada $\boldsymbol{\rho}_j$, de tal forma que sus grados de pertenencia tuviesen una distribución gaussiana, entonces J_q estaría definida en una forma tal que se vería como una mezcla parcial de gaussianas que se intersectan (traslapan) en las regiones más alejadas de sus centros (no se confunda el término simple ‘‘centro’’ con ‘‘centro paramétrico’’: $\boldsymbol{\rho}_j$), cada centro paramétrico tendría el aspecto siguiente: $\boldsymbol{\rho}_j = (\rho_1, \rho_2, ..., \rho_m, \boldsymbol{\mu_j}, \sigma_j)$, donde $\boldsymbol{\mu_j}$ y σ_j son la media y la varianza, respectivamente, de la función de densidad guassiana asociada al cluster[6] j. LAMDA

[5]Se define a $M_{n\times m}$, como el conjunto de matrices de $n \times m$.

[6]Para el caso particular de la distribución gaussiana, $\boldsymbol{\mu_j} = (x_1, x_2, ..., x_m)$ es el centro del cluster j; de modo que se puede escribir su centro paramétrico: $\boldsymbol{\rho}_j = (\boldsymbol{\mu_j}, \sigma_j)$.

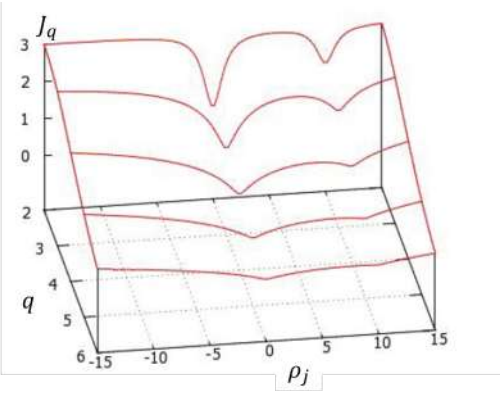


Figura 2.11: Ejemplo de una superficie de error para $J_q(\Theta, U)$ en términos de $q \in \{1, ..., 6\}$.

usa tanto la distribución gaussiana como la binomial, según se requiera, lo cual será discutido en el Capítulo 4.

Dado que, por lo general en un algoritmo difuso, se busca hallar el mínimo global en la superficie de error, dicha superficie podría llegar a ser impredecible (como sucede con la superficie de error de una red neuronal de retro-propagación) y por lo tanto será impredecible también la posibilidad de que el algoritmo quede atrapado en un mínimo local (no en el punto óptimo). Por lo tanto, el parámetro difusor q de la función probabilística de clustering es un elemento muy importante, no sólo porque convierte al algoritmo en difuso; básicamente controlando *qué tanto se pueden traslapar*[71] los clusters de la partición, sino también porque funciona como parámetro de manipulación de la superficie de error. En [71] se exponen una secuencia de experimentos con los que se puede establecer una heurística sencilla para estimar un valor adecuado de q, dependiendo de la dimensión m de los datos y el número k de clusters. Se puede observar en la Figura 2.11, bajo determinadas condiciones (m, k), que a medida que q alcanza un valor de 6, en la superficie de error se desvanecen los mínimos locales y la magnitud del error disminuye, asegurándose así la convergencia del algoritmo en el punto óptimo. Es entonces que se verifica que:

> *''Ningún algoritmo de clustering difuso es mejor que el mejor de los rígidos en términos de $J_q(\Theta, U)$; sin embargo, cuando $q > 1$ se suscitan algoritmos difusos que llevan a obtener valores de $J_q(\Theta, U)$ mejores de los que se obtienen con el mejor algoritmo rígido''*[51].

Minimización de $J_q(\Theta, U)$

Se busca minimizar J_q con respecto de las variables independientes Θ y U. Para tal efecto es necesario suponer que $J_q(\Theta, U)$ es la ecuación que describe a un sistema físico de tipo

conservativo, del cual se busca obtener la posición de *mínima energía* (en coordenadas generalizadas; es decir, independientes del observador) para su conjunto de partículas dinámicas; dicha posición, convenientemente para un agrupador de datos, sería el punto óptimo. Bajo dicha suposición, se tiene la encomienda de resolver un problema de optimización sujeto a las restricciones lineales (2.6). Para tal fin, es posible utilizar la teoría de Lagrange[51] para sistemas conservativos, en donde, considerando ya como función objetivo a $J_q(\Theta, U)$, se establece que se puede construir una función Lagrangiana a partir de J_q de la siguiente forma:

$$\mathscr{J}_q(\Theta, U, \boldsymbol{\lambda}) = J_q(\Theta, U) + \boldsymbol{\lambda} g_0(U), \tag{2.13}$$

donde $g_0(U)$ es la restricción (2.6b) de la función objetivo, adecuada con las condiciones de Karush-Kuhn-Tucker (KKT)[51], y $\boldsymbol{\lambda} = (\lambda_1, ..., \lambda_i, ..., \lambda_n)$ es el vector de multiplicadores de Lagrange cuyas componentes establecen escalares, uno para cada $\mathbf{x}^{(i)}$, tales que se asegura que existe una combinación lineal del gradiente que represente un extremo local en el punto U_a de la superficie descrita por (2.13); es decir, que se cumpla:

$$\left.\frac{\partial J_q(\Theta, U)}{\partial \Theta}\right|_{U_a} + \lambda_i \left.\frac{\partial g_0(U)}{\partial \Theta}\right|_{U_a} = 0$$

$$\left.\frac{\partial J_q(\Theta, U)}{\partial U}\right|_{U_a} + \lambda_i \left.\frac{\partial g_0(U)}{\partial U}\right|_{U_a} = 0.$$

Si el sistema de ecuaciones anterior se cumple de modo que $\exists \lambda_i \in \mathbb{R}^+$ para cada $x^{(i)}$, entonces U_a es el punto (punto óptimo) para el cual se define la posición de mínima energía del sistema conservativo en cuestión. De acuerdo con ésto, se obvian por simplicidad los cálculos de las derivadas parciales y la resolución del sistema de ecuaciones resultante, de forma que se tiene a continuación sólo el resultado de dichos cálculos; la ecuación de actualización para cada elemento de U:

$$\hat{u}_{ij} = \left[\frac{1}{\sum_{l=1}^{k} \left(\frac{f(\mathbf{x}^{(i)}, \boldsymbol{\rho}_j)}{f(\mathbf{x}^{(i)}, \boldsymbol{\rho}_l)} \right)} \right]^q, \tag{2.14}$$

además, la ecuación de actualización para los elementos de Θ:

$$\hat{\boldsymbol{\rho}}_j = \frac{\sum_{i=1}^{n} \hat{u}_{ij}^q \mathbf{x}^{(i)}}{\sum_{i=1}^{n} \hat{u}_{ij}^q}. \tag{2.15}$$

Si en la ecuación (2.14) f es la distancia euclidiana $d_2(\bullet, \bullet)$, entonces (2.15) es la ecuación de actualización de los centros paramétricos para el algoritmo básico de clustering *fuzzy c-means*.

2.8. Redes neuronales artificiales

Las RNAs (*Redes Neuronales Artificiales*) son un enfoque inspirado en la composición celular del cerebro humano, así mismo en su comportamiento cuando se trata de aprender y reconocer patrones. Para llevar a cabo tareas con alto grado de complejidad, el cerebro humano requiere de mucho[7] tiempo de entrenamiento, a diferencia de una computadora, a la que sólo le basta con que cada instrucción de un programa se ejecute, sin error alguno. No se hará algo más ni algo menos que lo que el programa indica con cada instrucción.

En una computadora a la que se programa con algún algoritmo complejo, la forma de llevar a cabo dicho programa es mediante la ejecución secuencial de comandos que componen el algoritmo con una dispocisión discretizada: las instrucciones que se ejecutan al final, y su resultado, dependen de que se ejecuten las que están al principio, de modo que si una sola instrucción falta o es errónea al inicio o en medio del programa, el resultado se pierde completamente. Actualmente se tienen máquinas con múltiples procesadores extremadamente rápidos; sin embargo, cada uno de ellos debe actuar básicamente de la forma tradicional[39].

El cerebro humano, a diferencia de una máquina, no tiene mucha rapidez al hacer operaciones aritméticas, sin embargo, cuenta con la suficiente capacidad de procesamiento para hacer ponderaciones, conexiones y desconexiones que permitan llevar a cabo correctamente actividades precisas que dependan de información mínima o con alto grado de ambigüedad. El procesamiento de la información se lleva a cabo en paralelo por un gran conjunto especializado de neuronas cuyas conexiones se reconfiguran continuamente para obtener conocimiento nuevo o reforzar el que ya existe[40]. Esto último es una ventaja que proporciona una robustez excepcional, a tal grado que en caso de ser destruidas muchas unidades, el efecto apenas es perceptible.

Estas últimas afirmaciones son las que motivan a que sea considerado el enfoque de las redes neuronales artificiales en el desarrollo de este trabajo. Es posible que el tiempo de entrenamiento requerido, como ya se dijo, sea considerablemente grande, sin embargo, se puede confiar en que las deducciones del algoritmo sean lo suficientemente precisas. Además, es un campo de estudio que siempre brinda la oportunidad de aportar y adquirir conocimientos nuevos, y plantean un gran reto en cuanto que nadie ha logrado establecer una teoría suficientemente concreta sobre su comportamiento. Por lo tanto, no queda hasta el momento más que definir algunas heurísticas prácticamente basadas en conocimiento tácito.

[7] Si se piensa en tareas complejas como leer, escribir o jugar con habilidad algún deporte.

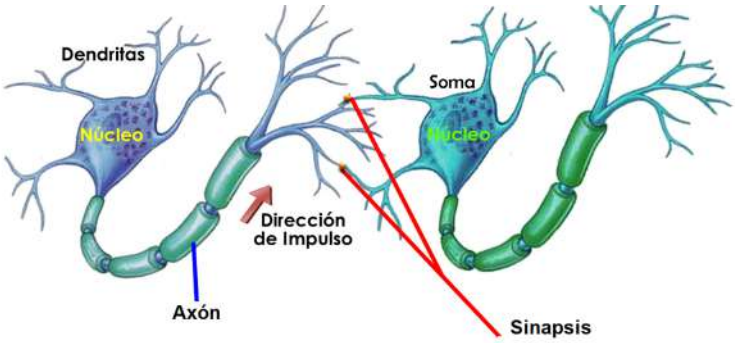


Figura 2.12: Disposición simplificada de una neurona orgánica. Esta Figura y la Figura 2.13, son versiones modificadas tomadas de `http://blog.cicei.com/erubio/2012/03/25/del-genoma-al-conectoma-%C2%BFsoy-yo-mi-conectoma/`

2.8.1. Modelo neuronal orgánico

El sistema nervioso del cuerpo humano está compuesto por unidades fundamentales llamadas *neuronas* (En promedio el sistema nervioso humano puede estar constituido hasta por 10^{11} neuronas[39]), las cuales están organizadas de manera muy compleja, de modo que se hallan en constante comunicación unas con otras. En conjuntos específicos, llevan a cabo tareas específicas tan sencillas como mantener respirando al cuerpo, mover un dedo para presionar una tecla o retirar el brazo tras sentir un pellizco.

Una de estas unidades fundamentales está compuesta, principalmente, por su cuerpo o *soma*, unas ramificaciones cortas que rodean al soma llamadas *dendritas*, una más larga y de espesor variado llamada *axón* (véase la Figura 2.12). Cada axón tiene al final otras ramificaciones llamadas *terminales pre-sinápticas*, en las cuales circulan cápsulas que contienen sustancias químicas llamadas *neurotransmisores* cuya función es el transporte de información desde una neurona a otra. Los neurotransmisores son partículas químicas que permanecen encapsuladas (en *cápsulas presinápticas*) antes de liberarse al final de la terminal pre-sináptica, para después permearse hacia el exterior de la célula, viajando a través de la *hendidura sináptica*, que existe entre la terminal pre-sináptica y la superficie pos-sináptica de la dendrita de la neurona que recibe a las partículas liberadas. Esta dendrita, como lo muestra la Figura 2.13, funciona entonces como terminal neuro-receptora que, en conjunto con las terminales presinápticas, hacen circular mensajes (en forma de impulsos de partículas químicas) que viajan a través de millones de neuronas, produciéndose así un proceso de comunicación intercelular llamado *sinapsis*, mediante el cual las células del cerebro procesan información y son configuradas para memorizar[40]. Una neurona puede tener hasta 10 000 entradas[39].

Dichas configuraciones (cambiantes con el tiempo) dependen del estado de cada neurona. Éstas pueden encontrarse en estado inhibitorio o de excitación. El primero sólo quiere decir que la transferencia de neurotransmisores tiene cierto grado de deficiencia o que no existe tal transferencia a través del axón y el segundo estado es una situación contraria; es decir,

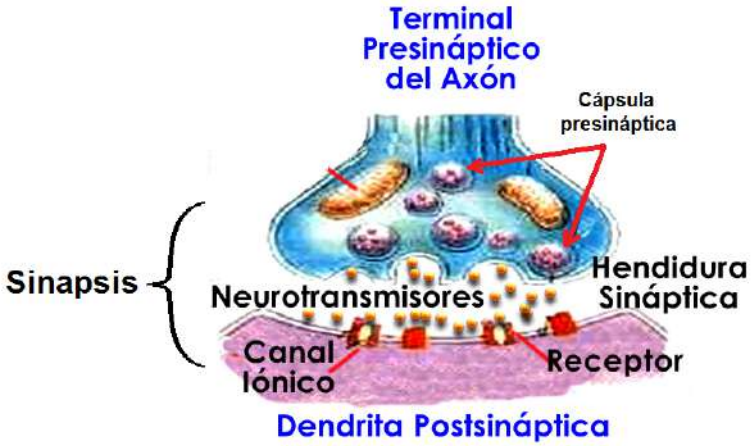


Figura 2.13: Proceso sináptico de las neuronas orgánicas.

la transferencia de neurotransmisores tiene cierto grado de eficiencia, de modo que hay la posibilidad de contribuir para inducir un estado exitatorio en otra neurona.

2.8.2. Modelo neuronal artificial

Una neurona artificial es un modelo matemático simplificado que simula el comportamiento de la células del sistema nervioso humano[39][41]. Ya hemos visto los elementos que forman el modelo orgánico, por lo que ahora sólo se verá cómo cada uno de ellos se relaciona con el modelo artificial, observando sus nombres y analogía funcional con el modelo orgánico. Pero primero se recorrerá la línea del tiempo que da cuenta sobre los escalones más importantes que el modelo artificial ha escalado, hasta el llegar al modelo de interés para este trabajo: *Redes Neuronales de Retropropagación* o Perceptrón Multi-Capa (*Multi-Layer Perceptron*).

Historia

En 1943, Warren McCulloch y Walter Pitts[40] fueron los primeros investigadores en proponer un modelo artificial del comportamiento de una neurona orgánica; sin embargo, este modelo sólo procesaba datos que se le proporcionaran introducía como datos de entrada. Unos años más tarde, en 1949, Donald O. Hebb propuso las bases para el desarrollo del primer sistema de aprendizaje para una red neuronal, plasmadas en su gran obra: *The Organization of Behavior: A Neuropsychological Theory*[42], una de las teorías más consistentes, todavía, sobre este respecto. Después, en 1957, Frank Rossenblatt logra entrenar una red neuronal monocapa usando el trabajo de McCulloch-Pitts y Hebb, obteniendo así el *Perceptrón*. Este último tiene la gran desventaja de que sólo puede resolver problemas que involucren espacios de entrada *linealmente separables* (poco comunes en situaciones reales) por lo que,

en 1969, Minsky y Papert cuestionan severamente su utilidad, de manera que la investigación relacionada con la redes neuronales, se mantuvo pasiva durante un poco más de 10 años. Algunos investigadores seguían trabajando y dichos cuestionamientos serían revocados: en 1982 el afán por la investigación acerca del enfoque neuronal, toma un nuevo respiro con los trabajos de John Hopfield y Teuvo Kohonen que desarrollaron redes neuronales capaces de resolver problemas linealmente no separables, a través del entrenamiento por mapas asociativos y reglas de competencia. No es hasta 1986 cuando David E. Rumelhart, *et al.* usan una *red de perceptrones* con más de una capa para presentar la *regla delta generalizada*, misma que se basa en la retro-propagación del error obtenido en la salida de la red. Dicho enfoque asume que cada capa oculta de la red neuronal, contribuye con el error de salida, de manera que en la proporción de dicha constribución se lleva a cabo ajuste de los parámetros de cada capa. Este enfoque es en la actualidad, uno de los más usados y que en este trabajo se ha considerado como objeto de estudio.

El modelo matemático

La base principal del modelo artificial es su concepción *bioinspirada*. Según Hebb[42], el axón de una neurona transmite pulsos eléctricos hacia las dendritas de otras neuronas subsecuentes, por medio de una interconexión llamada sinapsis, la cual mejora o deteriora la transmisión de información desde dicho axón, hasta la dendrita que recibe la señal. Estos pulsos, procedentes de n neuronas anteriores y que ya han pasado a través de cada sinapsis (ya circulan por las dendritas de la neurona actual), se pueden modelar, según [40], como un patrón de entrada $\mathbf{x}^{(i)}$, cuyas componentes x_j se presentan ponderadas a la neurona actual (véase la Figura 2.14). Este ponderado significa que cada componente es amplificada o atenuada mediante un factor multiplicativo llamado *peso sináptico*, denotado por $w_{j,k}$, antes de que la neurona realice alguna operación con ella. La operación efectuada consiste en la sumatoria de todas las entradas ponderadas. La neurona artificial deberá tener la posibilidad de cambiar desde un estado inhibitorio a uno de excitación, de modo que se tiene un *umbral de excitación*, el cual debe ser superado por la suma ponderada para que la excitación de la neurona tenga lugar. Se verá en la Sección 2.8.5, que umbral de exitación funciona como parámetro de desplazamiento de la *frontera de clasificación*. Una vez que la neurona ha recibido las componentes del patrón de entrada, el valor de la suma ponderada se pasa como parámetro a una *función de activación*, cuyo resultado es la salida y_k de la neurona. La finalidad principal de la función de activación es limitar la variabilidad y la magnitud de la salida de la neurona, de modo que la convergencia de su entrenamiento se lleve a cabo en el menor tiempo posible.

El proceso anterior se puede expresar mediante la ecuación (2.16), donde se ve formalmente

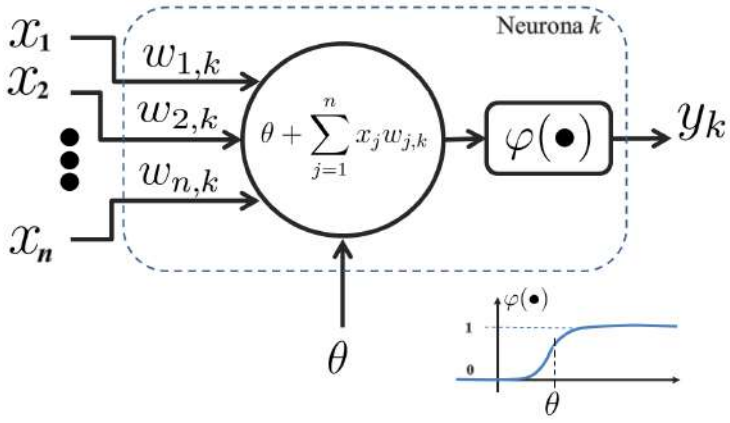


Figura 2.14: Modelo artificial de una neurona.

cada parámetro que interviene en la operación de una neurona artificial:

$$y_k = \varphi\left(\theta_k + \sum_{j=1}^{n} x_j w_{j,k}\right), \tag{2.16}$$

donde: k es el índice de la k-ésima neurona de una capa, x_j es la j-ésima componente del patrón de entrada $\boldsymbol{x}^{(i)}$, $w_{j,k}$ es el el j-ésimo peso sináptico de la neurona k, θ_k es el umbral de excitación, y_k es la salida y $\varphi(\bullet)$ es la función de activación. En la siguiente sección se verá lo referente a las funciones de activación más usadas.

Una neurona puede resolver una gran variedad de problemas debido a su capacidad de adaptabilidad, esto último permite interconectar varias neuronas para formar una *red neuronal*, la cual es capaz de resolver problemas con características no lineales.

2.8.3. Funciones de activación

En este trabajo se ha implementado una red cuyos nodos de procesamiento cuentan con la función de activación *tangente hiperbólica*, debido a que los elementos del espacio de entrada tienen valores bipolares. Usar esta función proporciona mayor adaptabilidad a dicho espacio de entrada y está definida en la ecuación (2.17):

$$\varphi(\nu) = \frac{e^{\nu} - e^{-\alpha\nu}}{e^{\nu} + e^{-\alpha\nu}} \quad \in [-1, 1]; \;\; \alpha \in \mathbb{R}, \tag{2.17}$$

donde α es el parámetro de pendiente de la curva descrita por (2.17) y ν es la suma ponderada de la neurona. Como ya se había mencionado, uno de los propósitos de $\varphi(\bullet)$ es suavizar la

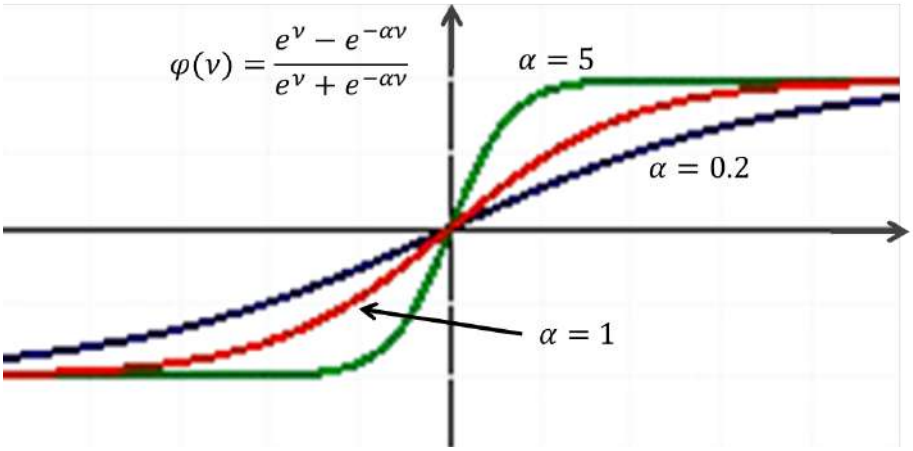


Figura 2.15: Función de activación tangente hiperbólica para los nodos de procesamiento de una red neuronal.

variabilidad de la salida del nodo; pero además, limita su valor. Se observa en la Figura 2.15, que la pendiente de la gráfica varía según sea el valor de α. Si α es grande (i. e. $\alpha > 2$), el entrenamiento puede volverse inestable, debido a la variabilidad y brusquedad que ésto induce en la salida; por otro lado, si el valor de α es pequeño (i. e. $\alpha < 0.7$), el entrenamiento podría tornarse lento, debido a cambios poco significativos en la salida de la neurona. Lo ideal es establcer un valor intermedio, lo que alienta la convergencia del entrenamiento, permitiendo cambios progresivos en la salida de la neurona. Lo último, induce a su vez la variabilidad correcta en el ajuste de los pesos sinápticos durante el entrenamiento.

Se tiene otra de las funciones más usadas: la función $sign(\bullet)$ (función *signo*) que, como se verá más adelante, es empleada en el cómputo de salidas del perceptrón binario. Su expresión se halla definida en la ecuación (2.18); su contradominio es bipolar, característica que la hace más conveniente que la *función escalón* (o función *Heaviside*).

$$sign(\nu) = \begin{cases} 1 & si \quad \nu \geq \theta \\ -1 & si \quad \nu < \theta \end{cases} \tag{2.18}$$

2.8.4. Estructura de las redes neuronales

Se ha expuesto antes que las unidades elementales de procesamiento (neuronas o *nodos*) se pueden interconectar para formar una red y según se realicen estas conexiones, existen tres maneras principales de tipificar a las redes neuronales según su estructura[43]:

- *Por número de capas.* Se puede decir que una red neuronal es *multicapa* si sus nodos están dispuestos en diferentes niveles, cada nivel con un número determinado de ellos.

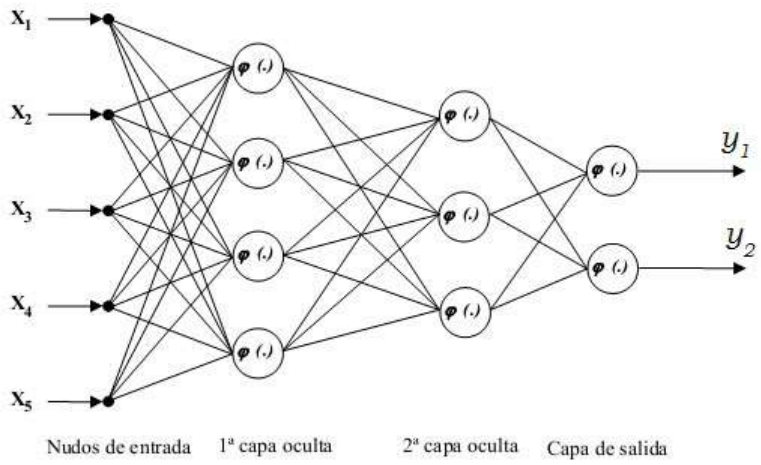


Figura 2.16: Estructura de una red neuronal multicapa (una capa de entrada con cinco nodos, dos capas ocultas de cuatro y tres nodos respectivamente, y una capa de salida de dos nodos: 5-4-3-2). Figura modificada de [46].

Estos niveles se llaman capas, dichas capas pueden ser: una al inicio llamada *capa de entrada*, una o varias capas intermedias llamadas *capas ocultas* y una más al final llamada *capa de salida*. En la mayoría de las ocasiones, la capa de entrada no es considerada al contar el número de capas de la red, puesto que cada nodo en ella sólo funciona como distribuidor de la información que recibe como entrada, hacia los nodos de la capa siguiente. Véase en la Figura 2.16 un ejemplo de una red multicapa.

- *Por su patrón de conexión.* Si se tienen n nodos en una capa y la salida de cada uno, sin excepción, está conectada como entrada de todos los nodos de la siguiente capa, entonces se dice que la red está *totalmente conectada*, como es el caso de la figura 2.16. Para el caso en que una sóla de estas salidas no se conecte a alguna de las entradas de algún nodo de la siguiente capa, se dice que la red es *parcialmente conectada*.

- *Por su flujo de información.* Ésto tiene que ver también con las conexiones entre nodos; sin embargo, esta tipificación se debe a la dirección del flujo de información de la salida de los nodos en la red. Si las salidas de todos los nodos se conectan hacia cualquier nodo de las capas siguientes se tiene una red de alimentación directa (*feedforward*). Por otro lado, si en una red dada la salida de algún nodo se conecta como entrada de sí mismo, de algún nodo de la misma capa o de una capa previa, se tiene una red retroalimentada (*feedback*) como la de la Figura 2.17.

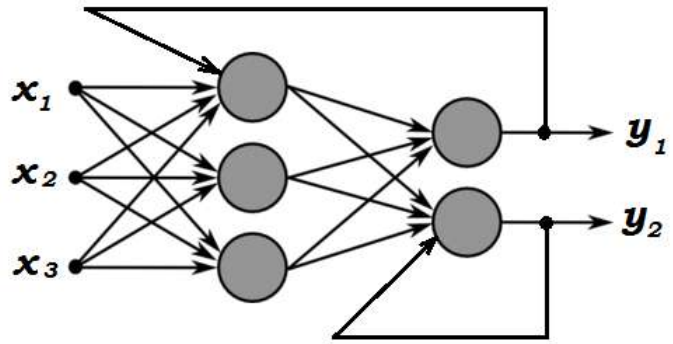


Figura 2.17: Estructura de una red neuronal feedback.

2.8.5. El perceptrón

Puesto que se busca fundamentar la posterior exposición sobre las redes neuronales de retropropagación, es necesario exponer primero el funcionamiento de una red basada en el perceptrón binario (también llamado *Perceptrón simple*) propuesto por Frank Rosenblatt en 1957[45]. Una red de retropropagación, está constituida por varios perceptrones, dispuestos en una o más capas; de aquí el nombre ‘‘Perceptrón Multi-Capa’’.

El perceptrón binario es el modelo de neurona propuesto por McCulloch-Pitts[40], pero con la característica adicional de contar con un algoritmo de *entrenamiento local*, lo que significa que sólo es posible ajustar los pesos sinápticos de una red monocapa. En esta sección se tratará el caso de una neurona (Véase la Figura 2.18). Si se tuviesen más nodos de procesamiento como se observa en la Figura 2.19, las condiciones que rigen al perceptrón simple son fáciles de generalizar[46]; de manera que en ocasiones que así lo requieran, se mecionará que se trata con una red de neuronas.

El perceptrón surge de la necesidad de contar con un elemento computacional con la capacidad de adaptarse a la resolución de problemas de clasificación para dos conjuntos de objetos C_1 y C_2, que sean separables mediante un *hiperplano* que pase entre ellos en el espacio de entrada; es decir, que sean *linealmente separables*. Dada esta condición, el perceptrón cuenta con una salida binaria, tal como se muestra en la Figura 2.18.

$$y = \begin{cases} 1 & si \quad \sum_{j=1}^{n} x_j w_j \geq \theta \\ -1 & si \quad \sum_{j=1}^{n} x_j w_j < \theta \end{cases} \tag{2.19}$$

Según se muestra en la ecuación (2.19), la salida tendrá un valor de 1 si la suma ponderada iguala o supera en valor al umbral de activación θ, y tendrá un valor de -1 si la

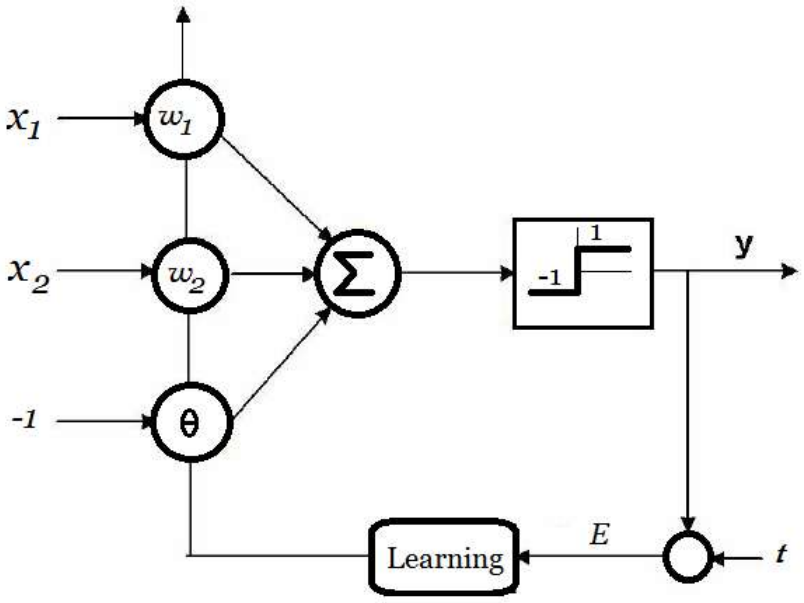


Figura 2.18: Estructura de un perceptrón simple.

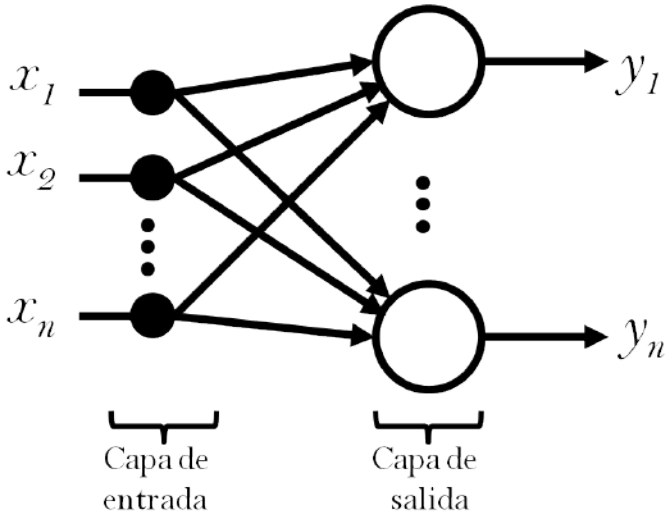


Figura 2.19: Estructura de una red con n perceptrones simples.

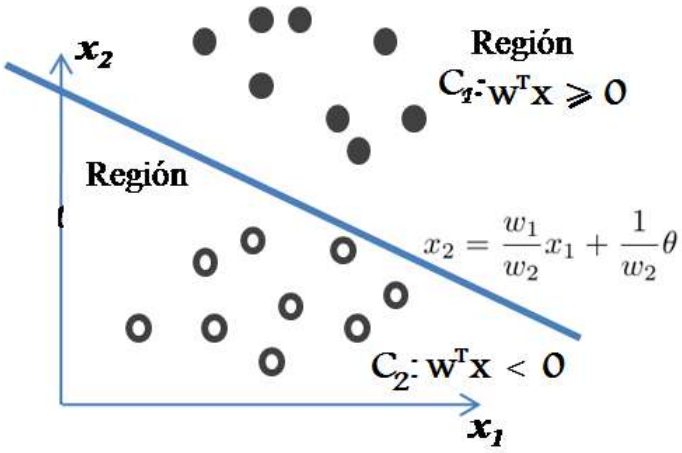


Figura 2.20: Espacio de entrada bidimensional para el perceptrón simple. Los puntos rellenos son de la clase C_1 y los vacíos son C_2.

suma ponderada permanece por debajo de dicho umbral. Nótese entonces, la simpleza del procesamiento realizado por una neurona de este tipo.

Ya se había descrito este procesamiento básico en la Sección 2.8.2 (‘‘Modelo neuronal artificial’’) y nociones sobre el entrenamiento en la Sección 2.3.1 (‘‘Tipos de aprendizaje’’), de modo que ahora sólo se formalizan estos conceptos para el perceptrón simple. Tomando en cuenta la ecuación (2.19), que es la definición por tramos de la función $sign(\bullet)$, se ve fácilmente que la ecuación (2.16) se convierte en (2.20) para proporcionar la salida del perceptrón binario:

$$y_k = sign\left(\sum_{j=1}^{n} x_j w_{j,k} + \theta_k\right) \tag{2.20}$$

y en forma vectorial:

$$y_k = sign(\mathbf{w}_k^T \mathbf{x}),$$

donde: $\mathbf{w}_k^T = (w_{1,k}, w_{2,k}, ..., w_{j,k}, ..., w_{n,k})^T$ es el vector de pesos sinápticos de la neurona k, además $\mathbf{x} = (x_1, x_2, ..., x_j, ..., x_n) \in X$ es el patrón de entrada y X el espacio de entrada. Nótese que $\sum_{j=1}^{n} x_j w_{j,k} + \theta_k$ es el hiperplano que separa las regiones de clasificación, de manera que para el caso de un espacio de entrada bidimensional, el hiperplano correspondiente es una línea recta descrita por:

$$x_1 w_1 + x_2 w_2 + \theta = 0. \tag{2.21}$$

Se observa que la pendiente de (2.21) está dada por la razón $\frac{w_1}{w_2}$, así mismo el desplazamiento de la recta sobre el eje vertical estaría dado por θ, tal como se observa en la Figura 2.20. Con esto ya se tiene parametrizada la línea divisoria de clases (o frontera de clasificación). El algoritmo de entrenamiento que se verá en la siguiente sección, se encarga de ajustar estos parámetros.

Entrenamiento del Perceptrón

Inicialmente se replantea la ecuación (2.20) como se indica en la Figura 2.18; es decir, se considera al umbral θ como un peso sináptico más, el cual pondera a un valor constante de 1. Con este planteamiento, supóngase la presencia de un patrón de entrada $\mathbf{x}^{(i)} = (1, x_1, x_2, ..., x_j, ..., x_n)$; $i \in I \subset \mathbb{Z}^+$ y el vector de pesos sinápticos $\boldsymbol{w}_k = (\theta, w_{1,k}, w_{2,k}, ..., w_{j,k}, ..., w_{n,k})^T$ de la k-ésima neurona, de la red de perceptrones[8] que se quiere entrenar. De manera que se tiene la suma ponderada de un perceptrón en su forma vectorial:

$$\nu(l) = \boldsymbol{w}^T \mathbf{x}^{(i)}.$$

En virtud de que la partición del espacio de entrada de un perceptrón simple es binaria, se establecen dos regiones de clasificación C_1 y C_2 que estarían definidas por la siguientes reglas $\forall \mathbf{x}^{(i)} \in X$:

$$\begin{array}{lll} \text{Si } \boldsymbol{w}^T \mathbf{x}^{(i)} > 0 & \Rightarrow & \mathbf{x}^{(i)} \in C_1 \\ \text{Si } \boldsymbol{w}^T \mathbf{x}^{(i)} \leq 0 & \Rightarrow & \mathbf{x}^{(i)} \in C_2. \end{array} \tag{2.22}$$

Dichas reglas, plantean el problema que se busca resolver con la ayuda del algoritmo de entrenamiento del perceptrón, pues con esto los parámetros de (2.20) quedan libres[9] y por lo tanto el algoritmo de entrenamiento debe ajustarlos, con la finalidad de que la regla (2.22) describa correctamente el comportamiento de la neurona.

Dado el planteamiento anterior, llámese conjunto de entrenamiento a $\Gamma = \Gamma_1 \cup \Gamma_2$, donde $\Gamma_1 = \{\mathbf{x}^{(r)} | r \in R \subset \mathbb{Z}^+\} \subset X$ y $\Gamma_2 = \{\mathbf{x}^{(s)} | s \in S \subset \mathbb{Z}^+\} \subset X$; $R \cup S = I$, $R \cap S = \emptyset$. Se quiere que los subconjuntos de Γ, sean equivalentes (que sus elementos sean asignados) a C_1 y C_2 respectivamente, por lo tanto, el entrenamiento debe utilizar esta información en su l-ésima iteración, quedando descrito de la siguiente manera[46]:

1. Si el i-ésimo elemento ($i \in R \cup S$) de Γ, es clasificado de manera correcta por el perceptrón, según el vector de pesos sinápticos $\boldsymbol{w}(l)$ en la l-ésima iteración, entonces no hacer cambios y mantener $\boldsymbol{w}(l)$ según el comportamiento actual:

$$y_k(l) = \begin{cases} 1 & \text{si} \quad \boldsymbol{w}^T(l)\mathbf{x}^{(r)}(l) > 0 \\ -1 & \text{si} \quad \boldsymbol{w}^T(l)\mathbf{x}^{(s)}(l) \leq 0 \end{cases}$$

2. En otro caso, actualizar el vector de pesos sinápticos de acuerdo a la siguiente regla:

$$\begin{array}{lll} \text{si } \boldsymbol{w}^T(l)\mathbf{x}^{(r)}(l) \leq 0 & \Rightarrow & \boldsymbol{w}(l+1) = \boldsymbol{w}(l) + \eta \mathbf{x}^{(r)}(l) \\ \text{si } \boldsymbol{w}^T(l)\mathbf{x}^{(s)}(l) > 0 & \Rightarrow & \boldsymbol{w}(l+1) = \boldsymbol{w}(l) - \eta \mathbf{x}^{(s)}(l), \end{array} \tag{2.23}$$

   regresar al paso 1) y verificar.

[8] Ya se mencionó que una red de perceptrones, para este apartado y para este algoritmo de entrenamiento, es monocapa.

[9] En la literatura más reconocida, a los pesos sinápticos de una red neuronal se les llama también *parámetros libres*[46].

En la regla (2.23), η es el *parámetro de aprendizaje* del perceptrón y determina qué tan rápido o lento será el aprendizaje. Según [46], su valor (siempre y cuando sea positivo) no afecta la separabilidad de las clases, dado que al establecerlas se ha buscado que lo sean. Generalmente se usan valores de $\eta \in (0, 1]$, de modo que el vector de pesos sinápticos no tenga diferencias desmedidas entre cada iteración. La prueba de convergencia de este algoritmo se puede consultar en [39][46] y se llama *teorema de convergencia del perceptrón.*

Se tiene otra forma de plantear la regla (2.23). En casos más generales en los que se tiene un conjunto de salidas deseadas o ejemplos $\{t_i\}_{i\in I} \subset \mathbb{R}$ (suponiendo una neurona) y cada t_i está emparejado con cada $\mathbf{x}^{(i)}$, entonces la salida del perceptrón no es necesariamente binaria ($y_k \in \mathbb{R}$) y por lo tanto:

$$\boldsymbol{w}(l+1) = \boldsymbol{w}(l) + \eta[t_i(l) - y(l)]\mathbf{x}^{(i)}(l). \tag{2.24}$$

En (2.24), se puede inferir que para un perceptrón simple las salidas deseadas serían:

$$t_i(l) = \begin{cases} 1 & \text{si } i \in R \\ -1 & \text{si } i \in S \end{cases}$$

y $t_i(l) - y(l)$ es el error $e(l)$ calculado en la iteración l, el cual será usado para saber si el algoritmo se detiene. La Ecuación (2.24), se conoce comúnmente como *Regla Delta.*

2.9. Extracción de información

Cuando se está supervisando el estado de un procedimiento complejo, el sistema de supervisión detecta cambios en el estado de los descriptores de tal procedimiento, de modo que si estos descriptores son, por sí sólos, cambios temporales, el algoritmo de supervisión se puede confundir detectando cambios que no aportan información o que aportan información irrelevante a la tarea de supervisión.

Para resolver esta problemática, es necesario implementar un sistema de *abstracción de señales* cuya finalidad principal sea el procesamiento de dichas señales para extraer de ellas únicamente información que sea de utilidad para el sistema supervisor. Existen diferentes métodos de abstracción de señales; sin embargo, ninguno puede ser aplicado a todos los casos. Dependerá de las características de cada señal en específico y del tipo de información que se quiera conocer de ella.

Se tienen a continuación diferentes técnicas de abstracción, mismas que pueden ser aplicadas en distintas circunstancias[81, 86]:

- Análisis espectral por Transformada de Fourier.

- Análisis mediante Transformada WaveletFourier.
- Modelos de autoregresión.
- Filtraje.
- Análisis en frecuencia-tiempo (Transformada de clase Cohen, Distribución de Wigner-Ville y Distribución de Choi-Williams).
- Análisis mediante algoritmos de clustering y RNAs.
- Análisis estadístico de orden superior.
- Ganancia de información y Entropía de Shannon.

Dado que se ha obtenido el conocimiento necesario del sistema que se va a supervisar con la implementación propuesta en este trabajo, las técnicas de extracción de información que fueron usadas en este trabajo son: el anális espectral por Transformada de Fourier y la media aritmética de un conjunto de muestras. Este último es un cálculo muy sencillo, por lo que no se abundará en ello.

2.9.1. Análisis espectral por transformada de Fourier para señales no estacionarias

El análisis de Fourier es una herramienta muy potente para estudiar la naturaleza de una señal, ya que permite interpretarla en términos de componentes de frecuencia; de forma que se dice que se hace un análisis en el dominio de la frecuencia. La Transformada de Fourier (*FT*, por sus siglas en inglés) considera que cualquier señal, por compleja que ésta sea, puede ser representada como una suma de señales seno. Con base en esta consideración, es posible obtener las componentes de frecuencia de la señal, las cuales son términos de dicha suma, tal como se observa en la Figura 2.21.

Se sabe que la DFT es aplicable para señales estacionarias y que, dado que la naturaleza no es estacionaria, se debe segmentar la señal objeto de estudio en ventanas que contengan un número fijo de muestras, con el fin de mantener estacionario un lapso adecuado de tiempo (proceso conocido como *windowing*). Se puede continuar de esta manera si se quiere analizar una señal durante mucho tiempo: ‘‘congelando’’ la señal en ventanas temporales de muestras para volver estática la señal objetivo por lapsos de tiempo de duración adecuada, de modo que se puedan analizar de manera individual y se pueda obtener la magnitud de la componente de máxima energía, la cual es de interés como característica del espacio de entrada de un clasificador. Dado que se están analizando lapsos de una señal para un número finito de muestras en tiempo discreto, la FT en este caso se llama STFT (*Short Time Fourier*

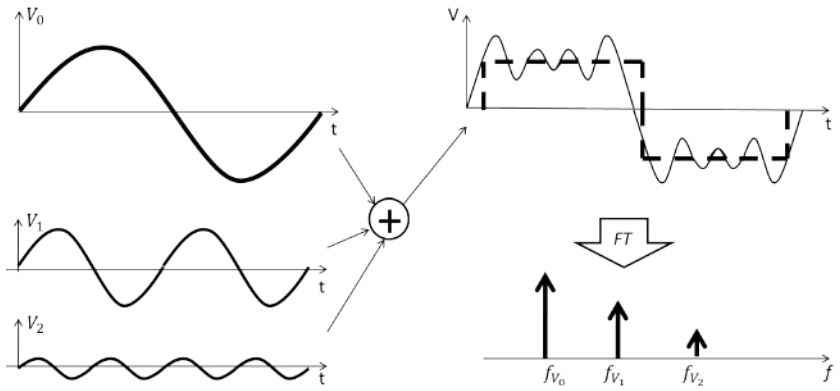


Figura 2.21: Ejemplo de la construcción de una señal cuadrada mediante una suma de señales seno, así como la visualización de las componentes de frecuencia en el dominio de la frecuencia. Véase que no se está ilustrando precisamente la transformada de Fourier en tiempo continuo de la señal, sin embargo, se muestra el objetivo que se busca alcanzar mediante su aplicación.

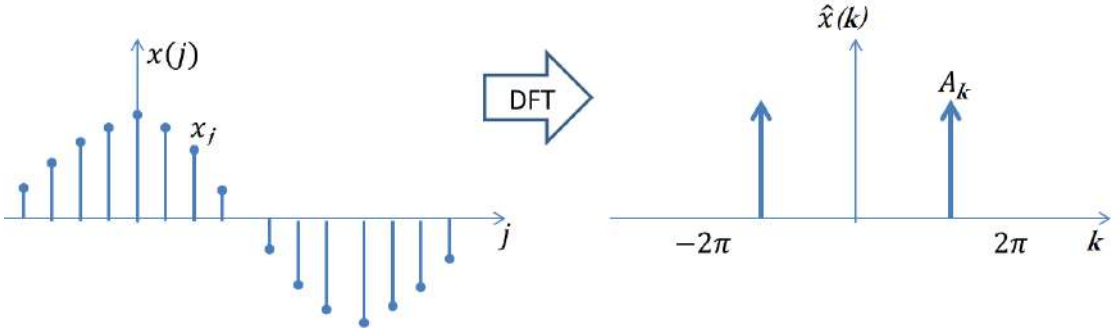


Figura 2.22: Ejemplo de la aplicación de la DFT a una señal coseno.

Transform); sin embargo, con esta variante sólo el tiempo es discreto. En una computadora toda información que se procesa debe ser discreta; por lo tanto, cuando se usa la FT para tiempo y frecuencia discretos, se conoce como FFT (*Fast Fourier Transform*)[86].

En vista de que sólo se busca ilustrar de manera muy concreta la metodología llevada a cabo para obtener información útil sobre las señales que se van a procesar con los algoritmos de reconocimiento de patrones, se define directamente la transformada de Fourier en tiempo discreto. Sea la serie de Fourier de una función f discretizada para una cantidad finita n de muestras:

$$f(j) = \left(\frac{2\pi j}{n}\right) = \sum_{k=0}^{n-1} A_k e^{2\pi i j k/n}; \quad j = 0, ..., n-1,$$

entonces se define a continuación la Transformada de Fourier en Tiempo Discreto[86]:

DEFINICIÓN **8** *Sea una función* $f(j) = a_j \ \forall j \in \{0, 1, 2, ..., n-1\}$*, entonces se llama Trans-*

formada de Fourier en Tiempo Discreto a la aplicación $\hat{f} : \mathbb{Z}^+ \to \mathbb{C}$ *tal que* $\hat{f}(k) = A_k \ \forall k \in \{0, 1, 2, ..., n-1\}$ *donde:*

$$a_j = \sum_{k=0}^{n-1} A_k e^{2\pi ijk/n}; \quad i = \sqrt{-1}$$

y a su vez cada

$$A_k = \frac{1}{n} \sum_{j=0}^{n-1} a_j e^{-2\pi ijk/n},$$

se llama coeficiente de Fourier.

En conclusión, si se aplica la DFT a una señal muestreada, por ejemplo:

$$x(j) = Acos\left(\frac{2\pi fj}{F_s} + \phi\right),$$

donde: A es la amplitud de la señal, f es la frecuencia, F_s es la frecuencia de muestreo y ϕ es la fase, entonces se tendría la información que se observa en la Figura 2.22, de donde si se toma en cuenta la frecuencia de muestreo y el número de muestras por periodo de la señal original, se puede obtener fácilmente la componente de frecuencia de máxima energía, verificando la posición de los coeficientes de Fourier.

Estos cálculos son realizados de manera muy sencilla haciendo uso de una función muy poderosa llamada `SingleToneInfo(x, 1/Fs, A, f, fase)`, incluida en la librería de análisis avanzado de LabWindows/CVI, la cual regresa por referencia los parámetros A, f y ϕ requeridos para el análisis. La rutina dedicada a la extracción de información (desarrollada en LabWindows/CVI) obtiene y escribe el valor de f a una variable (*descriptor*) representativa de la información requerida de esta señal en particular, la cual posteriormente será procesada por el algoritmo de reconocimiento de patrones.

Capítulo 3

Consideraciones de implementación para redes neuronales de retro-propagación *(Back-Propagation neural networks)*

Por sencillez, muy comúnmente se conoce a una red de perceptrones como *Red neuronal de retro-propagación* aunque el nombre largo (correcto) de esta red es *Perceptrón multicapa entrenado por retro-propagación del error* de la cual ya hemos visto, en la Sección 2.8, los atributos que la caracterizan; de modo que ya estamos en condiciones de decir que tipo específico de red vamos a someter a pruebas: Usaremos una red neuronal multicapa, feedforward, totalmente conectada y con un algoritmo de entrenamiento supervisado por corrección de error (retro-propagación). Este tipo de red está basada en el perceptrón simple, por lo que también es conocida simplemente como Perceptrón Multicapa (*MLP: Multi-Layer Perceptron*).

En una red MLP, a diferencia del perceptrón simple, cada nodo tiene como función de activación no lineal, dado que se busca establecer una relación no lineal entre patrones de entrada y valores de salida. En este trabajo se implementará la función *tangente hiperbólica* como función de activación (mencionada en el capítulo anterior; Figura 2.15) en lugar de la función signo. Además, por contar con capas de nodos internas (capas ocultas, *hidden layers*), el perceptrón multicapa es capaz de disolver la limitante de su versión monocapa propuesta por Rosenblatt: separabilidad lineal necesaria de las clases en el espacio de entrada.

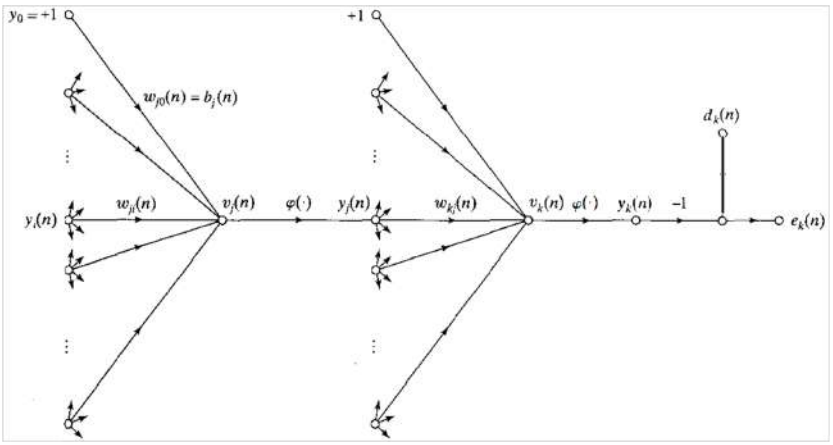


Figura 3.1: Flujo de señales a través de la red en alimentación directa (*feedforward*). Figura sacada de [46].

3.1. Entrenamiento de la red

El algoritmo de entrenamiento trabaja en dos direcciones a través de las conexiones de la red. Suponiendo que a la capa de salida se le llama $K = \{k\}_1^{n_O}$, a la capa oculta contigua $J = \{j\}_1^{n_h}$, a la capa de entrada I, con n_O nodos de salida y n_h nodos en la capa oculta y n_I nodos de entrada respectivamente y que $\nu(l)$ es la suma ponderada de cada nodo en la iteración l, entonces se tiene la descripción general del algoritmo de retro-propagación en alimentación directa y propagación hacia atrás:

Alimentación directa (*Feedforward direction*)

Desde los nodos de entrada hacia los de salida (véase la Figura 3.1), el patrón de entrada que se desea aprender $\boldsymbol{x}(l)$ se propaga desde los nodos de entrada hacia la capa oculta, generando el vector de salida $\boldsymbol{y}_J(l) = \boldsymbol{\varphi}(\boldsymbol{\nu}_J(l))$ en ésta última, hasta llegar a la capa de salida, en donde se tiene un vector de salida $\boldsymbol{o}(l) = \boldsymbol{y}_K(l) = \boldsymbol{\varphi}(\boldsymbol{\nu}_K(l))$. Dicho vector de salida es comparado con un vector de salidas deseadas $\boldsymbol{d}(l)$, de tal forma que se obtiene el vector de error:

$$\boldsymbol{e}(l) = \boldsymbol{d}(l) - \boldsymbol{o}(l). \tag{3.1}$$

Por último se calculan los vectores de pendientes $\boldsymbol{\varphi}'(\boldsymbol{\nu}(l))$ de la función de activación en cada capa de la red (K y J respectivamente). Estos vectores de pendientes (gradientes) serán usados por el algoritmo para llevar a cabo la propagación del error hacia atrás.

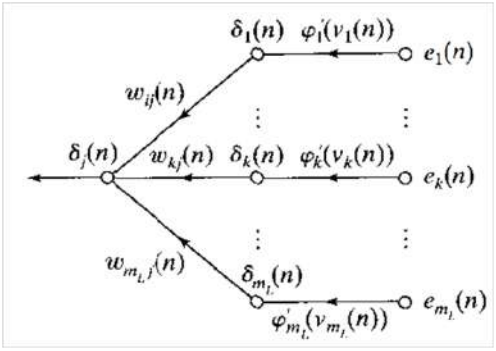


Figura 3.2: Flujo del vector de error a través de la red en propagación hacia atrás (*backward*). Figura sacada de [46].

Propagación hacia atrás (*Backward direction*)

Ahora desde la capa de salida hasta la capa de entrada se *propagan hacia atrás* las componentes del vector $\boldsymbol{e}(l)$. Como se muestra en la Figura 3.2, primero se multiplican elemento a elemento el vector de error por el vector de pendientes de la capa de salida (obtenido previamente). Esto produce un parámetro llamado vector *gradiente local* de la capa de salida, el cual se denota por $\boldsymbol{\delta}_K(l) = \boldsymbol{\varphi}'(\boldsymbol{\nu}_K(l))\boldsymbol{e}(l)$. Enseguida, éste último se multiplica vectorialmente por el vector de pesos sinápticos de cada nodo de la capa de salida y este producto a su vez, se multiplica elemento a elemento por el vector gradiente $\boldsymbol{\varphi}'(\boldsymbol{\nu}_J(l))$; obteniéndose así, las componentes del vector gradiente local de la siguiente capa: $\boldsymbol{\delta}_J(l) = \boldsymbol{\varphi}'(\boldsymbol{\nu}_J(l))\boldsymbol{\delta}_K^T\mathbf{w}_k$. De aquí en adelante, en caso de existir otras capas ocultas, ha de hacerse este último procedimiento, para obtener el siguiente vector gradiente local, supóngase, $\boldsymbol{\delta}_I(l) = \boldsymbol{\varphi}'(\boldsymbol{\nu}_I(l))\boldsymbol{\delta}_J^T\mathbf{w}_j$.

Para terminar la iteración l, se procede a actualizar los pesos sinápticos de acuerdo a lo que especifica la *regla delta generalizada* (cuya obtención se puede consultar, al igual que todo el procedimiento aquí descrito en [46]):

$$w_{j,i}^{(L)}(l+1) = w_{j,i}^{(L)}(l) + \alpha[w_{j,i}^{(L)}(l-1)] + \eta\delta_j^{(L)}(l)y_i^{(L-1)}(l). \tag{3.2}$$

En la ecuación (3.2), L es el número de capa en la cual se están actualizando los pesos sinápticos en la iteración l; η es la taza de aprendizaje, $\alpha[w_{j,i}^{(L)}(l-1)]$ es el término de ímpetu (momento) del aprendizaje (α es su constante de proporcionalidad), los subíndices j, i indican que el peso que se está actualizando está en una conexión que viene de la i-ésima neurona de la capa I hacia la j-ésima neurona de la capa J.

Tomando en cuenta a (3.1), se establece a continuación el criterio de evaluación para el aprendizaje de la red, el cual se usa a menudo como criterio de parada del algoritmo de

retro-propagación. Existen dos modalidades:

1. Para cada neurona de salida k se define el impacto de su error (*energía del error*[46]) como: $\frac{1}{2}e^2(l)$, de modo que si se suman estos escalares a través de la capa de salida K se tiene:
$$\xi(l) = \frac{1}{2}\sum_{K} e^2(l). \tag{3.3}$$
Además, se sabe que la red deberá aprender un conjunto de entrenamiento y supóngase que éste estría conformado por P patrones. El aprendizaje de la red será evaluado mediante el cálculo iterativo de (3.3) cada vez que el algoritmo haya ingresado a la red un patrón del conjunto de entrenamiento, sujetando la suma de impactos de error a un criterio preestablecido. A este modo de operación se le conoce como modo secuencial (*sequential mode*) y los pesos sinápticos serán actualizados (corregidos) una vez por cada patrón ingresado (en P ocasiones).

2. Este modo de operación del algoritmo se lleva a cabo por lotes (*batch mode*) y se evalúa el entrenamiento cada vez que se hayan ingresado a la red los P patrones contenidos en el conjunto de entrenamiento. Para cada uno de estos ciclos o *épocas*, se tiene la siguiente medida de error:
$$\xi_{Av} = \frac{1}{P}\sum_{P} \xi(l). \tag{3.4}$$
La ecuación (3.4) se considera como la función de costo de aprendizaje (o *error cuadrático medio*, MSE) para cada pasada del conjunto de entrenamiento y se considera como función objetivo de minimización.

3.2. Fundamentos de la regla delta generalizada

La regla delta generalizada es el resultado más importante en lo concerniente al algoritmo de propagación del error hacia atrás. Está basado en la suposición de que cada neurona oculta tiene un nivel de contribución en el error que comete la red; además, que esta contribución es función directa de los pesos sinápticos de toda la red. Es pertinente entonces exponer de manera rápida cómo se desarrolla la idea.

Considerando que la k-ésima neurona de la capa de salida K tiene m entradas $y_j(l)$, provenientes de la capa anterior J, se tiene que su suma ponderada se define como:

$$\nu_k(l) = \sum_{k=0}^{m} w_{k,j}(l) y_j(l) \tag{3.5}$$

Además la salida de esta neurona j está dada por:

$$y_k(l) = \varphi_k(\nu_k(l)). \tag{3.6}$$

Si sustituimos (3.6) en (3.1) y luego usamos la regla de la cadena para derivar (3.3) con respecto a $w_{k,j}(l)$ (lado derecho de (3.7)), entonces podemos definir la k-ésima componente del gradiente $\frac{\partial \xi}{\partial \mathbf{w}_k}$ de la superficie de error para la neurona k como:

$$\frac{\partial \xi(l)}{\partial w_{k,j}(l)} = \frac{\partial \xi(l)}{\partial e_k(l)} \frac{\partial e_k(l)}{\partial y_k(l)} \frac{\partial y_k(l)}{\partial \nu_k(l)} \frac{\partial \nu_k(l)}{\partial w_{k,j}(l)}. \tag{3.7}$$

Este vector gradiente $\boldsymbol{\nabla}_{\mathbf{w}_k} \xi(l)$, con base en el concepto de cálculo, representa la velocidad (módulo) y dirección (ángulo) de variación de $\xi(l)$, según los valores que se le asignen a $\mathbf{w}_k(l)$.

Como se puede observar en el lado derecho de (3.7), existen otras magnitudes que varían al variar el error, de modo que son obtenidos de manera más explícita las dependencias de cada término con el fin de poder manipular tales magnitudes:

De la ecuación (3.3) se tiene

$$\frac{\partial \xi(l)}{\partial e_k(l)} = e_k(l); \tag{3.8}$$

de la ecuación (3.1) se tiene

$$\frac{\partial e_k(l)}{\partial y_k(l)} = -1; \tag{3.9}$$

de la ecuación (3.6) se tiene

$$\frac{\partial y_k(l)}{\partial \nu_k(l)} = \varphi_k'(\nu(l)); \tag{3.10}$$

Por último, de la ecuación (3.5) se tiene

$$\frac{\partial \nu_k(l)}{\partial w_{k,j}(l)} = y_j(l). \tag{3.11}$$

Si ahora se sustituye desde (3.8) a (3.11) en (3.7) se tiene:

$$\frac{\partial \xi(l)}{\partial w_{k,j}(l)} = -e_k(l)\varphi_k'(\nu(l))y_j(l). \tag{3.12}$$

Si se sabe que la corrección de los pesos en cada iteración está dada por la *regla delta* de un perceptrón simple como[46]:

$$\Delta w_{k,j}(l) = -\eta \frac{\partial \xi(l)}{\partial w_{k,j}(l)} \tag{3.13}$$

y que usando (3.9) a (3.11) el gradiente local es:

$$\delta_k(n) = -\frac{\partial \xi(l)}{\partial \nu_k(l)} = e_k(l)\varphi'_k(\nu_k(l)), \tag{3.14}$$

de donde se tiene que sustituyendo (3.14) en (3.13) se llega a:

$$\Delta w_{k,j}(l) = \eta \delta_k(n) y_j(l). \tag{3.15}$$

Los miembros extremos de la ecuación (3.14) es el resultado que se mostró en la sección anterior para propagar el error hacia atrás, desde cada neurona hacia la capa oculta. Por otro lado, la ecuación (3.15) indica qué modificación, ponderada por η (la taza de aprendizaje), será necesario hacerle a los pesos de la capa de salida en cada iteración.

Para corregir los pesos de la capa oculta, se sigue un procedimiento similar, que mantiene la premisa de retro-propagación del error que ha sido medido en la capa de salida. Primero obsérvese que el error ya no presenta cambios, puesto que ya fue medido y sólo se está propagando; ahora, con ésto en mente, se redefine el gradiente local para la capa J:

$$\delta_j(n) = -\frac{\partial \xi(l)}{\partial y_j(l)} \frac{\partial y_j(l)}{\partial \nu_j(l)} = -\frac{\partial \xi(l)}{\partial y_j(l)} \varphi'_j(\nu_j(l)); \tag{3.16}$$

también si se deriva (3.3) con respecto a $y_j(l)$, la regla de la cadena da:

$$\frac{\partial \xi(l)}{\partial y_j(l)} = \sum_K e_k(l) \frac{\partial e_k(l)}{\partial \nu_k(l)} \frac{\partial \nu_k(l)}{\partial y_j(l)}. \tag{3.17}$$

Ahora se buscan los valores de cada término del lado derecho de (3.17). Si se sustituye (3.6) en (3.1) se obtiene:

$$\frac{\partial e_k(l)}{\partial \nu_k(l)} = -\varphi'_k(\nu_k(l)). \tag{3.18}$$

Para el siguiente término se deriva (3.5) con respecto a $y_j(l)$, considerando los índices para la capa oculta, y se obtiene

$$\frac{\partial \nu_k(l)}{\partial y_j(l)} = w_{k,j}(l). \tag{3.19}$$

Una vez hallados los términos buscados, se sustituye (3.18) y (3.19) en (3.17) de modo que se obtiene su miembro izquierdo en términos de $\delta_k(l)$ como lo indica (3.14):

$$\frac{\partial \xi(l)}{\partial y_j(l)} = -\sum_K \delta_k(l) w_{k,j}(l). \tag{3.20}$$

Y para finalizar, ya es posible expresar de manera explícita los valores que habrán de participar en el cálculo del gradiente local para la neurona j de la capa oculta, así como la corrección que habrá que hacerle a los pesos de esta capa. Se sustituye (3.20) en (3.16) y se llega a:

$$\delta_j(l) = \varphi'_j(\nu_j(l)) \sum_K \delta_k(l) w_{k,j}(l); \tag{3.21}$$

además, únicamente colocando los índices adecuados tomando como referencia la ecuación (3.15):

$$\Delta w_{j,i}(l) = \eta \delta_j(n) y_i(l).$$

Dado que $\mathbf{w}_k(l)$ y $\mathbf{w}_j(l)$ son los dominios de la superficie de error de la red, η definirá qué tan cerca o lejos estará el siguiente punto sobre el cual se evaluará ésta para determinar los parámetros de $\boldsymbol{\nabla}_{\mathbf{w}_k}\xi(l)$ y $\boldsymbol{\nabla}_{\mathbf{w}_j}\xi(l)$ respectivamente. Cabe señalar que α en la ecuación (3.2), prevendrá de errores también al entrenamiento al proponer η demasiado grande y de esta manera, se evitan oscilaciones sobre la superficie de error, involucrando en alguna medida a cambios hechos previamente. Dicho de otro modo, si se escogen valore adecuados para α y η, se reduce la posibilidad de que el entrenamiento visite zonas de la superficie sobre las que ya se ha evaluado. Es así como se llega a formular la regla delta generalizada, que es una expresión más general del análisis que se acaba de realizar.

3.3. Inconvenientes de entrenamiento y arquitectura

3.3.1. Superficie de error

El vector $\boldsymbol{e}(l)$ dado por la ecuación (3.1), describe una superficie sumamenete impredecible a medida que se lleva a cabo el entrenamiento de la red. El principal inconveniente de ésto, es que el algoritmo de retropropagación está basado en detectar los cambios de dirección y magnitud en el gradiente de dicha superficie de error, de forma tal que se obtenga un punto para el cual el valor del MSE (o del impacto de error) sea mínimo; sin embargo, el algoritmo no cuenta con un método para detectar si ha hallado un valor mínimo que no sea el óptimo; es decir, un *mínimo local*. Lo anterior plantea la controversial y antigua discusión sobre cómo puede el algoritmo determinar si al seguir con el entrenamiento, habría otro punto donde el vector de error sea de menor magnitud (aunque éste no sea el óptimo) que el que ya haya encontrado como mínimo, o cómo percibir en qué punto del entrenamiento se alcanza el valor óptimo: el *mínimo global* (véase la Figura 3.3). En general para una red neuronal de alimentación directa, hasta el momento no se tiene establecida una metodología que resuelva de manera concreta tales controversias.

El entrenamiento puede hacerse más preciso si se escoge un valor pequeño de la taza de aprendizaje, llevando a la red mucho más cerca o exactamente a un mínimo. Tal ajuste

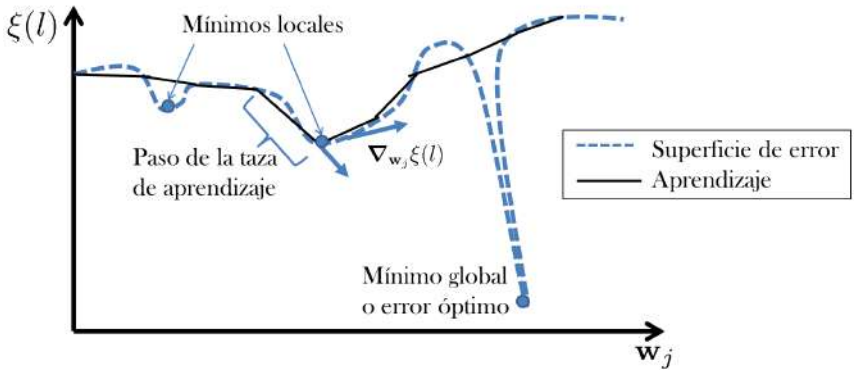


Figura 3.3: Forma hipotética de la superficie de error y el problema de los mínimos locales

asegura la convergencia; sin embargo, esto hace demasiado lento el aprendizaje dado que el algoritmo tiene que recorrer la superficie de error mediante pasos demasiado cortos (como caminando a ciegas) que aumentan el número de iteraciones necesarias para llegar un punto deseado. Ésto a su vez, demanda mucho tiempo. Por otro lado, si se intenta que la red converja más rápido aumentando el valor de la taza de aprendizaje, los pasos que da el algoritmo sobre la superficie de error son más grandes, lo cual acelera en gran medida el proceso de aprendizaje. Lo anterior también implica menor resolución al ‘‘muestrear’’ o recorrer la superficie de error, suscitándose el gran riesgo de ‘‘brincar’’ involuntariamente el punto óptimo, o de provocar que el entrenamiento oscile indeterminadamente[1] al rededor de dicho punto.

3.3.2. Sobre-ajuste (*overfitting*) y generalización (*pasticity*)

El algoritmo de entrenamiento por retro-propagación del error, es efectivo en términos generales; sin embargo, existen algunos inconvenientes que se han descubierto al evaluar el desempeño de una red MLP durante el reconocimiento de un conjunto de patrones distinto al que se ha usado como entrenamiento (incluso con el de entrenamiento).

Llevando a cabo un análisis más detallado, se encuentran problemas muy importantes que afectan negativamente el desempeño, tal es el caso del *overfitting*, el cual aparece cuando la red se ha dimensionado de manera equivocada, colocando en ella una cantidad de nodos mayor a la necesaria; es decir, cuando se tienen más de una capa oculta y que además se tengan demasiados nodos en dichas capas. Lo anterior provoca que la red memorice detalles muy poco relevantes (información innecesaria y perniciosa) de cada patrón de entrenamiento; lo cual a su vez, da como resultado una generalización pobre al tratar de reconocer patrones que no están en el conjunto de entrenamiento. Tener una capa es condición suficiente para

[1]Dichas oscilaciones son actualmente descartadas, programando al algoritmo de retro-propagación para que se detenga en caso de determinar que existen.

proveer capacidad aceptable de generalización a la red, para la gran mayoría de los problemas prácticos[55]; aunque también, se menciona y se ejemplifica en [55] que el tipo de problemas que requieren una excepción, se dan generalmente cuando se desea aproximar funciones discontinuas.

En la siguiente sección se verá un condensado de reglas importantes, las cuales proporcionan información que aproxima en la mayor medida posible, a obtener el mejor diseño de una red neuronal de alimentación directa.

3.4. Consideraciones valiosas para el diseño (Minimización de inconvenientes)

Mediante un viaje a través de los valles más prominentes de la literatura que dedica su esfuerzo a desespinar (aunque más bien a desempañar) el uso de la redes neuronales, han de observarse a continuación varias ‘‘reglas de oro’’ muy interesantes al respecto. Así mismo, se irá concluyendo cada uno de los puntos tratados en esta sección ‘‘tomando nota’’, de lo que resultó más adecuado para tomar en cuenta en el diseño de las redes neuronales que se implementaron en este trabajo de tesis.

Según se propone en [56], es posible tomar en cuenta en el diseño de una red neuronal una sucesión de pasos basados en la vasta experiencia de investigadores especializados en el tema, desde luego que tales reglas son subyacentes a la naturaleza de la red, al conocimiento, en gran medida empírico, sobre su comportamiento y entrenamiento; así mismo, deben su origen al continuo interés por desvanecer gradualmente la repercusión de los inconvenientes abordados en la sección anterior.

Obsérvense ahora cada uno de los pasos sugeridos de diseño. Puesto que el interés de este trabajo de tesis es la supervisión de un sistema electrónico, algunos de los pasos se han orientado de manera acorde.

3.4.1. Paso 1: Selección de variables

También conocido como *selección de sensores*, consiste en entender, antes que cualquier cosa, de manera clara la naturaleza del problema a resolver, de modo que se sepa qué señales de voltaje son relevantes para ser propuestas como descriptores que sean elementos de un patrón de entrada a la red. Desde luego que habrá señales que no sean necesariamente un voltaje, pero ya se verá que hacer en el paso 3.

3.4.2. Paso 2: Recolección de datos

Esta fase consiste en obtener los datos que componen cada señal del conjunto de patrones que se usarán para entrenar y probar la red. Existen dos opciones para llevar a cabo la recolección: 1) Consultar la correspondiente información que el fabricante o implementador poseen acerca del sistema electrónico o proceso que se va a supervisar. Esto facilita mucho el trabajo y ahorra tiempo muy valioso. 2) Otra opción mucho más complicada (pero tal vez mucho más fiable) es hacer mediciones directamente en el sistema o proceso, de modo que se adquieran y ordenen los datos a conveniencia propia.

3.4.3. Paso 3: Preprocesamiento de datos

Una red neuronal es sencible a los cambios en sus entradas en dos sentidos: 1) se dificulta el procesamiento y el aprendizaje en la red cuando los valores de entrada son de ordenes de magnitud muy separados y 2) las señales periódicas o con fluctuaciones que por sí mismas no aportan información pueden inducir un efecto contraproducente, haciendo que la red detecte cambios innecesarios. En estos casos será necesario llevar a cabo primero un pre-proceso de extracción de características (tales como frecuencia, amplitud, fase o corriente), mismas que se convertirán en un voltaje que pueda ser medido (ésto también se conoce como *abstracción de señales*). Después será necesario un pre-proceso de normalización: $N_s : \mathbb{R} \to [0, 1]$ ($N_t : \mathbb{R} \to [-1, 1]$ para el caso de la función de activación tangencial), el cual resuelve el problema de la desproporcionalidad en los órdenes de magnitud de las señales de entrada.

3.4.4. Paso 4: Construcción de los conjuntos de entrenamiento, prueba y validación

Cuando ya se tienen recolectados y pre-procesados los datos, entonces se tiene información necesaria para construir conjuntos de patrones que sirven a final de cuentas para validar que la red se comporta tal como es deseado[56]. El primer conjunto que se obtiene directamente de los pasos anteriores, es el conjunto de entrenamiento, éste es el más grande y general. Será usado para entrenar a la red, emparejado con un conjunto de salidas deseadas. El tamaño de este conjunto es sugerido que sea del 85 % del total de muestras recolectadas. El siguiente conjunto de datos que hay que tener en mente es el conjunto de prueba, el cual se sugiere que deba ser de 10 % a 30 % del tamaño del conjunto de entrenamiento, además puede ser parte de él. Éste será usado para hacer pruebas que indiquen qué tan bien entrenada ha quedado la red. Por último, el conjunto de validación debe ser construido con patrones que hayan sido adquiridos de manera separada a los conjuntos de entrenamiento y

prueba; o mejor aún, en recientes adquisiciones. También es posible construir un conjunto de validación a partir del de entrenamiento y pruebas, seleccionando patrones en orden distinto (si es posible aleatoriamente) al que se presentan en el conjunto al que pertenecen originalmente. Este último conjunto es de vital importancia, porque permite al diseñador saber qué capacidad de generalización ha adquirido la red; después de dimensionarla, entrenarla y probarla. Es preferible tener la precaución de no emplear el conjunto de validación, o partes de él, como conjunto de entrenamiento, puesto que esto afecta la evaluación de la capacidad de generalización de la red[55]. El tamaño sugerido para este último conjunto es del 25 % del total de datos adquiridos.

3.4.5. Paso 5: Dimensionamiento de la red

Como se verá en el Capítulo 5, esta etapa es de las más importante de todas, ya que el desempeño del aprendizaje (convergencia) y reconocimiento (capacidad de generalización) dependen mucho de la arquitectura de la red. El seleccionar un dimensionamiento adecuado, puede acercarla mucho a las características de un clasificador óptimo, descartando al máximo los inconvenientes vistos en la sección anterior, que por mucho tiempo han puesto a las redes neuronales como blanco de fuertes críticas o incluso descalificaciones.

Número de capas ocultas

Existe en la literatura una gran variedad de puntos de vista; sin embargo, tras la revisión de la misma, se han tomado en cuenta las opiniones más citadas, que por lo tanto, se convierten en las más representativas desde el punto de vista de los resultados experimentales. La mayoría de las pruebas de validación, se han hecho en base a aproximación de funciones que son de naturaleza referente.

En [57] se menciona precisamente lo más importante: [55] establece que en una inmensa mayoría de casos, una capa oculta es suficiente para lograr suficiente capacidad de generalización; así mismo, refiere que si una red neuronal contiene más de una capa oculta, la velocidad de aprendizaje se hallará seriamente comprometida, dado que las variaciones en la curva de error vuelven inestable al gradiente, ésto debido a que aumenta considerablemente el número de mínimos locales. También argumenta que una segunda capa oculta le quita capacidad de generalización a la red. Sin embargo, existen un grupo (no menos frecuente en el mundo real) de funciones con la singular característica de tener frecuentes discontinuidades ([55] muestra el ejemplo de una señal diente de sierra), con las cuales una red de dos capas ocultas muestra un desempeño ligeramente (pero considerablemente valorable) mejorado en comparación con una red que sólo tiene una capa oculta. Incluso esto último, puede llegar a ser una tarea imposible para una red mono-capa-oculta[62].

Por otro lado, con menos partidarios, Daniel L. Chester[63], de la Universidad de Delaware, demuestra que no necesariamente el hecho de que la red contenga más de una capa oculta implica que habrá sobre-ajuste y baja capacidad de generalización; más bien al contrario, *''dos capas ocultas son mejores que una''*, debido a que la primera se encarga de extraer características locales de cada patrón de entrenamiento, mientras la segunda lleva a cabo la tarea de extraer características globales del conjunto de entrenamiento. Lo anterior es válido si se lleva a cabo un dimensionamiento adecuadamente incremental de las capas ocultas, obteniéndose incluso una red con menos neuronas que las que tendría una red con una capa oculta[61], lo cual implica menos complejidad computacional tanto del entrenamiento como en fase de reconocimiento de patrones.

Se ha visto que las opiniones anteriores parecieran excluyentes; sin embargo, más bien son complementarias, ya que se pueden tomar en cuenta ambas para realizar un buen diseño de red de alimentación directa. La afirmación anterior, se basa en que durante la revisión de la literatura citada se proponen notebles diferencias en los casos en los que se aplica cada enfoque, básicamente dos capas son mejores que una si se tiene que aproximar funciones discontinuas[63, 55] o si se quisiera resolver un problema que resulte costoso o imposible para una red mono-capa-oculta[61]; así mismo, una capa oculta es suficiente en cualquier otro caso[55, 46].

Número de neuronas ocultas

Otro aspecto no menos importante sobre el dimensionamiento de una red neuronal es el número de neuronas (nodos) que deberá contener cada capa oculta. Para este respecto, existen opiniones todavía más variadas. También, para este caso sólo nos referiremos a los enfoques más destacados, empezando por los menos formales y terminando con los más probados.

Todos los enfoques que se puedan documentar sobre cómo obtener el tamaño de una capa oculta, son meramente empíricos. Se basan en la experiencia y habilidad del desarrollador (metodologías Heurísticas), también en el problema que se está resolviendo, el cual tiene que ver directamente con el número de entradas, el número de salidas, el tamaño del conjunto de entrenamiento e incluso el número de parámetros libres[2] que tendrá la red. De modo que es de suma importancia que el paso 1 se haya realizado de manera correcta.

Así pues, el enfoque clásico (*pruning*) considera que no son iguales todos los problemas. [55] sugiere empezar por elegir un criterio adecuado de verificación de rendimiento; construir

[2]Se puede probar que una red en particular cuenta con una propiedad llamada *VC-dimension* (Vapnik-Chervonenkis dimension), la cual establece la cantidad h de patrones que es capaz de aprender una red neuronal de alimentación directa. Es posible también mostrar que h esta acotada superiormente por $O(W^4)$ (W es el número de parámetros libres de la red), si los nodos de procesamiento de la red cuentan con funciones de activación *sigmoide*[46].

la red con una cantidad poco abundante de nodos (tal vez dos) en una capa oculta. Entrenar la red y verificar su desempeño; agregar neuronas de manera paulatina en cada pasada, teniendo cuidado de almacenar entrenamientos previos. Detener el procedimiento cuando deje de haber mejoras en el criterio de verificación, elegir el tamaño óptimo de la red y establecer en ella también el entrenamiento que haya resultado óptimo. Como se puede ver, es un procedimiento que puede llevar mucho tiempo (tal vez como sucedería en un sistema biológico) pero es muy eficaz; garantiza la obtención de una arquitectura muy cercana a la óptima.

Algunos otras sugerencias que establecen tamaños fijos están descritas en [57]. La primera sugiere que el número de neuronas ocultas n_h ha de ser el 75 % del número de neuronas de entrada n_I. El siguiente sugiere que n_h sea el promedio de la cantidad de neuronas de entrada y las de salida n_O; es decir: $n_h = \frac{n_I+n_O}{2}$. Otra sugerencia es colocar $2n_I + 1$ neuronas en la capa oculta; tal como se prueba para algunos casos cuyos resultados son satisfactorios[58]. Por último y de manera más formal, Xiao-Hu Yu establece el siguiente teorema[54]:

Teorema 3 *Sea $\boldsymbol{w} \in \mathbb{R}^n$ el vector de pesos sinápticos de una red y sea n_I el número de entradas de la red; la superficie de error $H(\boldsymbol{w})$ no tiene mínimos locales si a la red de alimentación directa se le aplica exactamente un conjunto de entrenamiento arbitrario con n_I vectores de entrada distintos y dicha red tiene una capa oculta de tamaño $n_I - 1$.*

El Teorema 3 basa también su existencia en una metodología inductiva, sin ponerlo de manifiesto explícitamente, afirmando que si la red cumple con estas condiciones, la respectiva superficie de error no tendrá mínimos locales y que el único mínimo hallado será el óptimo. Xiao-Hu afirma en su trabajo que después de entrenar la red un razonable número de veces, con el conjunto de entrenamiento y la red dimensionados de acuerdo con el teorema y si además después de cada ciclo de entrenamiento se tiene el mismo mínimo, esto significa que se trata del mínimo global.

Existe otra propuesta destacada que ha sido empleada en varios trabajos[55], incluyendo el presente trabajo, dicha propuesta es conocida con el nombre de *regla de la pirámide geométrica*. Ésta establece que el número de neuronas en la capa oculta deberá ser:

$$n_h = \sqrt{n_I n_O} \tag{3.22}$$

La ecuación (3.22) es válida para redes de una capa oculta. Se tiene una generalización para redes de 2 capas ocultas:

$$r = \sqrt[3]{\frac{n_I}{n_O}}$$

$$n_{h_1} = n_O r^2$$

$$n_{h_2} = n_O r$$

Las propuestas anteriores no funcionan en casos particulares y se consideran buenas aproximaciones del tamaño óptimo, aunque no se tenga una prueba formal sobre (3.22), ni se especifiquen características particulares de los problemas en los cuales serían funcionales; por ejemplo: aproximar una función no lineal de una variable (que implica tener sólo una entrada y una salida en la red). Para el caso mencionado, se tendría una red con una neurona de entrada y una de salida, pero no sería posible colocar una capa oculta cuyo número de unidades sea un porcentaje del número de entradas, ni sería suficiente con el doble o un número menor al número de entradas. Tampoco se podría aplicar la regla de la pirámide geométrica. De modo que se tendría que aplicar heurística nuevamente.

Cabe destacar que se tiene como buena práctica (para redes más usuales, diferentes a la descrita anteriormente) la siguiente[55]: empezar dimensionando la red usando la regla de la pirámide geométrica e ir agregando neuronas en cada pasada hasta alcanzar el criterio de verificación óptimo.

Existen, en otro nivel mucho más formal y apegado al empirismo básico, los *algoritmos heurísticos (self-pruning)* que llevan a cabo estas tareas de manera automática; tal como sugiere [59]: *The Cascade-Correlation Method*. En esta propuesta se plantea la generación autónoma (en contraposición, existen métodos degenerativos) de neuronas ocultas, partiendo desde la ausencia total de las mismas, agregando una (o unas pocas también es posible) a medida que el entrenamiento avanza. Con el brote de una nueva neurona, el entrenamiento previo es congelado de modo que se entrene sólo el nuevo nodo. El resultado de cada pasada es evaluado mediante una prueba de capacidad de generalidad (suma de errores cuadráticos por pasada [60]), de tal forma que el proceso es detenido cuando deja de haber mejoras en el desempeño de la red, según la prueba mencionada.

Por último, en [55] se destaca que los métodos de dimensionamiento automático (self-pruning) pueden hacer caer al diseñador en abusos que lleven a construir redes poco óptimas o que le quiten experiencia muy valiosa para el entrenamiento en casos especiales que la requieran. Sugiere también de manera notable que la meta del diseño debería ser siempre tener el menor número de neuronas ocultas posible y el mayor rendimiento posible; adicionalmente, que el ‘‘pruning’’ es el mejor procedimiento a seguir para alcanzar tal meta.

En lo que se refiere a este trabajo de tesis, se opta por probar el ‘‘pruning’’ como heurística incremental, iniciando el entrenamiento con la pirámide geométrica y agregando neuronas hasta que el criterio de verificación se cumpla. Por otro lado, se verificará lo estipulado en el Teorema 3.

Considerando tal decisión, se sabe que cada cajón del piloto automático tiene una cantidad diferente de salidas que se han de monitorizar; por lo tanto, ya vista la temática sobre el dimensionamiento y cómo éste está relacionado con el entrenamiento y el desempeño de la red, es de suponerse, sin temor a errar, que será absolutamente necesario diseñar una red neuronal para cada cajón. Véanse más adelante, en el Capítulo 5, los resultados de la

aplicación e importancia de las reglas vistas hasta el momento.

Número de neuronas de salida

Para tomar esta decisión no existe alguna regla que diera idea de cómo deberá dimensionarse la capa de salida; sin embargo, vamos a tomar en cuenta de manera fundamental lo establecido en [55, 62]; además, justificaremos el tamaño de la capa de salida hablando en términos de aproximación de funciones no lineales. Se define el tamaño de la capa de salida de una red neuronal como función directa de lo requerido por la aplicación; es decir, si se requiere predecir o estimar un espacio de salida multi-variable, entonces es necesario usar tantas neuronas de salida como variables requiera el problema. En [64] por ejemplo, se usan salidas múltiples para presentar posibles estados futuros de la salida principal. Adicionalmente, si el problema únicamente requiere que la red entregue un valor a la vez, estamos en la condición en que se requiere sólo una neurona de salida.

Es importante también otro aspecto que trata sobre la función de activación usada en la capa de salida que define a una red neuronal como un *Aproximador Universal*. En [55] se argumenta de manera muy clara el uso de neuronas de salida cuya función de activación deba ser lineal no limitada, exponiendo que de esta forma la red neuronal es capaz de aproximar casi cualquier función, cuyo dominio sea casi cualquiera. El uso del adverbio que denota duda (‘‘casi’’), se debe a que existe una condición cuya omisión podría llegar a incapacitar a la red para hacer una aproximación de manera satisfactoria (podría volverse inestable o impredecible su comportamiento); tal condición está formalmente definida en el siguiente *Teorema de aproximación universal*[65, 61] (También conocido como *Teorema de Hornik-Cybenko-Funahashi*):

Teorema 4 *Sea f_{NN} una red neuronal con una, dos o tres capas ocultas; para cualquier función continua no constante $f(\boldsymbol{x}_1, ..., \boldsymbol{x}_n)$ sobre un conjunto compacto $M \subset \mathbb{R}^n$ y $\forall \epsilon > 0$, existen vectores de pesos $\boldsymbol{w}_1, \boldsymbol{w}_2$ ó $\boldsymbol{w}_3$; $\forall \boldsymbol{x}_1, ..., \boldsymbol{x}_n \in M$, tales que*

$$|f(\boldsymbol{x}_1, ..., \boldsymbol{x}_n) - f_{NN}(\boldsymbol{x}_1, ..., \boldsymbol{x}_n)| < \epsilon,$$

donde los $\boldsymbol{x}_i$ son los patrones de entrada.

El resultado anterior es contribución original de Kur´ Hornik[66], aunque él propone una versión menos flexible en cuanto al acotamiento tanto del dominio de las funciones que se podrían aproximar (se especifica en términos de probabilidad) como del tipo de funciones de la capa de salida de la red (funciones de amplitud limitada, *squashing functions*, como la sigmoidal o la tangencial).

Nuestra aplicación en el diagnóstico de un cajón del P. A. del Metro, requiere que la red presente como salida únicamente un conjunto finito de índices (cuya etiqueta es el estado

temporal de las pruebas que han de realizarse), al cual podemos ver como una función lineal $f : D \to Y$ tal que $D \subset \mathbb{R}$ y $Y \subset \mathbb{R}$ son acotados. Dicha función habría de aproximarse mediante la red $f_{NN} : X \to Y_{NN}$ tal que $X \subset \mathbb{R}^n$, $Y_{NN} \subset \mathbb{R}$ y $Y_{NN} \approx Y$; siendo X e Y_{NN} conjuntos acotados también. Es suficiente entonces el uso de una capa de salida monolítica que se comporte como un combinador lineal[62]. Ésto en virtud de que la neurona que la constituye estaría provista de una función de activación lineal no-limitada.

3.4.6. Paso 6: Establecer un criterio de evaluación

Para seleccionar un criterio de evaluación, es necesario mirar un poco más hacia la aplicación. Ésto se debe a que cuantitativamente nosotros podemos tomar en cuenta medidas que nos indican cómo funciona la red; cometiendo el posible desatino de no verificar si en verdad nos entrega la información deseada. Cuando medimos porcentajes de error o el error cuadrático medio, estamos verificando el desempeño de la red de manera cuantitativa. Puede que para casos particulares no sea suficiente con eso.

En algunos casos, todas las salidas de la red forman conjuntos en el espacio de salida, además estos conjuntos tienen que ser interpretados para tomar una decisión; para tales casos, es necesario tomar en cuenta la cantidad de decisiones acertadas tomadas gracias al uso del espacio de salida de la red. Lo último no está directamente relacionado con la medida de error que se obtiene directamente en la capa de salida de la red cuando se compara con el espacio deseado, lo cual puede conducir a decisiones erróneas. El algoritmo de aprendizaje ya verifica los errores de la red, pero ahora el diseñador debe verificar que su diseño realmente ayude a que la aplicación esté interpretando bien la salida de la red entrenada.

> ''*Una red neuronal MLP de alimentación directa, PUEDE aprender cualquier función. Si se tienen problemas, éstos no son debidos al modelo en sí. Más bien se deben a un entrenamiento insuficiente sobre un número insuficiente (o no adecuado) de neuronas ocultas, o bien a que se intenta entrenar una funcion que se supone no determinística.*''[55]

3.4.7. Paso 7: Entrenar la red

Se ha hablado en la sección 3.1 sobre el entrenamiento de la red, sólo que ha sido en términos propiamente del algoritmo que se encarga de hacer tal tarea. Se hablará ahora a hablar no de lo conceptual del entrenamiento, sino de las metodologías que un investigador puede usar como guías para entrenar una red neuronal y decidir si ésta está entrenada o no. Pero como se dijo en el paso anterior, habrá que considerar el entorno en el que la red se desenvolverá para elegir un criterio.

La parte general del proceso es que habrá que presentarle a la red un conjunto de entrenamiento $\{\boldsymbol{x}_1, ..., \boldsymbol{x}_p\}$, dicho conjunto irá acompañado de su respectivo conjunto de ejemplos $\{\boldsymbol{t}_1, ..., \boldsymbol{t}_p\}$. A este acompañamiento le llamaremos conjunto par de entrenamiento $\{\{\boldsymbol{x}_1, \boldsymbol{t}_1\}, ..., \{\boldsymbol{x}_p, \boldsymbol{t}_p\}\}$. El conjunto de ejemplos le proporcionará al algoritmo la referencia requerida para calcular el error que comete la red y así corregir los pesos.

También se sabe que el proceso se lleva a cabo de manera iterativa, ya sea en ''*sequential mode*'' o en ''*batch mode*''. Se habrá de particularizar sobre las heurísticas principales que establecen criterios de desempeño de la red de forma tal que se defina un punto de parada para el entrenamiento [46, 56].

1. **Mejoramiento del error (*Error enhancement*)**. Este criterio establece que el entrenamiento no se detendrá hasta que la función de error deje de mostrar mejoras después de haber entrenado a la red e inicializado los pesos aleatoriamente un número razonable de veces. Se escoge la configuración que mostró mejores resultados. Para el ''*sequential mode*'' será considerada la función ξ y para ''*batch mode*'' sería ξ_{Av}.

2. **Descenso del gradiente (*Gradient descent*)**. Es éste el criterio más básico y en el cual se considera a detalle la naturaleza del algoritmo de retro-propagación. Consiste en verificar, ya sea en cada ''*batch*'' o por cada patrón, la magnitud de la norma euclidiana así como la dirección de $\boldsymbol{\nabla}_{\mathbf{w}_k}\xi(l)$ (el sentido está implícitamente negativo en el algoritmo). Una razón suficiente para detener el algoritmo, será que la norma $||\boldsymbol{\nabla}_{\mathbf{w}_k}\xi(l)|| = 0$; además de que su dirección deberá ser descendente, de modo que se garantice que se ha localizado un mínimo. La forma de hacer necesaria la detención del algoritmo es combinando la razón suficiente que hemos mencionado con lo establecido en el criterio anterior (Mejoramiento del error).

3. **Entrenar-probar (*Train-test*)**. Este criterio es posiblemente el más rápido de todos, dado que el conjunto de actividades que han de llevase a cabo está muy acotado; ésto es debido a que antes de empezar el entrenamiento, tendrán que fijarse un número de iteraciones y una medida mínima del criterio de evaluación de la capacidad de generalización de la red. Estas dos cantidades (demasiado generales tal vez) serán usadas como puntos de parada del algoritmo; con lo cual, ya sabremos cuándo se detendrá el algoritmo antes de empezar el entrenamiento. En cada iteración, han de inicializarse los pesos de la red aleatoriamente, de modo que se recorre la superficie de error en diferentes zonas aleatorias donde pudiera hallarse el mínimo global. Una vez alcanzado el punto de parada del entrenamiento, se elige la configuración con la mejor capacidad de generalización (donde muy probablemente se hallaría localizado el error óptimo).

Como sea ha observado, las tres son opciones totalmente válidas; sin embargo, se ha elegido como mejor opción la número 3 para hacer pruebas con el diseño basado en la pirámide

geométrica, en virtud de que por más analíticas que sean las otras dos, no garantizan de ninguna manera hallar el mínimo global; aún cuando se pueda invertir demasiado tiempo empleándolas. Por otro lado, esta opción pudo proporcionar una prspectiva de mayor versatilidad, dado que se tenía la misión de hacer pruebas relativamente numerosas.

3.4.8. Paso 8: Implementación

Llegado el momento de llevar a cabo este último paso, lo más complicado ya está hecho; sin embargo, el investigador deberá tener mucho cuidado en haber registrado de manera adecuada los parámetros obtenidos en los pasos anteriores, de modo que al elegir la mejor configuración, ésta pueda ser reproducida de manera exacta, conservando las características que la hicieron destacar; por ejemplo, en este trabajo de tesis, se obtuvieron redes neuronales con diferentes características y parámetros para cada cajón del PA, mismas que incluso dependen del número de patrones que cada red aprendió. Por lo tanto, en caso de haber elegido como mejor algoritmo a las RNAs, es de suponerse que pudieron haberse presentado contratiempos con las redes obtenidas en MATLAB, al momento de migrarlas a LabWindows/CVI [3]; tal vez inconsitencias en los parámetros de la red, en la implementación de cada nodo de procesamiento, o simples cambios en el formato de los tipos de datos que usan una y otra plataforma.

[3]Puesto que se ha elegido esta última como plataforma de implementacion del producto final (véase el Capítuo 6).

Capítulo 4

Clustering difuso LAMDA

LAMDA está basado en el concepto de *grado de adecuación*, cuyo valor indica una medida cuantitativa sobre cuánto se adapta un sujeto de entrada a cada miembro de un conjunto de clases definidas en el espacio de características. Es una función discriminante que reemplaza al enfoque usado en metodologías clásicas de clustering, basadas en el cálculo de funciones de proximidad (usualmente la distancia euclidiana o la distancia de Manhattan).

En la Sección 2.6 se expuso la manera detalla en que opera un algoritmo de clustering basado en optimización, posteriormente se mostraron los fundamentos de lógica difusa en la Sección 2.7.1 y después se analizó la estructuración de un algoritmo de clustering difuso en la Sección 2.7.2. La secuencia anterior, fundamenta en lo general y en lo particular lo que se trata específicamente en este capítulo.

Antes de entrar de lleno en materia, veáse primero con propósitos introductorios la estructuración del presente capítulo:

- *Paticionamiento difuso.* Se trata primero lo referente al particionamiento del espacio de entrada, ya que la explicación sobre cómo funciona el algoritmo, asume que tal partición ya existe o que será creada, y que cada objeto de entrada es asignado a alguna de las regiones (clases) que ya se tienen identificadas; por lo tanto, es preciso concer la manera en que tales regiones del espacio de entrada son parametrizadas por el algoritmo, de manera que en adelante sólo se hace referencia a tales parámetros.

- *Ausencia de información.* Como segunda sección del capítulo, se trata el caso en que un objeto de entrada no aporte suficiente información al algoritmo, para que éste decida a qué clase asignarlo. Lo anterior, representa el caso base en la secuencia de procesamiento del algoritmo y permite la creación automática de clases nuevas.

- *Apredizaje.* Se coloca como cuarto punto a tratar sobre LAMDA y complementa sutilmente lo tratado en el punto anterior: Creación automática de clases y evolución de las que ya existen. Adicionalmente, plantea una idea global sobre cómo se utilizan los conceptos tratados en la siguiente sección, así como de su importancia.

- *Grados de adecuación.* Dado que en las secciones anteriores del capítulo se expusieron diferentes opciones para configurar a LAMDA, ahora se formaliza una configuración en particular en términos propios del algoritmo y se plantean los criterios de elección que se consideraron para elegir dicha configuración como adecuada para experimentación.

- *Esquema general del algoritmo de clustering difuso LAMDA.* En esta sección se exponen las fases del algoritmo, puesto que ya se conocen sus elementos y aspectos particulares, sólo es necesario un esquema general.

- *Propiedades de LAMDA.* En base a lo expuesto en las secciones del capítulo, se expone únicamente un resúmen de las propiedades del algoritmo, mismas que justifican en gran manera su elección como propuesta para este trabajo de tesis.

4.1. Particionamiento difuso

El proceso mediante el cual se divide el espacio de entrada al diseñar un algoritmo de clustering difuso se llama particionamiento difuso. Consiste en asignar un conjugado de términos, que pertenecen a un conjunto de palabras intercambiables (*paradigma lingüístico*), que sirva para etiquetar a cada clase que existirá en el espacio de características que será referenciado por el algoritmo de clustering difuso; por ejemplo: si se tiene un vector que represente la descripción física de una persona, es posible definir un paradigma lingüístico para tres clases a las cuales posiblemente podrá pertenecer el descriptor `estatura`; dichas clases serían `baja, media` y `alta`. Se ha visto en la Sección 2.7.1 las propiedades principales de los conjuntos difusos y se puede ver que si no se tuviesen las facilidades que éstos brindan, sería muy complicado establecer un criterio de clasificación para el descriptor `estatura` en cualquiera de las tres clases que se han definido, puesto que los valores que podría tomar éste, son tan variados que no es posible representar la imprecisión del conjunto {`baja, media, alta`}.

Con el fin de determinar la forma en que se haría consideración de tales imprecisiones, ha de realizarse el particionamiento del espacio de entrada tomando en cuenta la naturaleza imprecisa de los datos de entrada. Como ya se ha visto en la Sección 2.7.2, la mayoría de los algoritmos usan una métrica ponderada por la matriz de particionamiento para determinar si tal o cual objeto pertenece o no a una clase (la matriz de particionamiento ya indica que tanto pertenece el objeto de entrada a cada clase). Dicha matriz de particionamiento representa una *partición difusa simple*, definida en la Sección 2.7. La estructuración de LAMDA es un

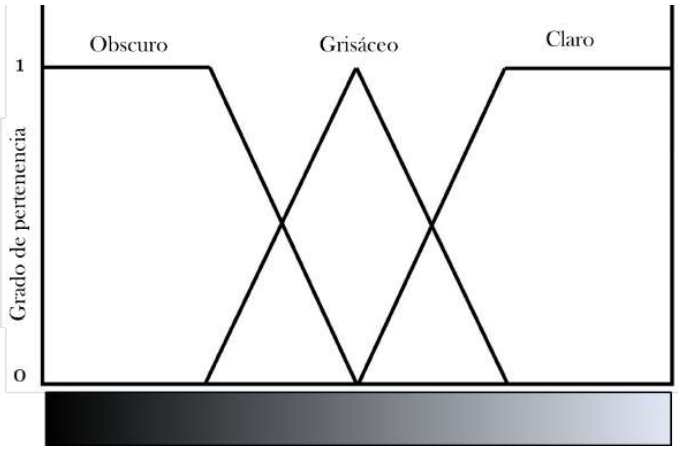


Figura 4.1: Ilustración del paradigma parmenídeo {`obscuro, grisáceo, claro`} etiquetando una partición difusa de colores en escala de grises.

tanto diferente, pues se prescinde de dicha ponderación haciendo uso directo de los valores de la función de pertenencia (valores *marginales*) de cada conjunto difuso de objetos.

Para describir el particionamiento que se usa en LAMDA, es necesario primero especificar que el paradigma lingüístico que habrá de etiquetar a los elementos del espacio de entrada, deberá hallarse dispuesto en cierto orden (así mismo tales clases, véase la Figura 4.1); formalmente[73]:

Definición **9** *Cuando el paradigma lingüístico que etiqueta a los conjuntos de un espacio de entrada se halla ordenado secuencialmente, de manera que los términos que se encuentran en los extremos de la secuencia ordenada son antogónicos, se dice que se tiene un paradigma parmenídeo.*

Antes de continuar, es necesario ahora aclarar que se definirá un único tipo de función de pertenencia, digamos f_C, para todos los conjuntos del espacio de entrada, de modo que se podrá distinguir que cada valor para el que dicha función está definida, será denominado de otra manera si se le trata de manera individual: **grado de adecuación** de cada descriptor del objeto de entrada a cada elemento correspondiente del representativo de una clase.

Sea $X = \{\mathbf{x}^{(i)} |\ i = 1, 2, ..., n\}$ el espacio de entrada m-dimensional, sea también la matriz de centros paramétricos $\boldsymbol{\rho} = \{\boldsymbol{\rho}_k |\ k = 1, 2, ..., l\}$, representativa de los elementos de la partición $C = \{C_k |\ k = 1, 2, ..., l\}$ etiquetadas mediante un paradigma parmenídeo; se define entonces uno de los conceptos más importantes en LAMDA, llamado vector de *adecuación* del objeto $\mathbf{x}^{(i)}$ a la clase C_k, denotado por $\boldsymbol{\mu}_k(\mathbf{x}^{(i)})$ y definido como:

$$\{\boldsymbol{\mu}_k(\mathbf{x}^{(i)}) = (\mu_1(x_1), .., \mu_j(x_j), ..., \mu_m(x_m)) \in [0, 1]^m |\ k = 1, 2, ..., l\},$$

donde cada $\mu_j(x_j)$ es el *grado de adecuación* de x_j a ρ_j, el cual más adelante será llamado *Grado de Adecuación Marginal (MAD, Marginal Adecuation Degree)*. Una vez fundamentados los elementos anteriores, podemos definir a una partición difusa simple en términos del vector de adecuación[73]:

DEFINICIÓN **10** *Una **partición difusa simple** es una colección de subconjuntos difusos de un universo de discurso C cuyas funciones de pertenencia tienen la propiedad[73]:*

$$\{\boldsymbol{\mu}_k(\boldsymbol{x}^{(i)}) = f_{C_k}(\boldsymbol{x}^{(i)})|\ \exists j :\ \ \mu_j(x_j) > 0\},$$

donde $f_{C_k}(\bullet)$ es una función de pertenencia elegida para todos los conjuntos de la partición C pero que es calculada para la clase C_k, cada $\mu_j(x_j)$ es el grado de adecuación de la j-ésima componente de $\boldsymbol{x}^{(i)}$ a la j-ésima componente de $\boldsymbol{\rho}_k$.

Usualmente la función de pertenencia es una función de densidad de probabilidad cuya naturaleza veremos más adelante. Además se incluye f_{C_k} en la Definición 10 para respetar la notación del creador la lógica difusa[72]; sin embargo, en adelante usaremos indistintamente los términos *función de pertenencia* y *vector de adecuación*, así mismo iremos desvaneciendo la notación $f_{C_k}(\bullet)$ para usar definitivamente $\boldsymbol{\mu}_k(\bullet)$, dado que la segunda es la usual en la literatura de LAMDA.

En una partición difusa (no sólo en la simple), se tienen varios conceptos que describen a cada uno de sus elementos[74], algunos de ellos no son de uso común cuando se habla en términos del algoritmo; sin embargo, es necesario definirlos para que nos sirvan como fundamento para los conceptos más generales de LAMDA, los cuales si utilizaremos más adelante de manera muy familiar.

Tenemos primero al conjunto de objetos para los cuales la función de pertenencia no es nula para la clase (conjunto difuso) C_k en la partición difusa C. A tal conjunto se le llama *soporte* de C_k y se define así:

$$\text{supp } C_k = \{\mathbf{x}^{(i)} \in X|\ \ ||\boldsymbol{\mu}_k(\mathbf{x}^{(i)})|| > 0\}$$

Cuando $\boldsymbol{\mu}_k(\mathbf{x}) = \mathbf{1}$, se dice que $\mathbf{x}$ es el *prototipo* de la clase C_k y que además tal clase está *normalizada*. Puede haber varios prototipos para esta clase, formalmente si esto sucede en un rango de índices consecutivos $\{\delta\}$, se tiene un conjunto de objetos para la k-ésima clase, el cual se dice que es el *núcleo* de dicha clase[1] normalizada y se define a continuación:

$$\nu_k = \{\mathbf{x}^{(\delta)}\};\ \ \{\delta\} \subset \{\eta\},$$

donde $\{\eta\}$ es el conjunto consecutivo de índices $\{i\}$ para el cual existe supp C_k.

[1]Véase que si $\{\delta\} = \{\eta\}$, entonces el conjunto (clase) C_k deja de ser difuso.

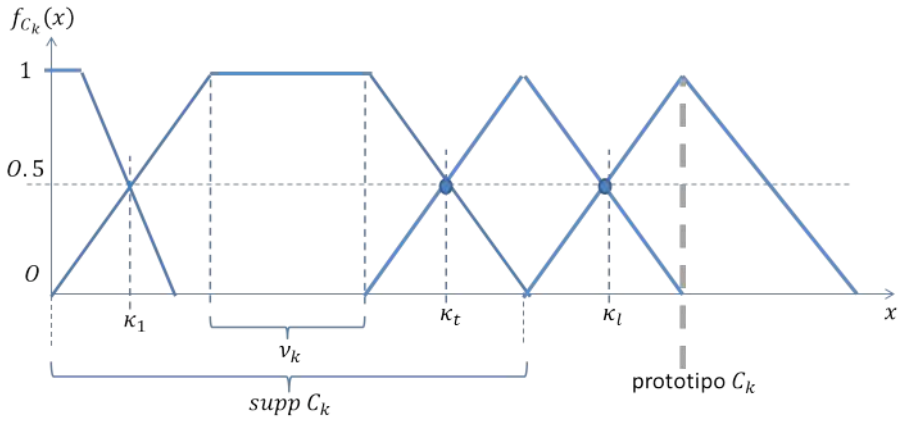


Figura 4.2: Ilustración de los elementos principales de una partición C sobre X.

Nótese en la Definición 10 que debido a su única restricción ($\exists j: \ \mu_j(x_j) > 0$) se ha de permitir bajo este esquema que los subconjuntos de C se hallen dispuestos de tal manera que un traslapo entre ellos no sea necesariamente descrito en dicha definición, pudiendo no existir y en caso de darse, no estaría especificado o controlado bajo ninguna restricción.

Por último, se definirá a los *puntos de corte* $\{\kappa_t \in [0,1]\}_{t\in\mathbb{N}^+} \subset X$, los cuales cumplen con las siguiente condición:

$$\mu_k(\kappa_t) = \mu_{k+1}(\kappa_t),$$

donde se puede observar que κ_t existe sólo para cada par de funciones de pertenencia contiguas. Obsérvense también algunas propiedades de los puntos de corte: si se tienen un par de conjuntos difusos contiguos C_r y C_s tales que $r \neq s, \ \ r \in \{1,2,..,l\}, \ s \in \{1,2,...,l\}$; contenidos en la partición difusa C (l-particionada) y dichos conjuntos (clases) se hallan traslapados, entonces el grado de separabilidad entre ellos cumple con $0 \leq D < 1$; y por lo tanto: $M_{C_r \cap C_s} = \mu_{C_r \cap C_s}(\kappa_t) > 0$. Si además este caso se cumple para todos los conjuntos de la partición ($r = 1,2,..,l; \ \ s = 1,2,...,l$), entonces el conjunto de puntos de corte en ella cumple con $\{\mu_{C_r \cap C_s}(\kappa_t) > 0 \mid t = 1,2,...,l-1\}$. Por otro lado, si para algún par C_r, C_s y algún $t \in \{1,2,...,l-1\}$ se tiene que $D = 1$, entonces $\nexists M_{C_r \cap C_s}$ y $\mu_{C_r \cap C_s}(\kappa_t) = 0$, puesto que $C_r \cap C_s = \emptyset$. Refiérase a la Figura 4.2 para crearse un constructo propio los conceptos mencionados.

4.1.1. T-normas y T-conormas (o S-normas)

Se tiene otro tipo de particionamiento difuso más restringido, pero con mayores prestaciones para su manipulación puesto que se plantea de forma totalmente parametrizada; sin embargo, debemos nuevamente fundamentar los conceptos que darán pie a su posterior definición.

DEFINICIÓN **11** *Sean los conjuntos P, Q y $X \subset (P \cap Q) \in [0,1]$ y sea $T : [0,1] \times [0,1] \to [0,1]$ la transformación que generaliza la intersección clásica llevando a cabo la intersección de conjuntos difusos, como operador lógico satisface la tabla de verdad* ***AND****, define en X a la función de pertenencia $\mu_{P \cap Q}(x)$, además vendrá definida por la operación binaria:*

$$\mu_{P \cap Q}(x) = T[\mu_P(x), \mu_Q(x)], \quad \forall x \in X,$$

que se llama ***norma*** *de x e indica el grado de pertenencia de x a $P \cap Q$ en X.*

DEFINICIÓN **12** *Sean los conjuntos P, Q, R, S y $\mu(x)$ una norma en X que tiene las propiedades:*

1. *Conmutativa: $\mu_{P \cap Q}(x) = \mu_{Q \cap P}(x)$.*
2. *Asociativa: $\mu_{(P \cap Q) \cap R}(x) = \mu_{Q \cap (P \cap R)}(x)$.*
3. *Existe un elemento neutro (el conjunto X) tal que: $\mu_{P \cap X}(x) = \mu_P(x)$.*
4. *Monótona creciente:*

$$Si\ \mu_P(x) \leq \mu_Q(x)\ y\ \mu_R(x) \leq \mu_S(x) \Longrightarrow \mu_{P \cap R}(x) \leq \mu_{Q \cap S}(x); \quad \forall x \in X,$$

entonces $\mu(x)$ es una norma triangular o ***t-norma*** *de x en X.*

Ahora sólo veáse cuáles son las diferencias entre las t-normas y las t-conormas (*conormas triangulares* o s-normas), ya que por esto se dice que son operaciones duales.

DEFINICIÓN **13** *Sea una aplicación de la forma $T^* : [0,1] \times [0,1] \to [0,1]$, definida de manera análoga con una t-norma (definiciones 11 y 12) a excepción de que:*

- *Una t-conorma generaliza la unión de conjuntos difusos.*
- *Como operador lógico, satisface la tabla de verdad* ***OR****.*
- *$\mu(x)$ indica el grado de pertenencia de x a $P \cup Q$ en X,*

entonces T^* es una ***t-conorma*** *en X.*

Operación	t-norma	t-conorma
Min-Max (Gödel)	$T(\mu_A, \mu_B) = min(\mu_A, \mu_B)$	$T^*(\mu_A, \mu_B) = max(\mu_A, \mu_B)$
Producto-suma	$T(\mu_A, \mu_B) = \mu_A \mu_B$	$T^*(\mu_A, \mu_B) = \mu_A + \mu_B - \mu_A \mu_B$
Łukasiewicz	$T(\mu_A, \mu_B) = max(\mu_A + \mu_B - 1, 0)$	$T^*(\mu_A, \mu_B) = min(\mu_A + \mu_B, 1)$
Frank's	$T_p(\mu_A, \mu_B)$ $= \log_p \left(1 + \frac{(p^{\mu_A}-1)(p^{\mu_B}-1)}{p-1}\right)$	$T_p^*(\mu_A, \mu_B)$ $= 1 - T_p(1-\mu_A, 1-\mu_B)$

Tabla 4.1: Operadores de t-normas y t-conormas empleados usualmente en LAMDA.

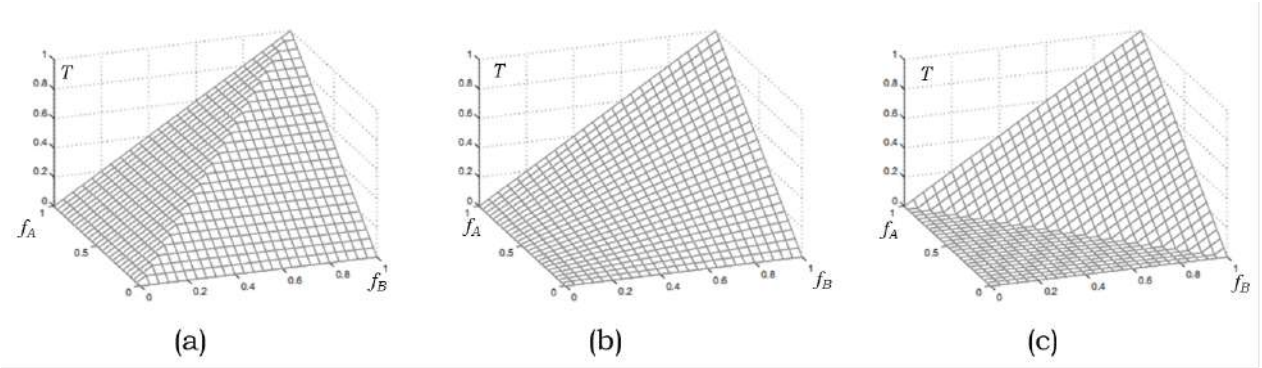


Figura 4.3: Gráficos de las t-normas (a) Gödel, (b) producto y (c) Łukasiewicz en términos de dos funciones de pertenencia f_A y f_B.

Se tienen en la literatura diferentes tuplas $\{T(\bullet,\bullet), T^*(\bullet,\bullet)\}$ que cumplen con lo estipulado por las definiciones 11, 12, 13 y además son continuas $\forall x$. En particular para LAMDA, suponiendo dos funciones de pertenencia μ_A y μ_B, las ecuaciones de la Tabla 4.1[11] son tuplas $\{T(\mu_A, \mu_B), T^*(\mu_A, \mu_B) |\ T^*(\mu_A, \mu_B) = 1 - T(1-\mu_A, 1-\mu_B)\}$, usadas para construir *conectivos difusos* que sirven para parametrizar una partición difusa. Véanse tres de ellas en la Figura 4.3. Se verán los conectivos difusos a mayor detalle con posterioridad.

Cabe mencionar que, según se observa en la Figura 4.3, una t-norma toma su nombre debido a que es un triángulo la figura geométrica que se proyecta sobre el plano $\overline{T\, f_B}$ (así mismo en el plano $\overline{T\, f_A}$); sin embargo, una t-norma (así también una t-conorma) puede convertirse en una *n-norma*[73] así como se agreguen argumentos a la aplicación de la Definición 12 (así mismo para la aplicación suscitada por la Definición 13), de modo que tendríamos una tupla con una forma que luciría como lo indica (4.1):

$$\{T(\mu_1, \mu_2, ..., \mu_n), T^*(\mu_1, \mu_2, ..., \mu_n)\}. \tag{4.1}$$

Dado que se han apuntalado los fundamentos necesarios, se habrá de retomar el tema de la partición difusa que ya habíamos mencionado que sería de mayor flexibilidad en comparación con la ya definida partición difusa simple. Se hace referencia explícitamente a la *partición difusa estricta*, que se define a continuación[73]:

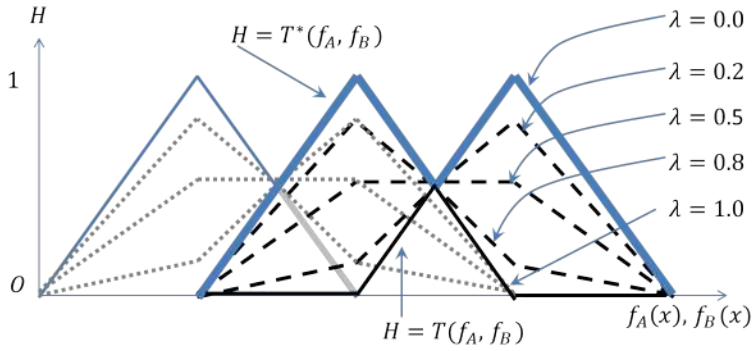


Figura 4.4: Ilustración de la acción básica de parametrización y manipulación de una partición de $X \subset \mathbb{R}^1$ que es posible realizar mediante un conectivo híbrido.

Definición **14** *Sea $X \ni x$ un espacio de entrada, dada una lógica difusa caracterizada por una terna de De Morgan[75]; una t-norma $T(\mu_1, \mu_2)$, una función de negación $N(\bullet)$ y por lo tanto una t-conorma $T^*(\mu_1, \mu_2) = N(T(N(\mu_1), N(\mu_2)))$, además escribimos $T^*\{\mu_k\}_1^l$, según (4.1). Es la n-conorma $T^*\{\mu_k\}_1^l$ única para k debido a la conmutatividad de $T\{\mu_k\}_1^l$, de forma que una* ***partición difusa estricta*** *es una colección de subconjuntos difusos tales que $\forall x \in X$:*

1. $T^*\{\mu_k(x)\} > 0$,
2. $T(\mu_k(x), \mu_j(x)) < 1 : \ \forall k \neq j \in \{1, 2, ..., l\}$.

4.1.2. Conectivos híbridos linealmente compensados

LAMDA estructura su fase de decisión en términos del concepto de *conectivo híbrido*[9] (el cual es un conectivo difuso, dada una lógica difusa en X). Como ya se había mencionado, un conectivo difuso es usado para parametrizar una partición difusa de funciones de pertenencia (adecuaciones), de forma que dicha partición presenta mayor traslapo entre las clases a medida que el conectivo es *menos exigente*; por el contrario, si el conectivo es *más exigente*, el traslapo entre clases es menor hasta el grado en que las clases llegan a ser mutuamente excluyentes, tal como se observa en la Figura 4.4. El concepto de conectivo difuso, significa una función de interpolación entre una *t-norma* T (intersección difusa total) y su operación dual, una *t-conorma* T^* (unión difusa total)[17]. Dicha interpolación es llevada a cabo mediante una combinación convexa de la forma $h(x, y) = \lambda x + (1 - \lambda)y$. Cuando un conectivo difuso H se encuentra expresado en términos de una correspondencia lineal dada por una combinación convexa entre una t-norma y su dual t-conorma, en el espacio X, entonces se tiene una *2-copula* que articula matemáticamente a un *conectivo híbrido linealmente compensado*. LAMDA usa el conectivo híbrido linealmente compensado de la ecuación (4.2) para obtener

los *Grados de Adecuación Global*, que determinan si un objeto de entrada pertenece (se adecúa más) a una clase, tal como se verá más adelante.

Definición **15** *El conectivo híbrido H asociado con una 2-copula T, dado un número real $\lambda \in [0,1]$, es la operación binaria en $[0,1]$ definida por:*

$$H = \lambda T + (1-\lambda)T^*, \tag{4.2}$$

donde: T^* es el operador dual de T (los cuales pueden ser una tupla como la de (4.1)) y λ es el *parámetro de exigencia* que permite *modular* la partición de X, modificando por ende su aspecto; es decir, si se escoge $\lambda = 0$, entonces $H = T^*$, lo cual produce la unión completa de todas las clases. Antagónicamente, si llevamos al parámetro de exigencia a su valor máximo, $\lambda = 1$, entonces $H = T$ y se produce la intersección completa entre todas las clases[16]. Refiérase a la Figura 4.4 donde se halla claramente esquematizado el concepto de la Definición 15, expresado por la Ecuación (4.2).

Cabe aclarar que no es la partición original dada por $\{\mu_i\}_1^l$ la que se modifica; de hecho, no es así en lo absoluto, sino que sólo se toma como subyacente (digamos que como parámetro) para que en función de ésta, una ‘‘nueva partición’’ parametrizada surja, reduciendo o aumentando el número de clases (con respecto a la original) proporcionalmente al valor de λ. Véase un hecho notable (inicialmente tómese en cuenta el intervalo $[0.5, 1.0]$), también la Figura 4.4, que para $\lambda = 0.5$ el número de clases en la nueva partición es exactamente la mitad, permitiendo un traslape total entre ellas, incluso modificando los puntos de corte originales y adquiriendo un grado de separabilidad nulo[2][72]; sin embargo, si $\lambda = 1.0$ el traslape desaparece, provocando que el número de clases se duplique con respecto a la partición subyacente (observándose así un grado de separabilidad total, así como un replanteamiento en el soporte de cada clase). Por otro lado, cuando $\lambda < 0.5$ el número de clases se conserva respecto del subyacente, conservando también el grado de separabilidad (así como el traslape entre ellas a través de los puntos de corte κ_t originales). Se destaca además, que en general para $\lambda > 0.0$ las clases generadas por el conectivo híbrido, dejan de ser *normales*[3][72]; es decir, $\forall k : \sup_x f_{C_k}(x) < 1, \quad \forall x \in X$.

Una forma más general de interpretar lo anterior es que cuando la tolerancia[4] es máxima, si al menos un descriptor x_j del sujeto observado $\mathbf{x}^{(i)}$ se adecua considerablemente al menos un parámetro ρ_j de alguna clase existente C_k, entonces se determina que $\mathbf{x}^{(i)}$ se adecua a la clase C_k en un grado considerable, de manera que es posible asignar $\mathbf{x}^{(i)}$ a C_k. De manera

[2]El grado de separabilidad D no es nulo, de hecho es 0.5. Sin embargo escribimos que es nulo desde el punto de vista de que la altura máxima de las clases para $\lambda \in [0.5, 1.0]$ es de 0.5, de modo que bajo esta observación se podría decir que la separabilidad es nula.

[3]Se define a un conjunto difuso A como *normal* si $\exists x \in \text{supp } A : \sup_x f_A(x) = 1$.

[4]En [73], se define al *parámetro de tolerancia* como $\beta = 1 - \lambda$, dado que la tolerancia es el complemento de la exigencia.

inversa (dual) cuando la tolerancia es mínima, el algoritmo estima que la totalidad de los descriptores de $\mathbf{x}^{(i)}$ debe adecuarse a todos y cada uno de los parámetros del representativo $\boldsymbol{\rho}_k$ de la clase C_k, con un alto grado, para que se produzca en buena medida la posibilidad de asignación.

Nótese entonces que λ permite adoptar valores intermedios de pertenencia a una clase, de forma que en una situación dada, en la cual tengamos un número determinado de clases almacenadas, el número de descriptores de un sujeto de entrada que no se reconozcan estará directamente relacionado con el parámetro de exigencia λ que se escoja. De modo que, si se hace variar λ, se tendrán todos los posibles criterios para formar particiones del espacio de características; es decir, que es posible determinar una partición diferente en X para cada elección de λ.

Véase entonces cómo una partición difusa queda restringida por la Definición 14, así como por otro lado, queda parametrizada y manipulable mediante un conectivo híbrido linealmente compensado.

4.1.3. Funciones de adecuación y densidades de probabilidad

Ya se ha mencionado en la Sección 2.7.2 que una función de *densidad* de probabilidad[5] determina la manera en que se hallan dispuestos los clusters de una partición. Se vio también que para el caso de LAMDA se tiene una partición difusa y además se ha mostrado a detalle cómo está caracterizada; sin embargo, se ha expuesto hasta el momento una idea un tanto generalizada al respecto, ya que la mayoría de las aplicaciones de LAMDA no implementan funciones de pertenencia de tipo triangular o trapezoidal, por lo general hallamos implementaciones en las cuales se usan densidades de probabilidad con formas más aplicables a situaciones del mundo real; tal es el caso de la densidad Gaussiana (o densidad normal) y la densidad binomial. Existen en teoría de probabilidad diferentes densidades; aún así, [73] elige las dos ya mencionadas como factibles de implementar, además de hacer una extensión de la densidad binomial: función de pertenencia *binomial-distancia*.

Sépase que aunque se usan funciones de densidad de probabilidad como funciones de pertenencia, no quiere decir ésto que los valores de $x \in X$ (X es el espacio de entrada o de características) sean variables aleatorias en un espacio muestral (eventos) asociadas a una probabilidad. Dado que tanto una función de pertenencia como una función de probabilidad (y una función de densidad) están definidas en [0, 1], se puede llegar a la confusión de dos conceptos que sólo comparten el intervalo de valores del contradominio para el cual están definidos, pues un grado de pertenencia (y por lo tanto de adecuación) no es en absoluto de naturaleza estadística o probabilística[72].

[5]Se dice que se tiene una función de *distribución* de probabilidades cuando el dominio de dicha función es discreto. En un dominio continuo, se tiene una función *densidad* de probabilidad.

La última afirmación se basa en los hechos de que para una densidad de probabilidad se busca obtener el área bajo la curva en cuestión para determinar la probabilidad acumulada (o sea, la densidad de probabilidad), mientras que para una lógica difusa en X el área bajo la misma curva es un concepto de naturaleza totalmente distinta: la *masa de la clase*[73] (conjunto difuso), sobre la cual aplica la función de densidad.

Aclarado lo anterior, vamos a dejar de llamar ''función de densidad'' a las curvas en cuestión y llamémosles simplemente función de adecuación; a modo de evitar confusiones. Ahora simplemente se toma la determinación incauta de describir únicamente las propiedades a partir de las cuales, se establece que cada función de adecuación es idónea para implementarse en LAMDA.

Se adoptan a continuación las ideas planteadas en teoría de probabilidad para definir las propiedades de las que ya habíamos hablado[76].

Función de adecuación Gaussiana *(densidad Gaussiana)*

Se dice que X sigue una ley *normal* de media μ y varianza σ^2, si su función de densidad f está dada por:

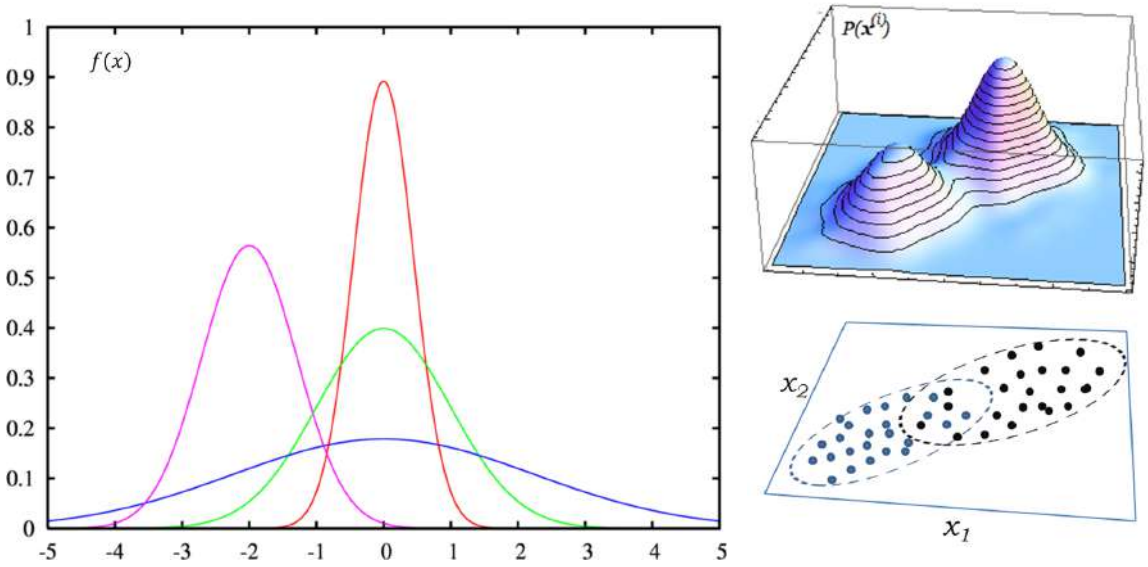
$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2}.$$

Es posible interpretar a μ como un parámetro de traslación; esto quiere decir que esta cantidad indicaría dónde se coloca el centro de la *campana de Gauss*. Así también σ^2 funciona como parámetro de dispersión (Obsérvese la Figura 4.5a). En términos de clustering, se está hablando del centro paramétrico de la clase. Véase la Figura 4.5b para mayor referencia.

Las implementaciones directas de la función de adecuación gaussiana toman una forma un poco más simple y familiar para quienes estudian lo relacionado con clustering:

$$\mu_j(x_j) = e^{-\frac{1}{2\sigma_{kj}^2}(x_j-\rho_j)^2}, \tag{4.3}$$

donde σ_{kj}^2 es la j-ésima componente del vector varianza del cluster C_k. Obsérvese que para la ecuación de la función de adecuación, Ecuación (4.3), la media μ de la densidad de probabilidad se ha sustituido convenientemente por ρ_j, que es la j-ésima componente del representativo $\boldsymbol{\rho}_k$ de la clase k. Además se observa en la Figura 4.5b que los clusters adoptan una alineación diagonal y una forma elipsoidal[11] en X. El factor multiplicativo de la función de densidad gaussiana ya no aparece en la función de adecuación, ya que induce modificación en la altura de la curva. Esto último no sería deseable, puesto que en términos de una función de pertenencia, ésta dejaría de ser normalizada.



(a) (b)

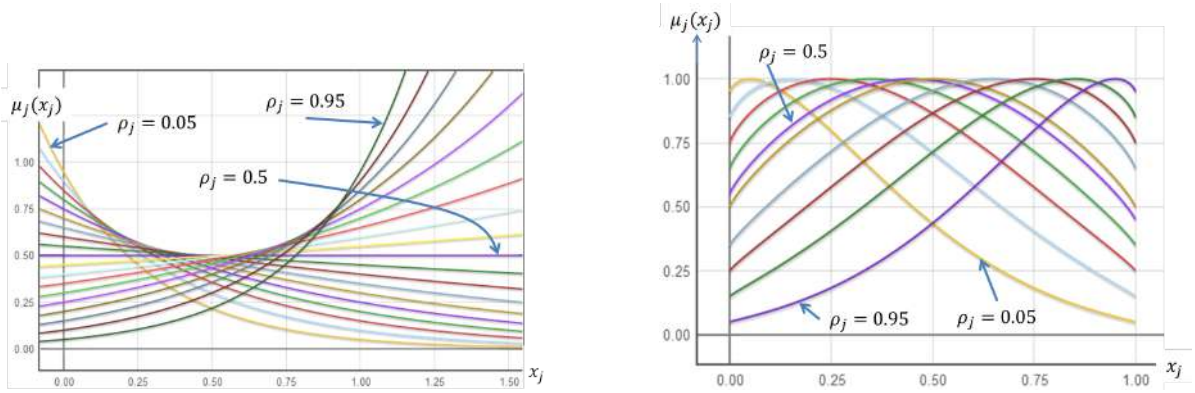
Figura 4.5: Ilustración de la función de (a) densidad Gaussiana para $X \subset \mathbb{R}^1$ y (b) para $X \subset \mathbb{R}^2$.

Función de adecuación Binomial *(densidad Binomial)*

A continuación hablaremos de la *función de densidad de probabilidad binomial*, que se usa como base para construir la función de adecuación Binomial. La densidad binomial tiene un enfoque común en términos de la forma que debería tener un cluster, sin embargo tiene además propiedades excelentes en el sentido de que a fin de cuentas, el algoritmo debe decidir si el objeto observado pertenece o no a alguna de las clases. Desde el punto de vista de probabilidad, esta densidad binomial realza los valores de la variable aleatoria x para los cuales se determina que una secuencia de n experimentos dan como resultado un *éxito* o un *fracaso*[76]. A medida que los resultados se acercan a puntos intermedios, se dice que es inútil generar una estimación, puesto que hay la misma probabilidad de que haya tanto un fracaso como un éxito. Sea p la probabilidad de que ocurra un éxito y $1-p$ la probabilidad de que ocurra un fracaso, entonces el esquema anterior se plantea mediante:

$$f(x) = \left(\begin{array}{c} n \\ x \end{array} \right) p^x (1-p)^{1-x}.$$

En [73], se hace el bosquejo de un escenario hipotético para esquematizar cómo los resultados intermedios pasan de ser anodinos en el punto de vista probabilístico a ser de vital importancia para la clasificación en LAMDA. Se tienen, supóngase, 3 cables tendidos hasta un arreglo de sensores (tres también); dos de los tres cables se hallan conectados a dichos sensores, que se encuentran instalados a una distancia imaginariamente considerable.



(a) (b)

Figura 4.6: Ilustración de (a) la familia de curvas de la función de pertenencia binomial y (b) de la familia de curvas de la función de pertenencia binomial-distancia para $X \subset \mathbb{R}^1$.

Es de suponer que, bajo estas condiciones, el sensor que no está conectado a su respectivo cable no emite ninguna señal útil para el sistema de medición; en vez de ello, el cable posiblemente esté siendo afectado por señales espurias del ambiente (ruido), tal vez con demasiada variabilidad, induciendo a su vez información irrelevante al sistema de medición. En este caso, se considera que existe la misma probabilidad de que el valor de la señal sea el máximo y de que sea el mínimo ($p = \frac{1}{2} = 1 - p$). Se ha dicho que para la clasificación con LAMDA esta información intermedia es muy importante, ya que el hecho de que un descriptor no aporte información sobre un fracaso o un éxito, no supone que el resultado no sirva, sino más bien al contrario: si se supone el mismo escenario hipotético y se supone además que LAMDA es usado para la supervisión de los sensores, el hecho de que uno de ellos quede desconectado aporta al algoritmo la información necesaria para determinar que se tiene un caso de *ausencia de información* (concepto que se verá en las Sección 4.2), lo cual ayuda a determinar que el sensor en cuestión está fallando o se desconectó. Dada la justificación planteada por el bosquejo anterior, se ve que en casos en donde las señales que se miden (descriptores) tengan una naturaleza dicotómica, la función de pertenencia binomial es, por ende, idónea. No así en otro caso. Defínase entonces la función de pertenencia binomial para la k-ésima clase:

$$\mu_j = \rho_j^{x_j}(1 - \rho_j)^{1-x_j}, \tag{4.4}$$

donde μ_j es la j-ésima componente del vector de pertenencia $\boldsymbol{\mu}_k$ y ρ_j es la j-ésima componente del centro paramétrico de la clase k. El elemento combinatorio (donde se incluye al número de experimentos n) que multiplica a la densidad de probabilidad binomial, se ha descartado para definir la función de pertenencia respectiva, puesto que induce un efecto de mayor dicotomicidad (evidentemente indeseable además de innecesario) en la familia de curvas; aparte, no se considera en una lógica difusa un número determinado de experimentos y se requiere que el valor máximo de cualquier función de adecuación sea 1, lo cual también impide dicho factor combinatorio.

A pesar de todo, el uso de la ecuación (4.4) puede inducir inestabilidad al algoritmo para valores de las componentes del centro paramétrico alrededor de 0.5, aún cuando se consideren como no habituales. Nótese lo anterior en la parte central de la Figura 4.6a.

Por último, obsérvense los valores extremos de la familia de curvas en la Figura 4.6a, en donde son crecientes las curvas parametrizadas con $\rho_j > 0.5$ y asignan mayor pertenencia a valores de x_j mayores a 0.5. Contrariamente, la familia de curvas que se definen de manera decreciente, se hallan parametrizadas con $\rho_j < 0.5$ atribuyendo mayor pertenencia a valores de x_j menores a 0.5. Para $\rho_j = 0.5$ se define una recta en 0.5, simultáneamente se observa que las curvas pertenecientes a $\rho_j \approx 0.5$ se aproximan a la curva constante en 0.5.

Función de adecuación Binomial-distancia

La función de adecuación binomial-distancia es una extensión de la binomial, propuesta por Julio Waissman[9], para corregir el problema de la inestabilidad que pudiera inducir (4.4) al algoritmo para valores de ρ_j cercanos a $\frac{1}{2}$:

$$\mu_j = \frac{\rho_j^{x_j}(1-\rho_j)^{1-x_j}}{x_j^{x_j}(1-x_j)^{1-x_j}}. \tag{4.5}$$

La Ecuación (4.5), se rige bajo los mismos principios que (4.4). Véase en la Figura 4.6b la familia de curvas para esta función de pertenencia.

4.2. Ausencia de información

Con base en el principio expuesto para articular la justificación en cuanto al uso de la densidad de probabilidad binomial, se fundamenta también la existencia de una *Clase No Informativa* (*NIC*, del inglés *Non-Informative Class*). La clase NIC es un elemento fundamental en el aprendizaje del algoritmo y se basa en la ausencia de información (máxima entropía[73]) acerca de un posible objeto de entrada; antes de procesar tal objeto, algoritmo se halla ignorante sobre su condición, de modo que deberá suponerse que dicho objeto podría tener el mismo grado de pertenencia para todas las clases. Bien podríamos decir que tiene la misma probabilidad de pertenecer a cualquier clase.

Bajo el supuesto anterior se plantea la imposición de un umbral de pertenencia para cada descriptor del objeto en observación, de forma tal que se considere que cualquier nivel de pertenencia cuyo valor se halle por debajo del dicho umbral (≤ 0.5), implique carencia de sentido para proceder a su clasificación. En vez de ello, ha de suponerse que la información

de que dispone el algoritmo es insuficiente para llevar a cabo tal clasificación; es decir, el objeto observado no se reconoce.

Una vez confirmada la ausencia de información acerca de un objeto en observación, LAMDA puede llevar a cabo dos acciones:

1. Notificación de rechazo del objeto en observación.
2. Uso del objeto no reconocido para suscitar una nueva clase.

Para la primera opción no hay más que hacer que determinar que el objeto observado, es parte del conjunto de objetos cuya información al respecto es desconocida. Sin embargo, para la segunda opción el procedimiento a seguir todavía sugiere mayor atención, puesto que es tal vez la parte más interesante y simple de LAMDA: La fase de aprendizaje.

4.3. Aprendizaje

LAMDA es un algoritmo cuya base de conocimiento es evolutiva, lo cual hace que cuando uno observa que dicha base de conocimiento se modifica y se mejora con cada objeto que se observa, resulte en una sensación de intriga que a la vez vuelve emocionante el hecho de pensar en que otra aplicación se le puede dar (no obstante del atisbo de informalidad que pudiera imprimir tal comentario).

La gran característica de esta habilidad de auto-aprendizaje (*Self-Learning*) es, como ya se mencionó, el proceso de refinamiento de la información de la que dispone el algoritmo para procesar un objeto de entrada. Dicha información comienza con una clase inicial: la clase NIC. Ya hemos dicho en que se basa la existencia de esta clase no informativa, de modo que basado en ésto y dado que en principio es la única clase de que dispone el algoritmo, sea entonces X el espacio de entrada (o espacio de características), supóngase al primer objeto de entrada $\mathbf{x}^{(1)} = (x_1, x_2, ..., x_j, ..., x_m)^T \in X \subset [0,1]^m$, entonces, puesto que no hay clases existentes (excepto la NIC), dicho objeto se determina como desconocido. Considerando que el algoritmo se encuentra dispuesto para aprender (no sólo rechazar un objeto desconocido), entonces el algoritmo determina que deberá crearse una nueva clase, usando como clase existente a la NIC (en términos algorítmicos: clase 0; C_0) y usando su centro paramétrico $\boldsymbol{\rho}_0 = (0.5, ..., 0.5, ..., 0.5)^T \in [0,1]^m$ como ‘‘*ingrediente*’’ de la nueva clase; de modo que ésta posea suficiente generalidad para usos futuros (que veremos un poco más adelante). La clase C_1 formará parte ahora de la base de conocimiento del algoritmo y estará representada por su respectivo centro paramétrico $\boldsymbol{\rho}_1 = (\rho_1, \rho_2 ..., \rho_j, ..., \rho_m)^T \in [0,1]^m$, cuyas componentes a su vez serán determinadas mediante:

$$\rho_j = 0.5_j + \frac{x_j - 0.5_j}{2}. \tag{4.6}$$

Nótese cómo participan en la ecuación (4.6) las componentes de $\boldsymbol{\rho}_0$.

El proceso anterior ha de llevarse a cabo aún cuando el algoritmo tenga una base de conocimiento previa; es decir, en toda situación en la cual un objeto de entrada sea determinado como desconocido y el algoritmo se halle dispuesto para aprender una nueva clase, en base a un objeto de entrada que no se adecue lo suficiente a las clases existentes.

El siguiente escenario que implica aprendizaje surge cuando ya se cuenta con una base de conocimiento previa, disponible para el algoritmo. En este caso se tiene ya una partición $C = \{C_1, C_2, ..., Ck, ..., C_l\}$ del espacio de características, donde cada C_k está representada por su centro paramétrico $\boldsymbol{\rho}_k = (\rho_1, \rho_2..., \rho_j, ..., \rho_m)^T \in [0,1]^m$. Supóngase ahora que se tiene un objeto de entrada $\mathbf{x}^{(i)} = (x_1, x_2, ..., x_j, ..., x_m)^T \in X$ y suponemos también que el algoritmo ha determinado que $\mathbf{x}^{(i)}$ se adecua a la clase C_k suficientemente y más que a las demás; por lo tanto, este objeto es asignado a dicha clase. En virtud de lo anterior, $\mathbf{x}^{(i)}$ será usado para aumentar la cuenta de objetos asignados a dicha clase y como ingrediente para refinar (actualizar) el centro paramétrico k mediante[73]:

$$\hat{\rho}_j = \rho_j + \frac{x_j - \rho_j}{N+1}, \tag{4.7}$$

donde: $N \in \mathbb{Z}^+ - \{0,1,2\}$ es el número de elementos asignados a la clase C_k y $\hat{\rho}_j$ es la j-ésima componente actualizada de $\boldsymbol{\rho}_k$ después de agregado (aprendido) un nuevo elemento a C_k.

El número de objetos asignados a una clase no puede ser menor que tres, puesto que en la *infancia del aprendizaje*[73] ya se tubo esta cantidad de objetos: Se ha tenido uno desde el hecho de que la clase NIC se usa como ingrediente para suscitar una nueva clase (la clase NIC es el primer elemento de cualquier clase) y el segundo elemento es en sí el objeto que crea la clase nueva.

Otro punto importante para considerar acerca del aprendizaje es que según [73] (aclarado también mediante una asesoría directa provista por Tatiana Kempowsky[9]) es posible prescindir de la ecuación (4.7) y simplemente calcular las componentes de $\boldsymbol{\rho}_k$ obteniendo la media aritmética de todos los objetos asignados a la clase C_k; sin embargo, esto restaría generalidad al proceso de reconocimiento, puesto que dicho cálculo no proporciona el refinamiento necesario de los centros paramétricos, provocando que el algoritmo sólo reconozca como miembros de alguna clase a objetos con demasiado parecido a su centro paramétrico; no precisamente que sean adecuados al mismo.

4.4. Grados de Adecuación

En esta sección vamos a tomar los conceptos ya vistos en la sección 4.1 para hacer de dichos conceptos una connotación que respalda de manera sólida la terminología que se usa

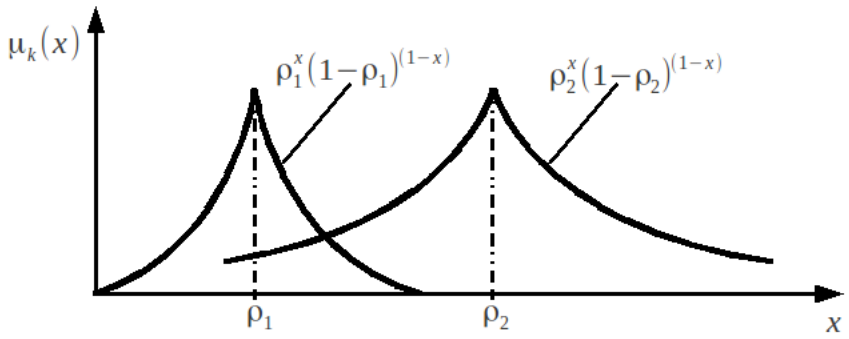


Figura 4.7: Partición difusa mediante la función de adecuación Binomial para $X \subset \mathbb{R}^1$ y diferentes valores de ρ_k.

regularmente en la literatura que trata a LAMDA.

4.4.1. Grado de Adecuación Marginal (MAD)

Para estar en condiciones de calcular grados de adecuación es necesario que tanto el sujeto observado como las clases existentes (suponiendo que ya hubo aprendizaje) sean de las mismas dimensiones. Durante el desarrollo del proyecto que incentiva la realización de este trabajo de tesis, se ha recopilado información suficiente para determinar que la mayoría de los descriptores que LAMDA tendría que supervisar son de naturaleza dicotómica[6]; por lo tanto, se ha elegido implementar la función de adecuación binomial. Cabe destacar que la notación cambia un poco con respecto de lo establecido en la Sección 4.1.3. En general, el vector de adecuaciones $\boldsymbol{\mu}_k$ cuyas componentes las define una función de adecuación, es en sí el *grado de adecuación marginal* del objeto de entrada $\mathbf{x}^{(i)}$ a la k-ésima clase, por lo que ahora lo denotamos por $MAD(\mathbf{x}^{(i)}|C_k)$, de las siglas en inglés *Marginal Adequation Degree*. El término ‘‘marginal’’ viene de la teoría de probabilidad e indica que una función es calculada para una variable individual (probabilidades marginales), cuando regularmente se calcula para un conjunto de variables (probabilidades conjuntas)[76]. De modo que se tiene una versión actualizada de la ecuación (4.4) para definir a la j-ésima componente del grado de adecuación marginal para la clase C_k, en términos de su centro paramétrico $\boldsymbol{\rho}_k$:

$$MAD(x_j|C_k) = \rho_{kj}^{x_j}(1-\rho_{kj})^{1-x_j}. \tag{4.8}$$

Se ha denotado la componente del centro paramétrico con los dos índices kj para imprimir mayor claridad. Véase en la Figura 4.7 para verificar la forma de los clusters en la partición difusa.

[6]Dicha tendencia dominante en la naturaleza de las señales, se puede observar a detalle en los conjuntos de datos de entrada que fueron recolectados para cajón del PA y que se muestran en el Capítulo 5.

4.4.2. Grado de Adecuación Global (GAD)

El *grado de adecuación gobal* (GAD por sus siglas en ingles, *Global Adequation Degree*) es un concepto cuyas bases se hallan cimentadas en los conceptos expuestos en la Sección 4.1.2, donde vimos a detalle cómo una partición parametrizada a partir de una partición simple se crea mediante un conectivo híbrido linealmente compensado, el cual se halla planteado en términos de una combinación convexa entre una n-norma y su dual n-conorma. Por lo tanto, el vector de grados de adecuación global (denotado por $GAD(\mathbf{x}^{(i)}|\{C_k\}_1^l) \in [0,1]^l$) es obtenido para un objeto de entrada $\mathbf{x}^{(i)} \in [0,1]^m$ con respecto a cada clase existente a partir de la partición creada por el conectivo difuso elegido según:

$$GAD\left(\mathbf{x}^{(i)}|\{C_k\}_1^l\right) = \lambda T\left[\{MAD(x_j|C_k)\}_{j=1}^m\right] + (1-\lambda)T^*\left[\{MAD(x_j|C_k)\}_{j=1}^m\right], \quad (4.9)$$

donde $\{T, T*\}$ es cualquiera de las tuplas (n-norma, n-conorma) listadas en la Tabla 4.1. El cálculo del vector de adecuaciones globales supone un paso previo al paso final del algoritmo de clustering difuso LAMDA, el cual se expone de manera condesada en la Sección 4.5. El paso final del que estamos hablando es muy sencillo, sólo habrá que aplicar el operador $max\{\}$ al vector de adecuaciones globales (siendo las componentes de dicho vector los argumentos del operador). Lo anterior es debido a que se considera que desde el punto de vista de la partición generada por los conectivos híbridos, la clase que mejor se adecue al objeto de entrada (incluyendo la clase no informativa), será la que mayor grado de pertenencia global otorgará a dicho objeto.

4.5. Esquema general del algoritmo de clustering difuso LAMDA

Sea el objeto de entrada $\hat{\mathbf{x}}^{(i)} = (\hat{x}_1, \hat{x}_2, ..., \hat{x}_j, ..., \hat{x}_m)^T \in \mathbb{R}^m$, sea también la colección de clases $C = \{C_1, C_2, ..., C_k, ..., C_l\}$ representada por su matriz de centros paramétricos $\boldsymbol{\rho} = (\boldsymbol{\rho}_1, \boldsymbol{\rho}_2, ..., \boldsymbol{\rho}_k, ..., \boldsymbol{\rho}_l)$; donde cada $\boldsymbol{\rho}_k = (\rho_1, \rho_2 ..., \rho_j, ..., \rho_m)^T \in [0,1]^m$.

Lo primero que realiza el algoritmo con cada descriptor que captura es hacer un tratamiento sencillo de *normalización*, el cual consiste en una aplicación $N : \mathbb{R}^m \to [0,1]^m$, de tal manera que los valores adquiridos directamente desde el exterior[7] sean escalados a cantidades difusas (“*fuzzification process*”[74]), sin importar las magnitudes en que dichos valores se encuentren expresados. Para tal efecto se tiene la siguiente ecuación, expresada en

[7]Se ha dicho *“directamente desde el exterior”* por simplicidad; no obstante, se sabe que los valores que se adquieren desde el exterior son procesados mediante rutinas de extracción de características; tales características son las que en realidad se denominan como descriptores, se normalizan y son procesadas por el algoritmo.

términos de cada descriptor $\hat{x}_j$ no normalizado:

$$x_j = \frac{\hat{x}_j - x_{min}}{x_{max} - x_{min}}, \tag{4.10}$$

donde: $x_j \in [0,1]$ es el descriptor normalizado, j-ésima componente del objeto de entrada normalizado $\mathbf{x}^{(i)} = (x_1, x_2, ..., x_j, ..., x_m)^T \in [0,1]^m$, $\{x_{min}, x_{max}\} \subset \mathbb{R}$ son los valores mínimo y máximo que el descriptor $\hat{x}_j \in \mathbb{R}$ puede llegar a alcanzar (medido directamente del exterior). Los valores $\{x_{min}, x_{max}\}$ pueden ser proporcionados por el usuario experto del sistema que se busca tratar o bien el algoritmo puede estar programado para establecerlos de manera automática, implementando una rutina de normalización en tiempo real (tal como se ha hecho en este trabajo de tesis).

La segunda fase de LAMDA (obsérvese la Figura 4.8) consiste en determinar los vectores de Grados de Adecuación Marginal (MAD) para cada clase, que están dados por la función de adecuación de $\mathbf{x}^{(i)}$ a cada clase C_k. Ya se ha visto en la Sección 4.1.3 que existen diversas maneras de calcular la función de adecuación. En particular, se ha elegido la ecuación (4.8):

$$MAD(x_j|C_k) = \rho_{kj}^{x_j}(1 - \rho_{kj})^{1-x_j}.$$

El tercer paso es calcular los Grados de Adecuación Global (GAD) mediante la ecuación (4.9):

$$GAD\left(\mathbf{x}^{(i)}|\{C_k\}_1^l\right) = \lambda \min\left[\{MAD(x_j|C_k)\}_{j=1}^m\right] + (1-\lambda)\max\left[\{MAD(x_j|C_k)\}_{j=1}^m\right]. \tag{4.11}$$

La Ecuación (4.11) se construye usando el conectivo híbrido *mín-máx* de la Tabla 4.1, por simplicidad.

En la cuarta fase sólo se debe calcular:

$$\hat{k} = \arg\max_k GAD\left(\mathbf{x}^{(i)}|\{C_k\}_1^l\right),$$

donde $\hat{k} \in \{0, 1, 2, ..., k, ..., l\}$ es índice de la clase a la cual será asignado el objeto de entrada.

El algoritmo tiene tres opciones posteriores a las cuatro fases ya mencionadas para terminar con el procesamiento de un objeto de entrada:

1. Asignar el objeto de entrada en una de las clases ya aprendidas con anterioridad y sólo informar $\hat{k}$ (si el algoritmo no se halla en modo de aprendizaje).
2. Si $\hat{k} = 0$ y el algoritmo está en modo de aprendizaje entonces crear una clase nueva haciendo uso de la ecuación (4.6).

$$\rho_j = 0.5_j + \frac{x_j - 0.5_j}{2}.$$

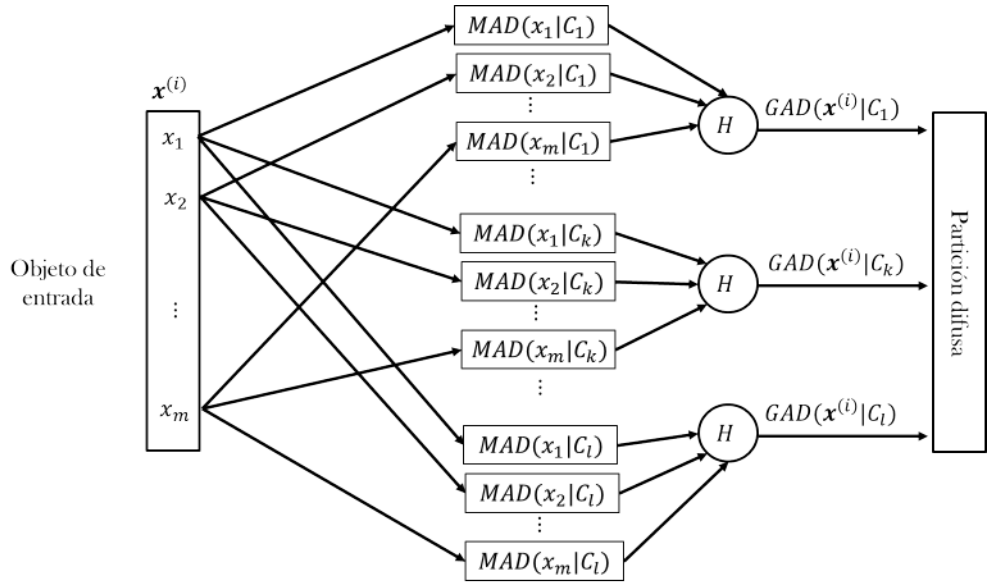


Figura 4.8: Fases del *Learning Algorithm for Multivariable Data Analysis* (LAMDA).

3. Si $\hat{k} > 0$ y el algoritmo está en modo de aprendizaje, entonces asignar el objeto a la clase respectiva y refinar su centro paramétrico mediante (4.7):

$$\hat{\rho}_j = \rho_j + \frac{x_j - \rho_j}{N+1}.$$

■

4.6. Propiedades de LAMDA

LAMDA posee las siguientes propiedades[11]:

1. El aprendizaje puede ser supervisado y no supervisado.
2. Capacidad de procesamiento simultáneo de sujetos cuantitativos y cualitativos.
3. El aprendizaje es efectuado en forma secuencial e incremental.
4. El reconocimiento está basado en conectivos híbridos linealmente compensados, los cuales son usados para calcular Grados de Adecuación Global (GADs) en base a los Grados de Adecuación Marginal (MADs) de un sujeto a cada clase.

5. La nula distición de un sujeto significa homogeneidad caótica y es modelada mediante una clase no informativa (NIC, *Non Informative Class*), misma que proporciona un umbral entre la posibilidad de clasificar o no a un sujeto, obteniéndose así una clase nueva.

6. Existe la posibilidad de subdividir clases ya existentes, de tal forma que se tengan diferentes subclases de elementos de una misma clase. La selectividad de esta subclasificación puede ser regulada por un parámetro llamado *parámetro de tolerancia*.

Capítulo 5

Implementación de redes neuronales y resultados de la Comparación con LAMDA

En este capítulo se expone de manera detallada lo referente a la implementación (en caso de requerirse. Véanse los objetivos, Sección 1.4) y pruebas de los algoritmos propuestos para su análisis. Primero se expone cómo fueron realizadas las pruebas piloto que se llevaron a cabo mediante herramientas experimentales, las cuales se especifican. Se expone después qué resultados se han obtenido de dichas pruebas, cuya naturaleza también es expuesta en este capítulo, a efecto de justificar de manera cuantitativa cómo se ha llegado a una conclusión sobre el algoritmo que resultaría idóneo para su implementación en el núcleo de software del SPAT-135.

5.1. Implementación experimental de Redes neuronales de retro-propagación

Como primera fase del trabajo práctico que se ha realizado, se explican los detalles sobre cómo fueron llevadas a cabo las pruebas necesarias para evaluar el rendimiento de diferentes arquitecturas de redes neuronales, de modo que fue posible elegir una arquitectura que se adecúe a cada cajón del PA. Cada red estaría implementada en una herramienta experimental de reconocimiento de patrones de funcionamiento correcto del Sistema de Pilotaje Automático.

Dicha implementación ha sido llevada a cabo en MATLAB, con la ayuda del asistente para la creación y entrenamiento de redes neuronales llamado *''Neural Network Toolbox''*

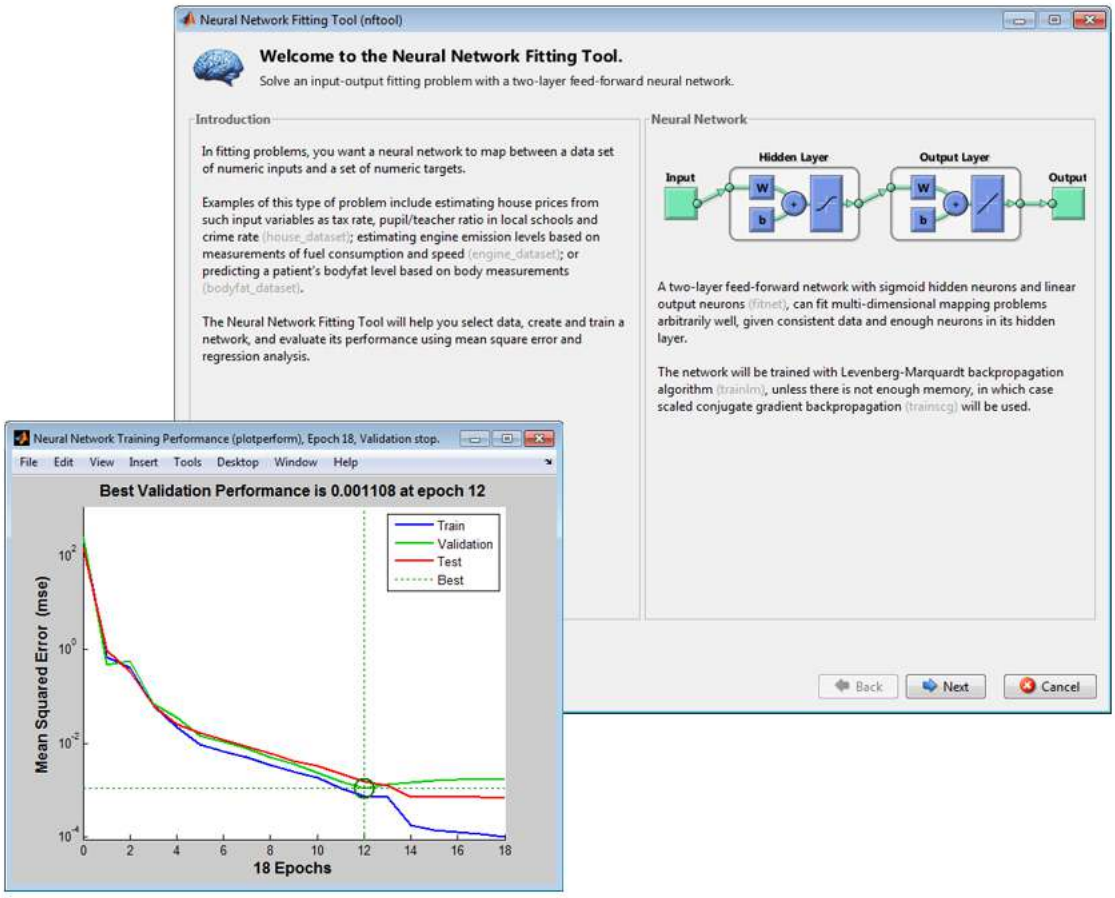


Figura 5.1: Captura de pantalla del ‘‘Neural Network Toolbox’’ de MATLAB.

(véase Figura 5.1), el cual cuenta con una interfaz gráfica para realizar el diseño de redes de dos capas (una capa oculta y la capa de salida) de manera muy flexible, creando objetos (cada red neuronal es modelada mediante un objeto) cuyas propiedades y métodos son totalmente manipulables usando código de tal manera que sea posible formar una red con diferentes funciones de activación, cualquier número de neuronas, cualquier número de capas y cualquier arquitectura. Lo anterior provee la oportunidad de experimentación sin tener como problema el código intermedio; sino más bien, el problema que se desea resolver. En este trabajo de tesis, sólo se desea verificar el rendimiento en la inferencia de resultados de reconocimiento de patrones de entrada, de modo que resulta anodino cualquiera que pudiera ser el tiempo de aprendizaje de la red, lo que justifica el uso del entrenamiento por retro-propagación. Adicionalmente hubo la necesidad forzosa de optar por la segunda opción expuesta en la Sección 3.4.2, puesto que no se tenía la información necesaria para evitarla. Ha sido una de las etapas que más tiempo, dedicación y cautela ha requerido por parte del equipo de desarrollo. Inicialmente en este capítulo, se reporta la capacidad de generalización de diferentes arquitecturas de red como medida de rendimiento y como criterio de elección de una arquitectura en particular.

Se han visto en la Sección 3.4.7 (paso 7 del proceso de diseño) los diferentes enfoques para dirigir el entrenamiento de una red neuronal. Se ha mencionado en este respecto la elección de la opción 3: Prueba-entrenamiento. Para hacer aún más viable dicha opción, se ha procedido a echar mano de las prestaciones que nos ofrece el ''Neural Network Toolbox'', el cual hace posible el uso de algunas variantes del algoritmo tradicional de retro-propagación, mismas que se encuentran incluidas en MATLAB. El método de Levenberg-Marquardt (LM), es una variante del entrenamiento por retro-propagación, ampliamente usado para aplicaciones de optimización. Según la descripción del LM revisada en [82], cuando la solución del problema se encuentra lejos del punto óptimo el algoritmo se comporta como se vio en la sección 3.1 (gradiente-descendente), recorriendo lentamente la superficie de error pero garantizando la convergencia en un mínimo. Una vez que se estima que el proceso de entrenamiento se halla cerca del punto óptimo, el algoritmo procede a comportarse como en el *Método de Gauss-Newton*[82], asumiendo que la función de error es de forma cuadrática y buscando el mínimo de dicha función. En la propia documentación de ayuda de MATLAB, se hayan algunas pruebas genéricas que indican que para aplicaciones donde se requiere aproximar una función, el método de LM es la elección idónea debido a su rapidez y efectividad.

Como ya se ha mencionado, en este trabajo de tesis se busca obtener resultados experimentales del desempeño de una red neuronal durante la fase de reconocimiento de patrones de entrada (Figura 5.2); no profundizando en lo referente al proceso de entrenamiento, puesto que el usuario final del SPAT-135 no está considerado para realizar dicho proceso. Por lo tanto, se ha elegido el uso del método de LM como variante del algoritmo tradicional gradiente-descendente. Además, es de suma importancia centrar esfuerzos en la conclusión eficiente de los objetivos de este trabajo.

Para cada cajón del PA se recolectaron un número de muestras de datos brutos cercano a 70,000. Se hicieron experimentos con diferentes tamaños de ventana (128, 256, 512, 1024 y 2048) configurados en el filtro pasa-bajas que se implementó para eliminar el ruido de las señales de CD, y se obtuvieron mejores resultados para un ancho de ventana $w = 512$, presentándo el adecuado mejoramiento en la forma de onda de dichas señales con resolución temporal suficiente.

De lo anterior, se tiene en la Tabla 5.1, la cantidad de muestras en tres pruebas de cada cajón y la cantidad de muestras obtenidas después del filtraje. Por ejemplo: si se tienen N muestras de la señal o descriptor $x_j(t) = \{x_j(1), x_j(2), ..., x_j(N)\}$, variante en el tiempo discreto t, y a dicho conjunto de muestras se le aplica un filtro por media aritmética con ancho de ventana w, entonces la salida del filtro es la señal $\bar{x}_j(t') = \{\bar{x}_j(1), \bar{x}_j(2), ..., \bar{x}_j(n)\}$, donde $n = \frac{N}{w} \in \mathbb{Z}^+$ y t' es el tiempo reducido para el nuevo conjunto de muestras (conjunto de entrada). En realidad un filtro de estas características, no modifica de esta manera el número de muestras original; sin embargo, se ha procedido de esta forma porque 1) fue necesario reducir costo computacional en la implementación del núcleo de reconocimiento de patrones final, para sincronizar los tiempos de procesamiento de este software con los tiempos de procesamiento de la tarjeta de adquisición de datos; y también para 2) no procesar las

Cajón del PA	N	n	m
Cajón I PA/CMC	174,592	341	28
Cajón II PA/CMC	251,392	491	12
Cajón III PA/CMC	100,352	196	20
Cajón IV PA/CMC	145,920	285	19
Cajón I CML/CMR	153,088	299	17
Cajón II CML/CMR	230,912	451	15

Tabla 5.1: Cantidad de datos recolectados por cajón del PA: N es el número de muestras de datos brutos, n es el número de muestras por cajón, para el conjunto de entrada construido después del filtraje y m es la dimensión del espacio de entrada (número de descriptores de entrada, para las redes neuronales $n_I = m$).

muestras ‘‘aplanadas’’ por el filtro, ya que tenían todas el mismo valor (el promedio de cada ventana, repetido w veces) y resultaba redundante para los experimentos, tratar de reconocer con LAMDA o RNAs dichas muestras repetidas.

Durante el desarrollo de los siguientes apartados del capítulo, se detallan los resultados cuantitativos del desempeño de las redes neuronales que se han propuesto probar para cada cajón del PA, con el fin de obtener la arquitectura de red con mejor capacidad de generalización a partir de un conjunto de muestras de entrada en particular. De la misma manera, para cada cajón, se utiliza el mismo conjunto de entrada para entrenar a la herramienta de software SALSA cuyo núcleo de reconocimiento de patrones está basado en LAMDA. Para cada cajón, se ha inspeccionado el conjunto de datos brutos adquiridos, dicho conjunto consta de 93 pruebas en total para los seis cajones de las cuales, se han seleccionado tres pruebas sucesivas que se considera cuentan con mayor variedad de cambios de estado de los patrones de comportamiento. No se confunda el término ‘‘prueba’’ con el término ‘‘experimento’’, ya que con el primero se hará referencia a una prueba que se hace a un cajón en particular, con fin de verificar su funcionameinto; con el segundo, se hará referencia a uno de los doce (trece para los primero dos cajones) experimentos realizados para cada cajón, y que corresponden a una red neuronal por cada experimento, de manera que se usarán de manera indistinta los términos ‘‘experimento’’ y ‘‘red’’. Cabe señalar para cada experimento, se realizó la inicialización aleatoria de los pesos sinápticos y el entrenamiento de una red en quince ocasiones, seleccionándose el mejor resultado (con base en el MSE de validación) de estos quince entrenamientos para realizar el experimento correspondiente. Se exponen con mayor precisión los experimentos hechos con el Cajón `I` - PA/CMC, con la intención de exponer de manera general los detalles que se considerarán como implícitamente llevados a cabo en las secciones posteriores, de manera que en estas últimas, sólo se exponga al lector la información relevante para los objetivos de este trabajo.

5.2. Cajón I - PA/CMC

5.2.1. Redes neuronales artificiales

En la Tabla 5.2, se tienen los resultados del desempeño de trece redes neuronales que se entrenaron, probaron y validaron para reconocer patrones sin normalizar[1] provenientes de las salidas del Cajón I - PA/CMC del Piloto Automático (Material NM-79, Chopper), durante tres[2] pruebas de diagnóstico que realiza actualmente el personal de mantenimiento en el taller del STC.

Aún cuando la cantidad seleccionada de datos facilita los experimentos, se ha diseñado un proceso experimental que ha considerado trece experimentos por cada cajón (trece redes neuronales, al menos en esta sección), lo que mantiene un alto, pero necesario, costo de tiempo para llevarlos a cabo.

En la Tabla 5.2, la primera columna muestra el número de experimento (número de red) que se ha realizado para el cajón en cuestión, la segunda columna muestra el Error Cuadrático Medio[3] mínimo (dado que el entrenamiento se realiza por lotes) que se obtuvo durante el procesamiento del conjunto de validación, del cual se puede tener una estimación cuantitativa de si el desempeño de alguna red en cuestión tendrá capacidad de generalización (plasticidad). Ya se verá más adelante que la medición anterior sólo proporciona un indicador y que es necesario proponer un experimento adicional (descrito también más adelante en esta sección; Ecuación 5.1) para tener mayor certeza sobre la capacidad de generalización de cada red. En la tercera columna, se tiene la arquitectura de cada red, seguida por el enfoque de dimensionamiento (entre paréntesis). La arquitectura de cada red está especificada por su número de nodos de entrada[4] n_I, número de nodos en la capa oculta n_h y número de nodos en la capa de salida n_O. Las muestras del conjunto de entrada 28-dimensional[5] de la Figura

[1] Se han hecho pruebas con valores normalizados también; sin embargo, debido al ruido que presentan las señales y la pérdida de información que implica un mayor filtrado, no fue viable procesar dichas señales normalizadas con las redes neuronales, de modo que se opta por utilizar señales sin normalizar para obtener mejores resultados.

[2] Para cada prueba fue necesario adquirir un promedio de 70,000 muestras, cada una de entre 12 y 28 descriptores. Dado el volúmen de datos que se han adquirido, se ha tomado en cuenta únicamente un conjunto de tres pruebas, de un total de 30 (en promedio) que se hacen para diagnosticar a cada cajón del PA. Lo anteior se considera suficientemente justificado para cumplir con los requerimientos académicos de este trabajo.

[3] En adelante MSE (denotado por ξ_{av} en la Sección 3.1), por sus siglas en inglés: *Mean Squared Error*.

[4] La capa de entrada no se considera en el número de capas de la red, de modo que decimos que se prueban redes de 2 (dos) capas: capa oculta y capa de salida.

[5] Para los casos de los cajones I PA/CMC y IV PA/CMC, el número de salidas indicado en la Tabla 5.1 es diferente al real, puesto que al momento de la realización de esta investigación, el equipo de desarrollo de hardware del proyecto no contaba con la información necesaria. Lo anterior no afecta los objetivos de este trabajo.

Red/ experimento	$MSE_{\text{mín}}$ (validación)	Arq. ($n_I/n_h/n_O$)
1	4.46×10^{-2}	[28/01/01] (pruning)
2	2.56×10^{-2}	[28/02/01] (pruning)
3	5.65×10^{-3}	[28/03/01] (pruning)
4	8.49×10^{-3}	[28/04/01] (pruning)
5	7.38×10^{-3}	[28/05/01] (pirámide)
6	1.43×10^{-3}	[28/06/01] (pruning)
7	3.68×10^{-3}	[28/07/01] (pruning)
8	4.48×10^{-2}	[28/08/01] (pruning)
9	2.81×10^{-3}	[28/09/01] (pruning)
10	6.43×10^{-3}	[28/10/01] (pruning
11	3.07×10^{-3}	[28/11/01] (pruning)
12	3.90×10^{-3}	[28/12/01] (pruning)
13	1.89×10^{-2}	[28/27/01] (Xiao-Hu Yu)

Tabla 5.2: Datos resultantes del entrenamiento de trece redes neuronales con diferente enfoque de dimensionamiento, para el Cajón I PA/CMC. La información de la segunda columna se tomó del valor mínimo del MSE después de 15 (quince) inicializaciones aleatorias de los pesos sinápticos (y entrenamiento respectivo a partir de los mismos). El conjunto completo de entrada contiene 341 patrones (muestras), que se segmentan aleatoriamente en: Entrenamiento = 273 (80 %), Prueba = 34 (10 %) y Validación = 34 (10 %). Haciendo una inspección del conjunto de entrada, se identifican 11 clases que tendrían que reconocerse durante las pruebas 1, 2 y 3.

5.2, se han elegido de manera aleatoria para formar los conjuntos de entrenamiento, prueba y validación (véanse en la Tabla 5.2 más detalles). Una vez integrados éstos, no se modifican a lo largo del proceso de experimentación[6]. La columna número dos de la Tabla 5.2 es la más importante, puesto que muestra cuantitativamente el rendimiento de la red neuronal para la que ha sido obtenida cada medición. Dichas medidas, así como su interpretación, son claves para saber qué red podría ser la más adecuada para su aplicación en el diagnóstico del PA del metro. Se han entrenado trece redes con diferentes arquitecturas, empezando desde una neurona en la capa oculta ($n_h = 1$), pasando por el dimensionamiento que sugiere la regla de la pirámide geométrica (de la ecuación (3.22): $n_h = \sqrt{(28)(1)} = 5.29 \rightarrow 5$; se tiene la arquitectura: $n_I = 28, n_h = 5$ y $n_O = 1$), hasta tener 12 neuronas en dicha capa ($n_h = 12$). Se hace un poco de énfasis en la red número 5 debido a que se especifica en el paso 5 (Sección 3.4.5) que ha de verificarse la viabilidad de esta arquitectura. Por último se verifica el Teorema de Xiao Hu-Yu mediante la construcción de la red número 13 (Arquitectura: $n_I = 28, n_h = 27$ y $n_O = 1$).

Se observa en la Tabla 5.2 que la red número 6 aparentemente supera a todas las demás

[6]Para la red No. 13, el conjunto de entrada se construyó según lo que establece el Teorema 3. En este caso, se usaron 28 muestras (entre dos y tres muestras de cada patrón que se deseaba entrenar a la red) para el conjunto de entrenamiento; de modo que fuera el 80 % del total.

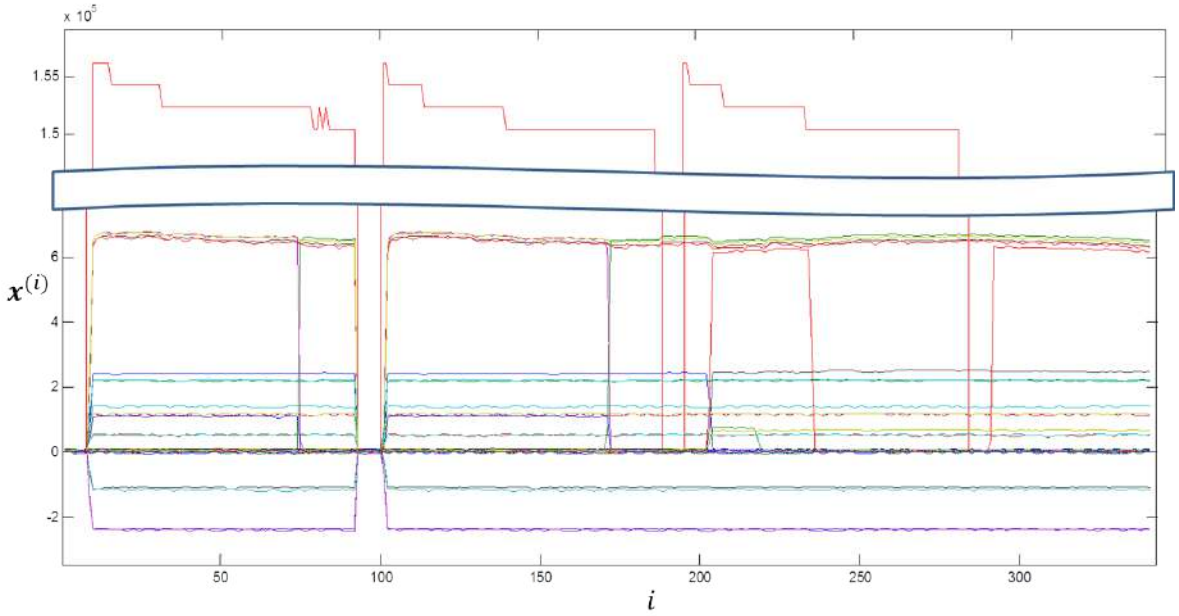


Figura 5.2: Conjunto de entrada 28-dimensional para el entrenamiento de las trece redes neuronales de la Tabla 5.2.

Patrón	Segmento
1	1-4; 7-8; 93-100
2	5-6
3	11-74; 102-171
4	75-92; 172-187; 196-203
5	101
6	188-195
7	204-217
8	218-235
9	236-283
10	284-291
11	292-341

Tabla 5.3: Segmentación de patrones del conjunto de entrada. La primer columna corresponde al número de patrón identificado por inspección del conjunto de entrada; es decir, el valor deseado asignado (t_i) a cada patrón en la salida de una red neuronal. La segunda columna corresponde a la partición del espacio de entrada que se ha realizado de manera manual; es decir, cada segmento de muestras del conjunto de entrada que han sido asignadas a cada patrón descubierto. Nótese que esta tabulación corresponde al dominio y contradominio respectivamente de la función que cada red debe aproximar, desde el punto de vista del Teorema del Aproximador Universal (Teorema 4).

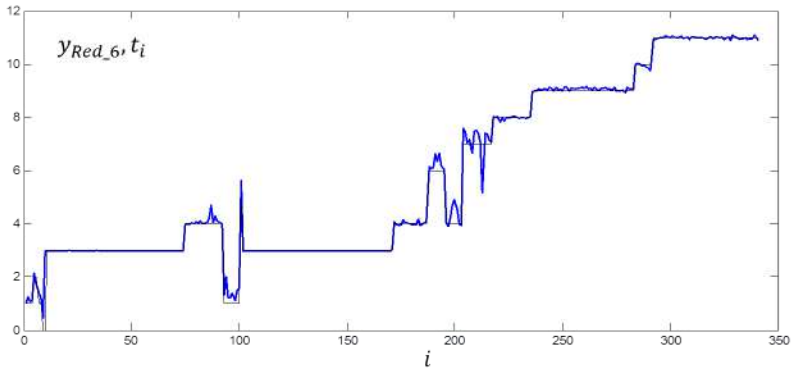


Figura 5.3: Rendimiento durante la validación para la red neuronal del experimento 6 en la Tabla 5.2. Aún cuando presenta el mejor MSE de todos los experimentos, se observan errores considerables al tratar de reconocer los patrones de las muestras cercanas a la 85, 100 y 200. Arquitectura: $n_I = 28, n_h = 6, n_O = 1$.

(con un $MSE_{\text{mín}} = 1.43 \times 10^{-3}$); sin embargo, si después del entrenamiento se procesan los conjuntos de prueba y validación (excluyendo al de entrenamiento) para hacer reconocimiento de patrones (véase la Tabla 5.3) con dicha red, se observan errores importantes que cuestionan severamente su capacidad de generalización en la salida y_{Red_6}, tal como se observa en Figura 5.3. Por otro lado, se siguió el proceso generativo de ‘‘*pruning*’’ obteniéndose resultados ‘‘poco útiles’’, hasta llegar a la red cuya arquitectura presenta 12 neuronas en la capa oculta ($n_I = 28, n_h = 12$ y $n_O = 1$). Aunque se han usado los conjuntos de prueba y validación para intentar validar el desempeño de todas la redes dimensionadas por ‘‘pruning’’ (incluyendo por pirámide geométrica), no fue posible obtener información concluyente acerca de una red que sobresalga prominentemente de las demás, observándose desempeños muy similares al que se observa en la Figura 5.4 (el mejor de todos, como se verá), de manera que se propone un experimento extra: se diseñó un conjunto de validación tomando como base las 341 muestras del conjunto completo de entrada, sumando a éste una función de ruido blanco (aditivo-sustractivo) acotada en su definición para diferentes amplitudes:

$$\{\frac{1}{a}N_k : i \to [-1,1]^m \;\mid\; i = 1,2,...,n\}, \tag{5.1}$$

donde: $a = 1, 10, 20, ..., 90$ es el factor de amplitud para $N_k(\bullet)$, m es la dimensión del espacio de entrada para cada cajón del PA, i es el índice de cada muestra en el conjunto de entrada y $k = 1, 2, .., 9$ es el índice de la curva de tendencia generada para cada amplitud a de la función de ruido; cada conjunto de entrada contiene n muestras. Por lo tanto, el conjunto de funciones (5.1) simula situaciones reales, dado que es ruido precisamente el factor externo que afecta en mayor medida a los patrones de voltaje provenientes de los cajones del PA. Dicha simulación del entorno se efectua afectando cada muestra $\mathbf{x}^{(i)} \in X$ de la siguiente manera:

$$\tilde{\mathbf{x}}^{(i)} = \mathbf{x}^{(i)} + \frac{1}{a}N_k(i); \;\; i = 1,2,...,n$$

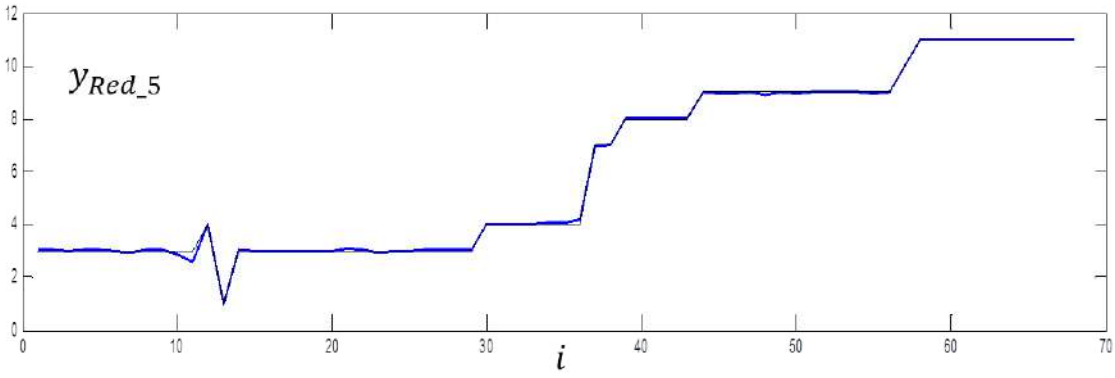


Figura 5.4: Rendimiento durante la validación para la red neuronal del quinto experimento en la Tabla 5.2; se observa una gran precisión al seguir la función descrita por el vector de objetivos. Arquitectura: $n_I = 28, n_h = 5, n_O = 1$.

donde k está emparejado con cada valor de a y $\tilde{\mathbf{x}}^{(i)}$ es la i-ésima muestra modificada por ruido. El código fuente en MATLAB que se implementó para este experimento, se puede consultar en el Apéndice B. En el caso del cajón que se trata en esta sección se sabe que $m = 28$ y $n = 341$ (véase la Tabla 5.1).

Sorprendentemente se obtuvo una mejor capacidad de generalización en la red del experimento 5, que se manifiesta al procesar el conjunto de entrada con diferentes intensidades de ruido blanco aditivo-sustractivo (llámese conjunto modificado al conjunto de entrada adicionado con ruido). En la Figura 5.5 se aprecian 9 curvas resultantes de dicho experimento adicional, cada una representa el valor del MSE para cada arquitectura de red (desde 1 hasta 12 neuronas en la capa oculta; eje horizontal) y para cada amplitud de la función de ruido blanco. Véase cómo para todas las amplitudes, el MSE que presenta la red 5 en su salida y_{Red_5} siempre es el mínimo; mientras que para las redes que se acercan a una neurona en la capa oculta, converge a un valor inaceptable. Por el otro lado, véase también cómo las curvas divergen en valores abrumadoramente inaceptables a medida que el número de neuronas en la capa oculta aumenta a 12, de tal forma que es de suponerse que este comportamiento se multiplique al aumentar aún más n_h. En la Figura 5.6a se muestra la evolución de MSE de validación (curva marcada con un círculo) para la red 5. Véase en el mismo gráfico, como las curvas de error de los tres conjuntos de entrada descienden con pendientes muy similares, hasta que el entrenamiento es detenido por considerarse suficiente el desempeño de esta red; aunque no por llegar al punto óptimo de la curva de error, ya que éste puede estar mucho más abajo si se sigue iterando. Se observa también que en ninguna iteración, la curva de error presenta mínimos locales (al menos hasta que el entrenamiento es detenido).

Otro resultado derivado de los análisis correspondientes al cajón que se considera en esta sección, es que se tiene la verificación práctica del Teorema 3 (Arquitectura: $n_I = 28, n_h = 27$ y $n_O = 1$). Si se observa la evolución del entrenamiento de dicha red, se nota un gran desempeño durante el aprendizaje del conjunto de entrenamiento (con un error de hasta

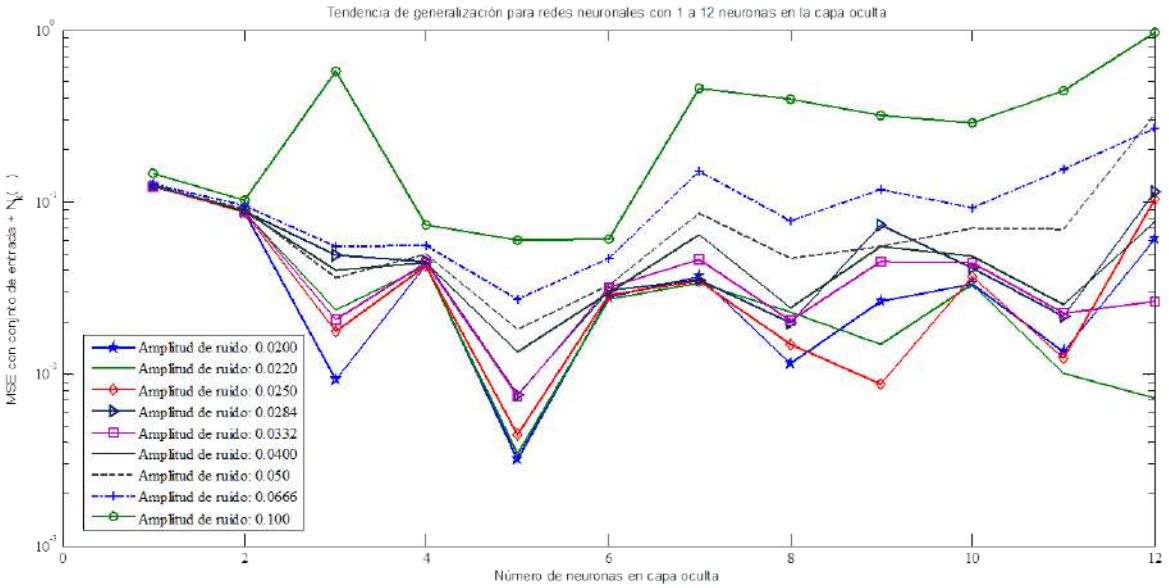


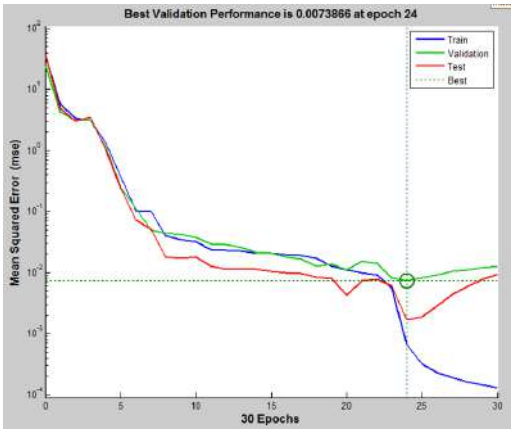
Figura 5.5: Tendencia de generalización por arquitectura de red.

1.21×10^{-17}; Figura 5.6b); sin embargo, al hacer que la red procese un conjunto de entrada un tanto diferente[7], la capacidad de generalización (véase la Figura 5.7) presenta un desempeño de naturaleza totalmente opuesta, cuando la salida y_{Red_13} de la red (línea remarcada) trata de seguir la función inducida por el vector de objetivos (línea delgada). Lo anterior, sugiere claramente que el exceso de neuronas en la capa oculta provoca memorización y, desde luego, mucho sobre-ajuste (‘‘*overfitting*’’) de la red, lo que hace totalmente inviable considerar esta arquitectura como posible opción para un sistema de diagnóstico. El análisis anterior confirma aún más el mejor desempeño de la red cuya arquitectura es: $n_I = 28, n_h = 5, n_O = 1$, diseñada empleando el enfoque de la pirámide geométrica.

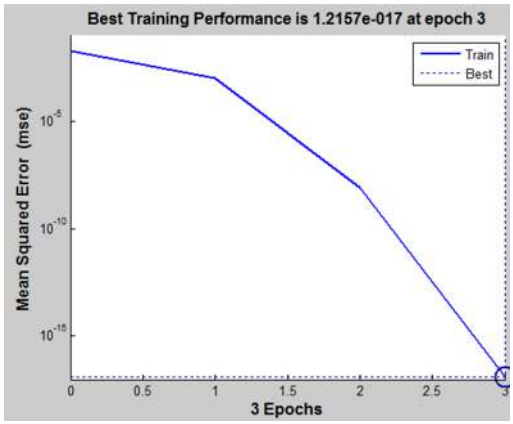
La importancia de la contribución de [54] suena sustancial (se ha demostrado cuantitativamente que no lo es para casos prácticos); sin embargo, el trabajo citado se deslinda del resultado que pueda tenerse al usar cualquier algoritmo de entrenamiento. Además, se ha observado que este enfoque, lejos de poner atención en la utilidad práctica de la red, sólo busca analizar el comportamiento de la curva de error dependiendo del número de entradas, ligado al tamaño de la capa oculta y del conjunto de entrenamiento. Dado que el Teorema 3 no se fundamenta en la articulación de algún algoritmo de entrenamiento en particular, entonces:

> ‘‘*Debe remarcarse que la no existencia de un mínimo local en la superficie de error de la red feedforward, no implica que algún algoritmo de entrenamiento gradiente-descendente no pueda quedar atrapado en un mínimo local*’’[54].

[7]Se diseñó, al igual que en experimentos anteriores, un conjunto de validación tomando como base el de entrada.



(a)



(b)

Figura 5.6: (a) Evolución del entrenamiento de la red 5, dimensionada mediante la regla de la pirámide geométrica y entrenada usando retro-propagación por el método de LM y (b) evolución del entrenamiento de la red 13 dimensionada por el Teorema 3 y entrenada usando retro-propagación por el método de LM.

Como último resultado, se tiene el porcentaje de reconocimiento para la red que se ha seleccionado para el cajón en cuestión ($n_I = 28, n_h = 5, n_O = 1$). Para obtener un resultado plausible en este respecto, se usa el conjunto de entrada modificado por ruido ($a = 20$) y se obtienen las gráficas de la Figura 5.8, donde se puede observar cómo la red 5 confunde algunos patrones que suman un 1.17 % del total (98.8 % de reconocimiento).

Hasta el momento se observan algunas características poco deseables en las redes diseñadas usando el Teorema 3; así también, se puede suponer que el desempeño, en cuanto a plasticidad de una red neuronal, empeora a medida que n_h crece, y se estanca en valores altos cuando $n_h \to 1$. Adicionalmente, se puede apreciar la eficacia de la regla de la Pirámide geométrica. Aún así, es de sugerirse por el momento recorrer varias dimensiones del espacio de pesos sinápticos de modo que se llegue a una arquitectura de tamaño razonable, la cual estaría determinada por el valor mínimo del MSE para un conjunto de entrada modificado por ruido. Será en el transcurso de los experimentos posteriores cuando podamos ir afirmando o desechando con certeza tales hipótesis, de manera que se tenga una conclusión al respecto.

5.2.2. Entrenamiento y validación de LAMDA (a través de SALSA)

La herramienta de software SALSA es muy utilizada por diferentes investigadores que desean obtener detalles sobre todo el proceso que sigue LAMDA al particionar un espacio de entrada y al hacer sólo reconocimiento de patrones una vez que se ha definido una partición

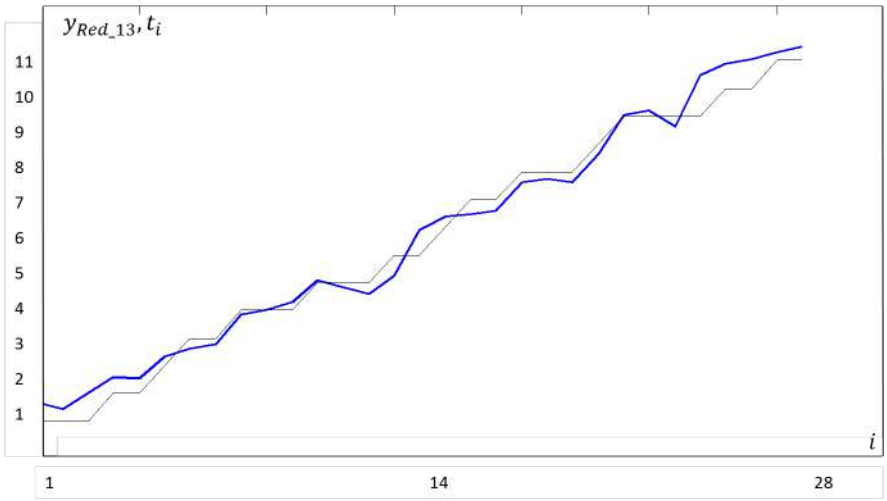


Figura 5.7: Desempeño de la red 13, Cajón I-PA/CMC, al tratar de seguir la función descrita por el vector de objetivos.

(ya sea por autoclasificación o por entrenamiento supervisado). Ya se han visto en el Capítulo 4 los detalles del algoritmo; si el lector además se encuentra interesado en conocer información sobre el uso de SALSA, puede consultar el Manual de Usuario[83] (Véase la Figura 5.9).

Se ha utilizado el mismo conjunto de entrenamiento que para las redes neuronales del apartado anterior, sólo que para LAMDA no es necesario dividir en tres partes (entrenamiento, prueba y validación) el conjunto de entrada. LAMDA tiene la gran ventaja de que puede crear clases de manera automática a partir del conjunto de entrenamiento que le sea provisto. Dicho conjunto de entrenamiento consta de 341 muestras, mismo que fue usado para construir el conjunto modificado por ruido (igual al que se usó para redes neuronales[8], con $a = 20$), con objeto de realizar una fase de reconocimiento que sirva como medida de validación de LAMDA para el cajón bajo estudio.

En la Figura 5.10a se tiene la interfaz gráfica de usuario de SALSA, que muestra tanto el conjunto de entrenamiento normalizado[9] (panel inferior) como el resultado de la clasificación automática (panel superior). Se han probado varios valores del parámetro de exigencia, determinándose al final que empleando $\lambda = 0.6$, se obtiene una clasificación aceptablemente

[8]En experimentos posteriores se usaron valores más grandes de a, con la finalidad de aumentar la dificultad en la tarea de reconocimiento.

[9]El conjunto de entrada sin normalizar (únicamente datos pre-procesados: valores promedio y frecuencia), es exactamente el mismo que el empleado para probar las redes neuronales.

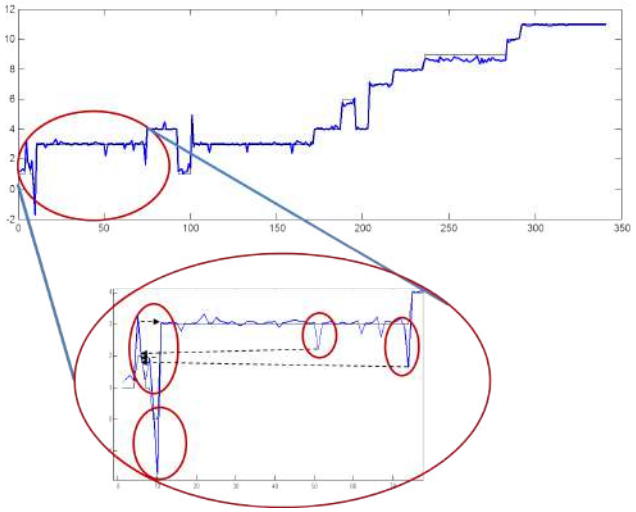


Figura 5.8: Desempeño de la red 5 al tratar de seguir la función descrita por el vector de objetivos cuando el espacio de entrada es un conjunto de entrada modificado por ruido. Las flechas punteadas indican en su origen la salida equivocada y con su punta el patrón que la red ha **confundido**. La otra parte encerrada con una elipse (parte más baja de la figura) es un patrón **no reconocido** y se toma en cuenta en el porcentaje final.

coherente[10] con lo que ya se había observado previamente en el particionamiento por inspección para las redes neuronales; se obtienen también 11 clases, lo que resulta acorde con la Tabla 5.3.

Una vez más, se aclara que no se está haciendo un estudio profundo sobre el desempeño de la fase de aprendizaje de estos algoritmos (LAMDA y RNAs); además, no se ha fijado como punto de comparación el error que cometen dichos algoritmos en el sentido en el que se haría al comparar dos redes neuronales, como en el apartado anterior, sino que se toma como punto de comparación el porcentaje de reconocimiento que pueden alcanzar los clasificadores en fases de reconocimiento, al distinguir, confundir o no reconocer patrones presentes en un conjunto de entrada modificado por ruido. Por lo tanto, en la Figura 5.10b se muestra el gráfico de clasificación (panel superior de la GUI de SALSA) que ha realizado LAMDA cuando el espacio de entrada contiene exclusivamente a un conjunto modificado por ruido. Nótese en dicha figura cómo el algoritmo confunde la muestra número nueve, clasificándola en la clase 5; cuando debería ser en la clase 3. Lo anterior muestra que LAMDA, para esta aplicación y para el cajón en cuestión, confunde el 0.29 % de los patrones; es decir, manifiesta un porcentaje de reconocimiento de 99.70 %.

[10]Se considera coherente debido a que se han obtenido el mismo número de clases, variando el valor de λ; sin embargo, las particiones no son idénticas. Lo anterior no afecta los resultados de los experimentos, puesto que se está evaluando el porcentaje de de reconocimiento. Se tiene la misma consideración para todos los experimentos de este estilo que se hacen a lo largo del capítulo.

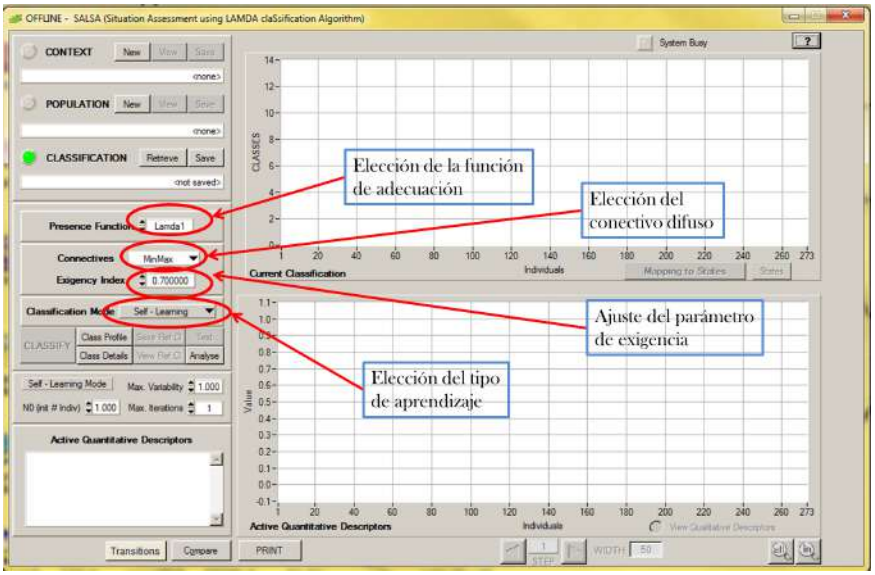
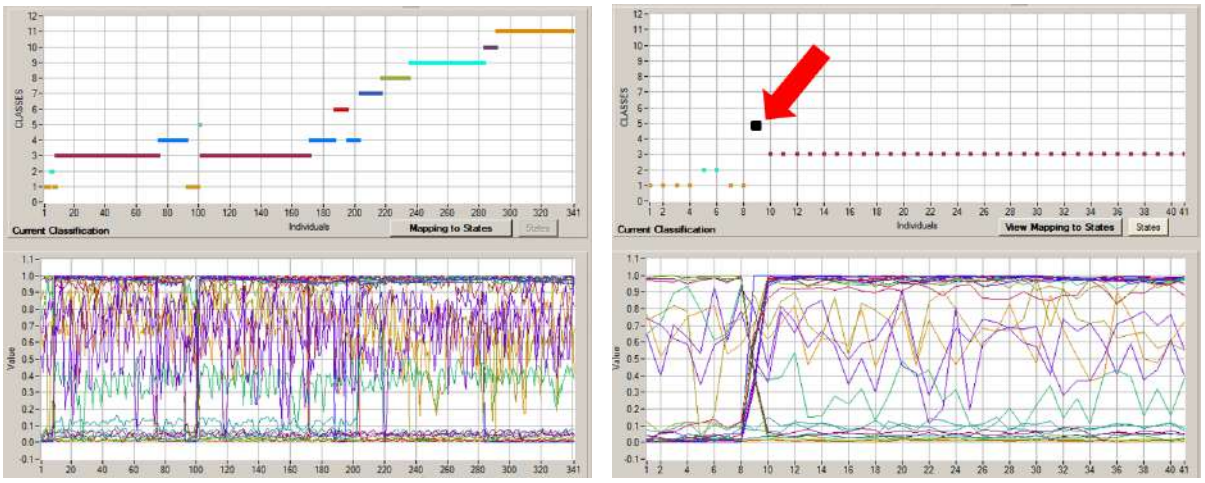


Figura 5.9: Interfaz gráfica de usuario S. A. L. S. A.



(a) (b)

Figura 5.10: Ilustración de (a) Salida de clasificación automática (aprendizaje no supervisado) de S. A. L. S. A. y (b) la salida de clasificación de patrones que hace LAMDA cuando se le presenta el conjunto modificado por ruido ($a = 20$). Nótese la confusión del algoritmo al tratar de clasificar la muestra número 9.

5.3. Cajón II - PA/CMC

5.3.1. Redes neuronales artificiales

Al igual que en la sección anterior, se presentan en la Tabla 5.4 los resultados del MSE mínimo (columna 2) para cada arquitectura de red con $n_h \in \{1, 2, ..., 12\}$; después de 15 inicializaciones aleatorias de los pesos sinápticos. Obsérvese cómo la red 8 presenta el menor MSE para el conjunto de validación; se verá, como en la sección anterior, que esto no es una medida directa y exacta de la capacidad de generalización de dicha red; pues en esta ocasión la red número 4 es la que ofrece mayor desempeño, después de haber procesado 10 conjuntos modificados por ruido de diferente intensidad.

Para el Cajón II, se tiene un espacio de entrada 12-dimensional, lo que implica que la regla de la pirámide geométrica permita inferir la siguiente arquitectura: dados $n_I = 12$ y $n_O = 1$, entonces $n_h = \sqrt{(12)(1)} = 3.46 \rightarrow 3$ (arquitectura: $n_I = 28, n_h = 3, n_O = 1$); por lo que el renglón número 3 de la Tabla 5.4 corresponde a este enfoque de dimensionamiento. Se observa en la Figura 5.11 que en esta ocasión la regla de la pirámide geométrica no resulta tan efectiva; aunque sí aproximada con el mejor valor.

Ya se ha adelantado que la red número 4, dimensionada por ‘‘*pruning*’’ generativo, fue la mejor. Lo anterior puede verificarse en las curvas de la Figura 5.11, donde se observa cómo el MSE con respecto a conjuntos modificados por ruido, de diferente intensidad, aumenta y converge a un rango de valores claramente inaceptables a medida que $n_h \rightarrow 12$. Al igual que para los experimentos del Cajón I, se intuye que si n_h crece aún más, la capacidad de generalización de la red empeora. Por otra parte, para $n_h \rightarrow 1$ sucede algo parecido a lo que se observaba para el Cajón I (sección anterior), sólo que para el Cajón II el MSE converge a un rango de valores inaceptables; en lugar de converger a un valor. Nótese también que, mientras que en el Cajón I el MSE diverge cuando $n_h \rightarrow 12$ (los valores lucen cada vez más dispersos), en el Cajón II convergen a un rango todavía más inaceptable[11] que cuando $n_h = 1$.

Expuesto lo anterior como notas relevantes para esta sección, se muestran ahora los resultados de las pruebas para la medida final de desempeño de la red 4, elegida para el Cajón II. En la Figura 5.12 se observa la evolución del entrenamiento de esta red (la de mejor desempeño; Figura 5.11). El resultado de reconocimiento de patrones de dicha red se puede observar, primero para $a = 90$ en la Figura 5.13a y luego para $a = 1$ en la Figura 5.13b. Dado que ahora se está tomando como resultado de validación el reconocimiento de patrones

[11] Véase en la Figura 5.11, que los conjuntos de valores del MSE para $n_h = 1$ y $n_h = 12$ de hecho se intersectan, por lo que no se puede considerar que el conjunto de valores que se obtienen cuando $n_h = 12$ es totalmente más inaceptable que cuando $n_h = 1$ y viceversa.

Red/ experimento	$MSE_{\text{mín}}$ (validación)	Arq. $(n_I/n_h/n_O)$
1	1.81×10^{-2}	[12/01/01] (pruning)
2	1.68×10^{-4}	[12/02/01] (pruning)
3	2.30×10^{-3}	[12/03/01] (pirámide)
4	1.65×10^{-7}	[12/04/01] (pruning)
5	1.88×10^{-5}	[12/05/01] (pruning)
6	9.00×10^{-9}	[12/06/01] (pruning)
7	6.85×10^{-7}	[12/07/01] (pruning)
8	1.32×10^{-9}	[12/08/01] (pruning)
9	1.29×10^{-8}	[12/09/01] (pruning)
10	9.63×10^{-6}	[12/10/01] (pruning
11	1.65×10^{-6}	[12/11/01] (pruning)
12	1.73×10^{-6}	[12/12/01] (pruning)
13	8.77×10^{-4}	[12/11/01] (Xiao-Hu Yu)

Tabla 5.4: Datos resultantes del entrenamiento de trece redes neuronales con diferente enfoque de dimensionamiento, para el Cajón II PA/CMC. La información de la segunda columna se tomó del valor mínimo del MSE después de 15 (quince) inicializaciones aleatorias de los pesos sinápticos (y entrenamiento respectivo a partir de los mismos). El conjunto completo de entrada contiene 491 patrones (muestras), que se segmentan aleatoriamente en: Entrenamiento = 393 (80 %), Prueba = 49 (10 %) y Validación = 49 (10 %). Haciendo una inspección del conjunto de entrada, se identifican 7 clases que tendrían que reconocerse durante las pruebas 33, 34 y 35.

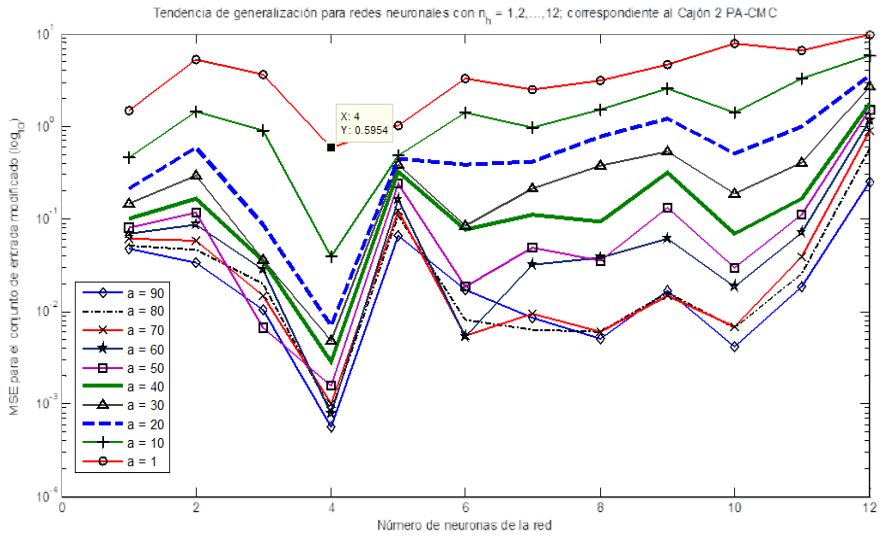


Figura 5.11: Tendencia de generalización por arquitectura de red.

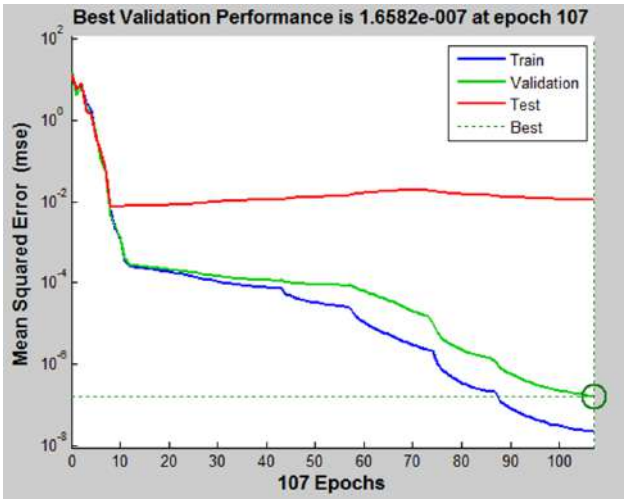


Figura 5.12: Evolución del entrenamiento de la red 4 ($n_I = 12, n_h = 4, n_O = 1$). La curva de error para el conjunto de validación se halla marcada con un círculo y se observa cómo es aproximadamente igual a la curva de entrenamiento (más abajo) después de la época número 40.

para $a = 1$, con un conjunto de entrada modificado por ruido de mayor intensidad[12] (Figura 5.13b), se tiene que la red seleccionada confunde o desconoce 180 muestras de un total de 491, lo que implica un 36.86 % (63.14 % de reconocimiento).

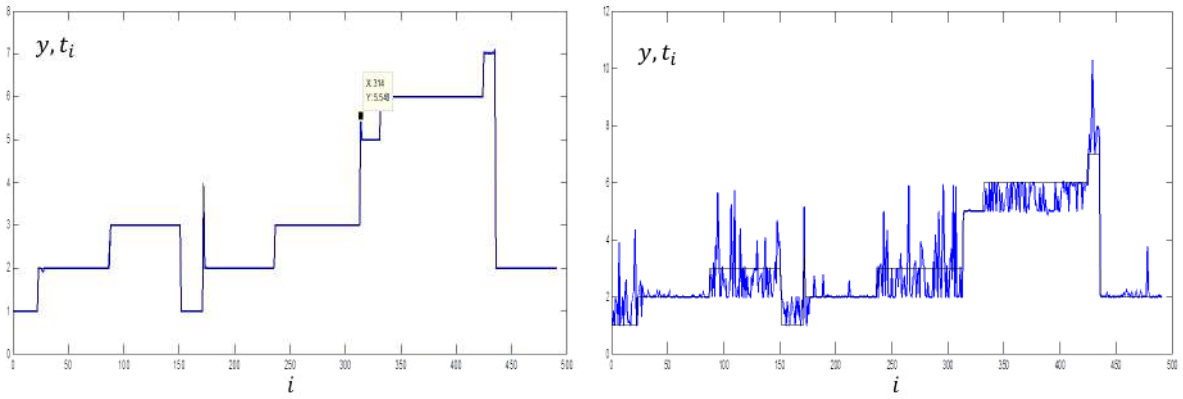
Como último resultado, se experimentó con la red dimensionada mediante el Teorema 3, obteniendo resultados similares a los de la sección anterior. Refiérase a la Figura 5.14a, donde se observa la evolución del entrenamiento de dicha red, es evidente que la curva de error del entrenamiento (curva que desciende más rápido) se mantiene sin mínimos locales; sin embargo, la capacidad de generalización de la red es muy pobre (Figura 5.14b) y tanto la arquitectura como el método de entrenamiento siguen siendo inviables para propósitos prácticos.

5.3.2. Entrenamiento y validación de LAMDA (a través de SALSA)

Nuevamente se ha utilizado el mismo conjunto modificado por ruido (Figura 5.15a), usado para obtener las gráficas de tendencia de generalización de las redes neuronales probadas en esta sección. En la Figura 5.15b se observa el conjunto de entrada original.

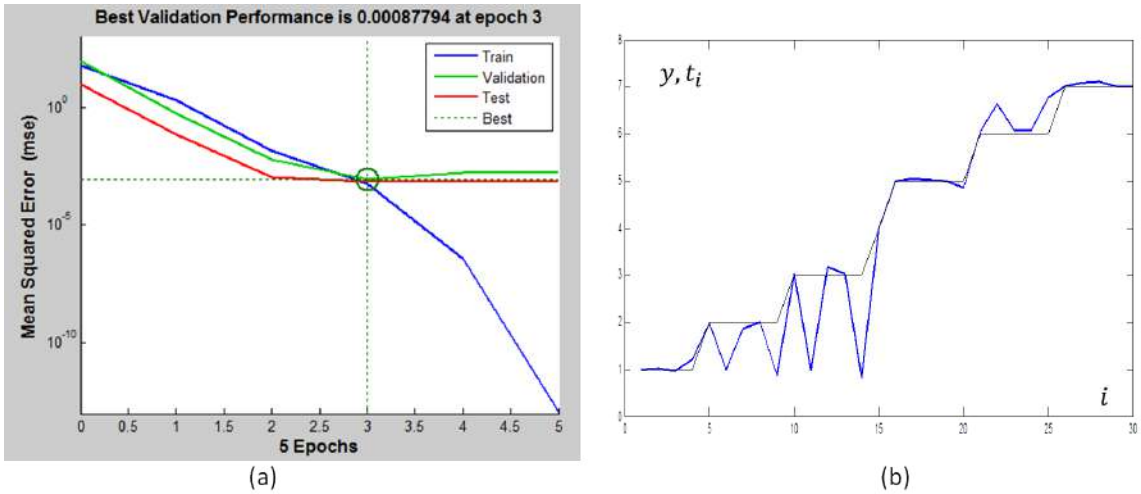
En lo referente al desempeño de reconocimiento de patrones que se obtuvo para LAMDA, se ha fijado el parámetro de exigencia en $\lambda = 0.6$ (como se hizo para el Cajón I), lo que

[12]La mayor intensidad de ruido para contaminar el conjunto de entrada original, se obtiene con $a = 1$; es decir, $\frac{1}{a}N_k(t) \in [-1, 1]$.



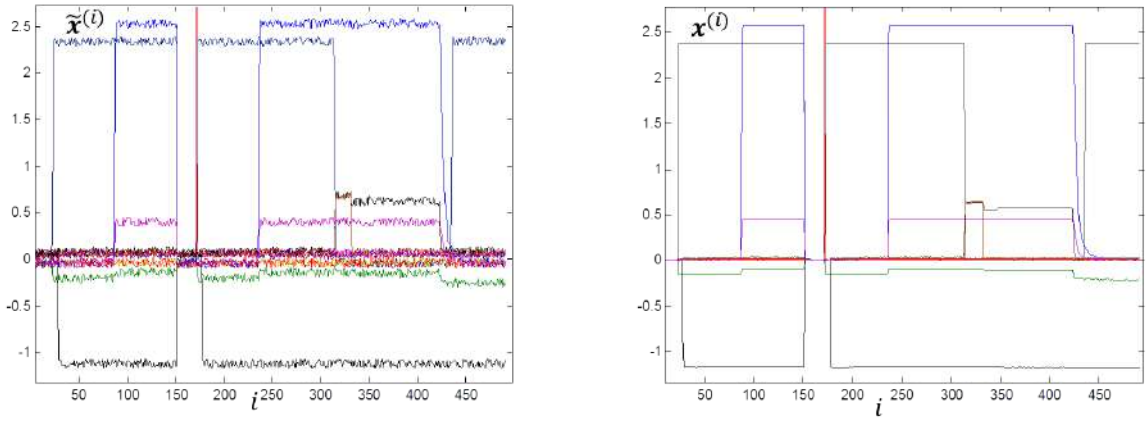
(a) (b)

Figura 5.13: Ilustración de (a) salida y de reconocimiento de patrones de la red 4 con $a = 90$, donde la red sólo confunde la muestra número 314 y (b) salida y de reconocimiento de patrones de la red 4 con $a = 1$.



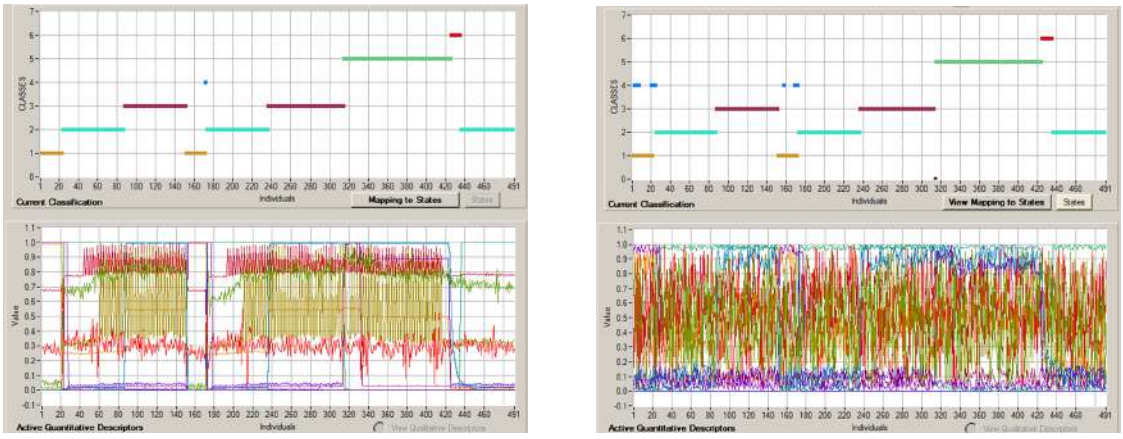
(a) (b)

Figura 5.14: (a) Evolución del entrenamiento de la red 13, dimensionada y entrenada mediante el Teorema 3. (b) Prueba la salida de reconocimiento de patrones y para dicha red (sin ruido). Cinco vectores de entrada, en el rango de muestras 5-l5 y en 20-25, que no se incluyeron en el conjunto de entrenamiento (de doce elementos), son claramente confundidos.



(a) (b)

Figura 5.15: Ilustración de (a) conjunto de entrada modificado por ruido para el Cajón II - PA/CMC ($a = 1$) y (b) conjunto de entrada original; ambos de 491 muestras.



(a) (b)

Figura 5.16: Ilustración de (a) resultado de clasificación de LAMDA para el Cajón II - PA/CMC ($\lambda = 0.60$) y (b) resultado del reconocimiento de patrones en un espacio de entrada que contiene exclusivamente a un conjunto modificado por ruido. El algoritmo confunde 10 muestras y desconoce una, haciendo un total de 11.

lleva al algoritmo a generar 7 clases que se observan en la Figura 5.16a. Esta vez, LAMDA confunde 10 muestras en la clase 4; que en realidad pertenecen a la clase 1, y desconoce[13] una, que debería pertenecer a la clase 3; asignándola a la clase NIC. Lo anterior, hace un total de 11 muestras que no se reconocen adecuadamente (2.24 %), obteniéndose un 97.75 %

[13]Ya se ha establecido en la Sección 4.2, que el hecho de que LAMDA asigne un objeto de entrada a la clase NIC, significa que existe una situación de ausencia de información que implica que dicho objeto puede pertenecer con el mismo grado a cualquiera de las clases existentes.

Red/ experimento	$MSE_{\text{mín}}$ (validación)	Arq. ($n_I/n_h/n_O$)
1	8.40×10^{-3}	[20/01/01] (pruning)
2	5.51×10^{-4}	[20/02/01] (pruning)
3	1.88×10^{-6}	[20/03/01] (pruning)
4	5.28×10^{-7}	[20/04/01] (pirámide)
5	1.99×10^{-5}	[20/05/01] (pruning)
6	2.41×10^{-7}	[20/06/01] (pruning)
7	7.70×10^{-6}	[20/07/01] (pruning)
8	4.80×10^{-4}	[20/08/01] (pruning)
9	3.40×10^{-3}	[20/09/01] (pruning)
10	3.64×10^{-9}	[20/10/01] (pruning
11	1.13×10^{-4}	[20/11/01] (pruning)
12	3.99×10^{-8}	[20/12/01] (pruning)

Tabla 5.5: Datos resultantes del entrenamiento de 12 redes neuronales con diferente enfoque de dimensionamiento, para el Cajón III PA/CMC. La información de la segunda columna se tomó del valor mínimo del MSE después de 15 (quince) inicializaciones aleatorias de los pesos sinápticos (y entrenamiento respectivo a partir de los mismos). El conjunto completo de entrada contiene 196 patrones (muestras), que se segmentan aleatoriamente en: Entrenamiento = 156 (80 %), Prueba = 19 (10 %) y Validación = 19 (10 %). Haciendo una inspección del conjunto de entrada, se identifican 5 clases que tendrían que reconocerse durante las pruebas 51, 52 y 53.

de reconocimiento.

5.4. Cajón III - PA/CMC

5.4.1. Redes neuronales artificiales

En la Tabla 5.5, se tienen los resultados experimentales que se obtienen de 12 redes neuronales que se entrenaron para reconocer patrones provenientes del Cajón III - PA/CMC. Once de ellas se han dimensionado por ‘‘*pruning*’’ generativo y una por la regla de la pirámide geométrica (red 4). Se ha determinado el descartamiento del Teorema 3, dada su ineficacia en el ámbito práctico que propone este trabajo de tesis. El Cajón que se trata en esta sección tiene 20 salidas, de modo que la regla de la pirámide geométrica calcula el número de neuronas en la capa oculta: $n_h = \sqrt{(20)(1)} = 4.47 \rightarrow 4$, con lo que se tiene la siguiente arquitectura: $n_I = 20, n_h = 4, n_O = 1$.

En la Figura 5.17, se muestran los patrones que se han usado para entrenar, probar y validar las redes de la Tabla 5.5; además se muestran los patrones que han sido utilizados

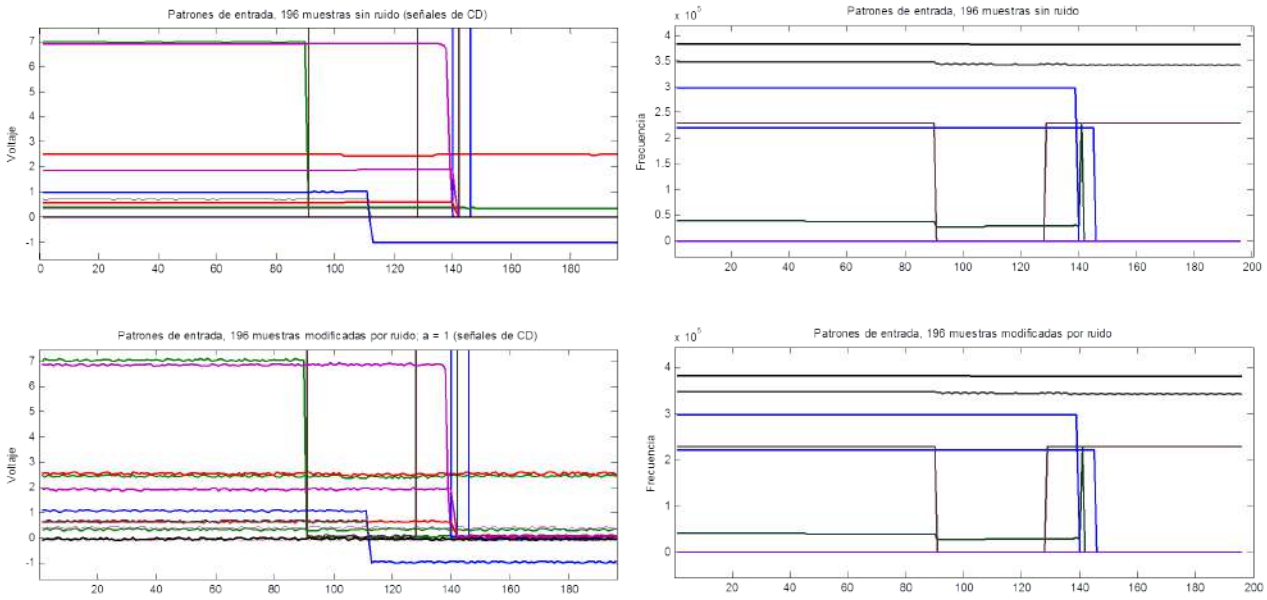


Figura 5.17: Patrones de entrada para entrenamiento, prueba y validación (gráficas de arriba) y patrones del conjunto modificado por ruido para validación de capacidad de generalización (gráficas de abajo) de las redes neuronales estudiadas en esta sección.

para validar la capacidad de generalización de cada red. Obsérvese en la misma tabla, un comportamiento del MSE similar al de las redes de la sección anterior. Nuevamente (al igual que en la sección que estudia al Cajón I) la regla de la pirámide geométrica resulta ser idónea, pues es posible observar en la Figura 5.18 cómo el MSE que se calcula para la red 4 tiende a ser simpre el mínimo a medida que la amplitud del ruido aumenta. Adicionalmente, obsérvese que para la red 9 se tiene un comportamiento de naturaleza recíproca.

En la Figura 5.19, se observan tanto la evolución del entrenamiento de la red elegida, como su desempeño al reconocer patrones contenidos en un conjunto de entrada modificado por ruido, cuya amplitud ha sido crítica hasta ahora tanto para LAMDA como para RNAs ($a = 1$), lo que ha proporcionado un método de experimentación muy útil. Como resultado para esta sección se tiene que dado el conjunto de entrada modificado por ruido de la Figura 5.17, entonces la red número 4 confunde o desconoce 114 muestras de las 196 que se le presentan, lo que hace un total de 58.16 %; es decir, se tiene un 41.84 % de reconocimiento.

5.4.2. Entrenamiento y validación de LAMDA (a través de SALSA)

A continuación se tienen los resultados obtenidos en cuanto al entrenamiento y validación de LAMDA. En la Figura 5.20a, se tiene el resultado que proporciona LAMDA después

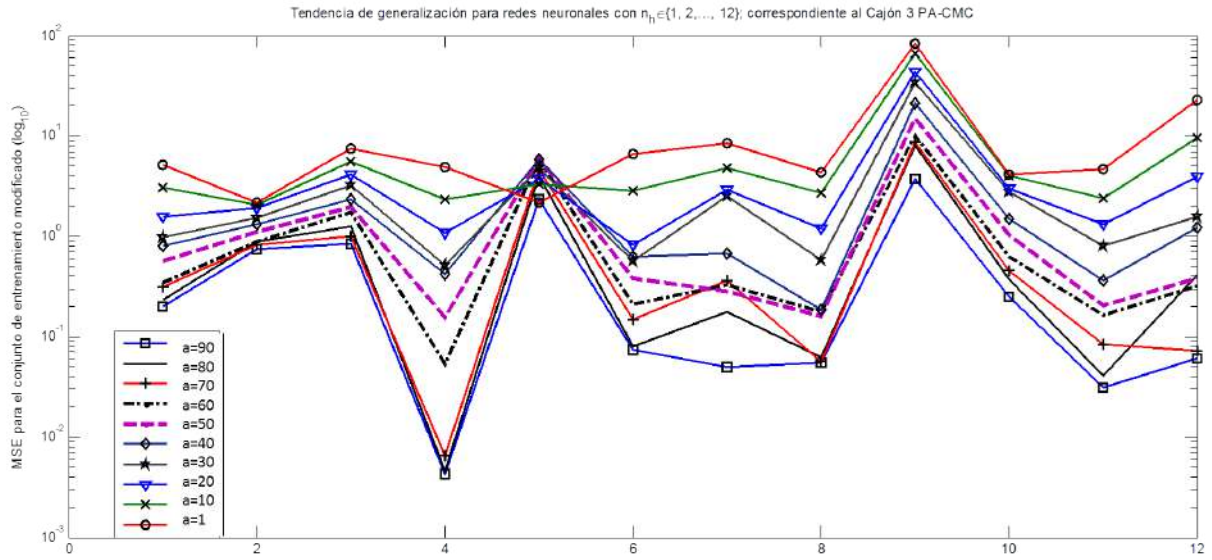


Figura 5.18: Curvas de tendencia de generalización.

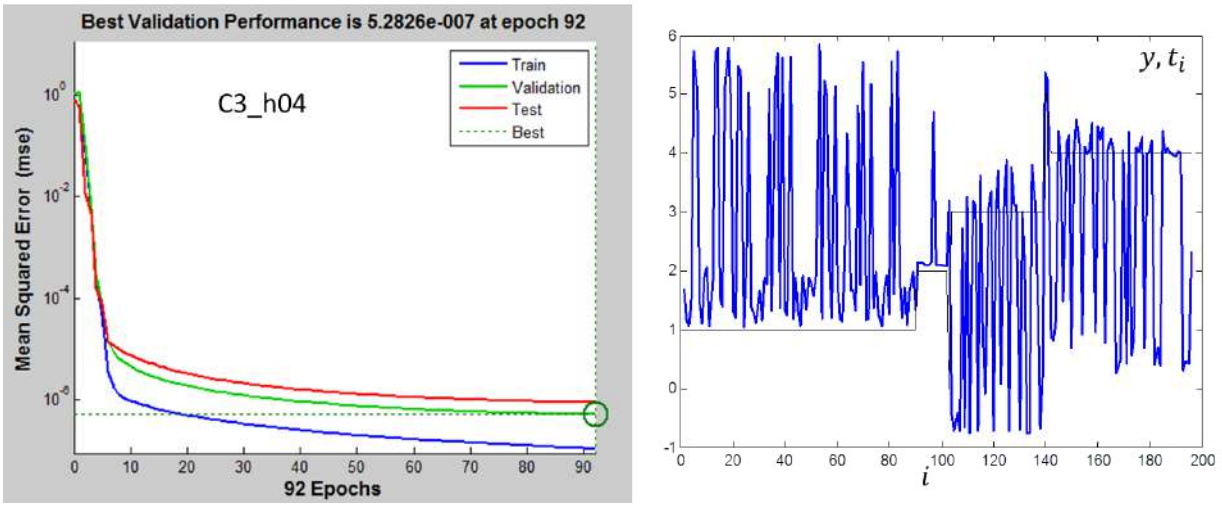
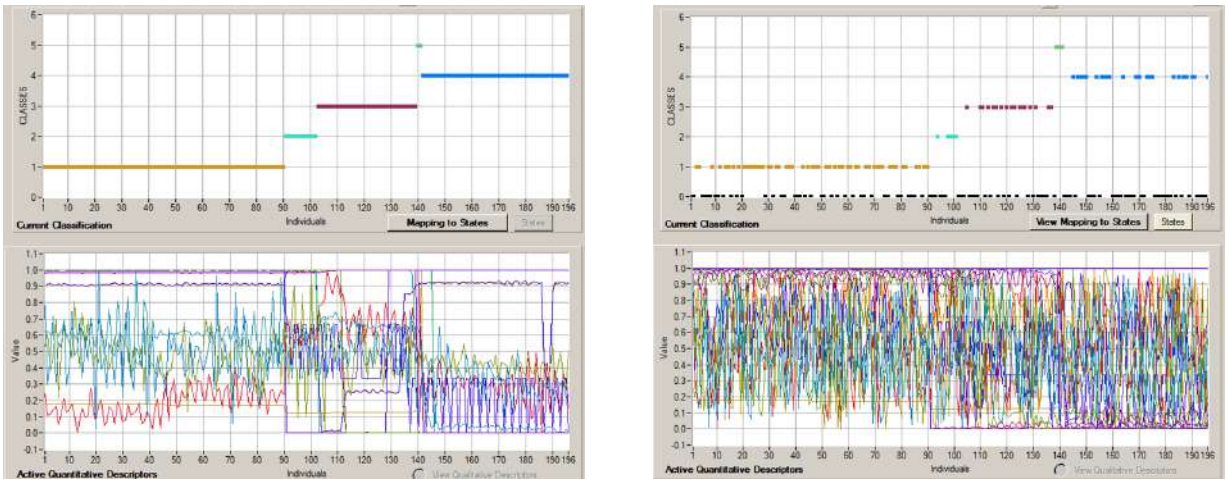


Figura 5.19: (a) Evolución del entrenamiento. (b) Reconocimiento de patrones de un conjunto modificado por ruido ($a = 1$) de la red número 4.



(a) (b)

Figura 5.20: Ilustración de (a) resultado de clasificación de LAMDA para el Cajón III - PA/CMC ($\lambda = 0.60$) y (b) resultado del reconocimiento de patrones en un espacio de entrada que contiene exclusivamente a un conjunto modificado por ruido. El algoritmo desconoce 96 muestras de las 196.

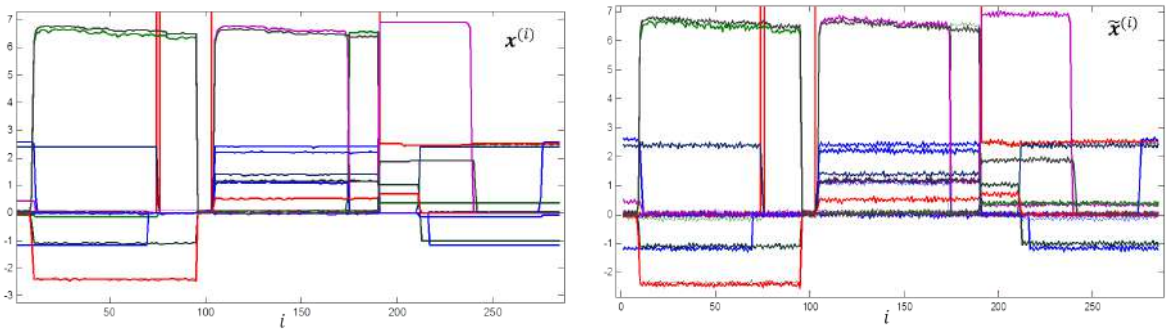
de realizar el proceso de entrenamiento y auto-organización. Se utilizado nuevamente un parámetro de exigencia $\lambda = 0.60$, lo que hasta ahora ha brindado buenos resultados[14], obteniéndose cinco clases; como en el caso de las RNAs. En lo que se refiere a la validación a través de un conjunto de entrada modificado por ruido, la Figura 5.20b muestra claramente cómo LAMDA tiene problemas para reconocer varias de las muestras ruidosas. De hecho no muestra confusión; sino más bien, indica que no cuenta con la información necesaria acerca de 96 de las 196 muestras que se le han presentado y por lo tanto, las asigna a la clase NIC. De esta manera, para el cajón en cuestión, LAMDA presenta un porcentaje de reconocimiento igual al 51.03 %, lo cual muestra que, hasta el momento, todavía supera a las RNAs.

5.5. Cajón IV - PA/CMC

5.5.1. Redes neuronales artificiales

Nuevamente se han entrenado 12 redes neuronales, ahora de 19 entradas (que se muestran en la Figura 5.21a), pues el Cajón IV PA/CMC cuenta con 19 salidas que el sistema de

[14]Dependerá de la necesidad de clasificación que se tenga para aplicaciones diferentes. Para el caso de este trabajo, la naturaleza de las señales (con demasiada variabilidad) que se analizan y, más que otra cosa, los cambios de estado en el comportamiento de los cajones no exigen que se generen muchas más clases de las que se tienen. Así que se estará experimentando con $\lambda = 0.6$, a menos que se requiera un valor diferente, mismo que será indicado explícitamente.



(a) (b)

Figura 5.21: Ilustración del (a) conjunto de entrada usado para entrenamiento, prueba y validación y (b) el conjunto modificado por ruido, basado en el original.

diagnóstico deberá registrar. El cálculo de la regla de la pirámide geométrica es: $n_h = \sqrt{(19)(1)} = 4.35 \rightarrow 4$, de manera que ahora los resultados del entrenamiento y arquitectura obtenidos, están en la fila número 4 de la Tabla 5.6.

En esta ocasión, la regla de la pirámide no ha resultado efectiva, puesto que la capacidad de generalización de la red correspondiente a este enfoque de dimensionamiento, resultó superada por la red número 9. Se puede observar en la Figura 5.21b, el conjunto de entrada modificado por ruido ($a = 1$) con el cual se ha validado la capacidad de generalización de las redes entrenadas en esta sección. Se ha concluido (según la Figura 5.22), que la red cuyo número de neuronas en la capa oculta es igual a 9 ($n_h = 9$) es la que mejor capacidad de generalización puede aportar, tendiendo a ser en la mayoría de los casos la que menor MSE presenta, a medida que se varía la amplitud del ruido.

Véanse en la Figura 5.23, tanto la manera en que evoluciona su entrenamiento y cómo se desempeña al reconocer los patrones del conjunto modificado con mayor intensidad de ruido ($a = 1$). Como resultados de las pruebas, se tiene que la red número 9 confunde 8 muestras de las 285, lo que representa un 3.15 % (96.85 % de reconocimiento).

5.5.2. Entrenamiento y validación de LAMDA (a través de SALSA)

Se han hecho experimentos con LAMDA, ahora para el Cajón IV se tiene el resultado de clasificación que se muestra en la Figura 5.24a. Se han detectado 9 clases, adicionalmente para este experimento se ha usado un parámetro de exigencia $\lambda = 0.55$, puesto que resulta en una clasificación más adecuada.

Red/ experimento	$MSE_{\text{mín}}$ (validación)	Arq. $(n_I/n_h/n_O)$
1	5.70×10^{-3}	[19/01/01] (pruning)
2	2.10×10^{-3}	[19/02/01] (pruning)
3	3.78×10^{-4}	[19/03/01] (pruning)
4	2.29×10^{-4}	[19/04/01] (pirámide)
5	3.02×10^{-5}	[19/05/01] (pruning)
6	8.42×10^{-5}	[19/06/01] (pruning)
7	1.52×10^{-5}	[19/07/01] (pruning)
8	3.91×10^{-5}	[19/08/01] (pruning)
9	1.89×10^{-7}	[19/09/01] (pruning)
10	1.20×10^{-6}	[19/10/01] (pruning
11	1.13×10^{-4}	[19/11/01] (pruning)
12	2.50×10^{-5}	[19/12/01] (pruning)

Tabla 5.6: Datos resultantes del entrenamiento de 12 redes neuronales con diferente enfoque de dimensionamiento, para el Cajón IV PA/CMC. La información de la segunda columna se tomó del valor mínimo del MSE después de 15 (quince) inicializaciones aleatorias de los pesos sinápticos (y entrenamiento respectivo a partir de los mismos). El conjunto completo de entrada contiene 285 patrones (muestras), que se segmentan aleatoriamente en: Entrenamiento = 229 (80 %), Prueba = 28 (10 %) y Validación = 28 (10 %). Haciendo una inspección del conjunto de entrada, se identifican 9 clases que tendrían que reconocerse durante las pruebas 63, 64 y 65.

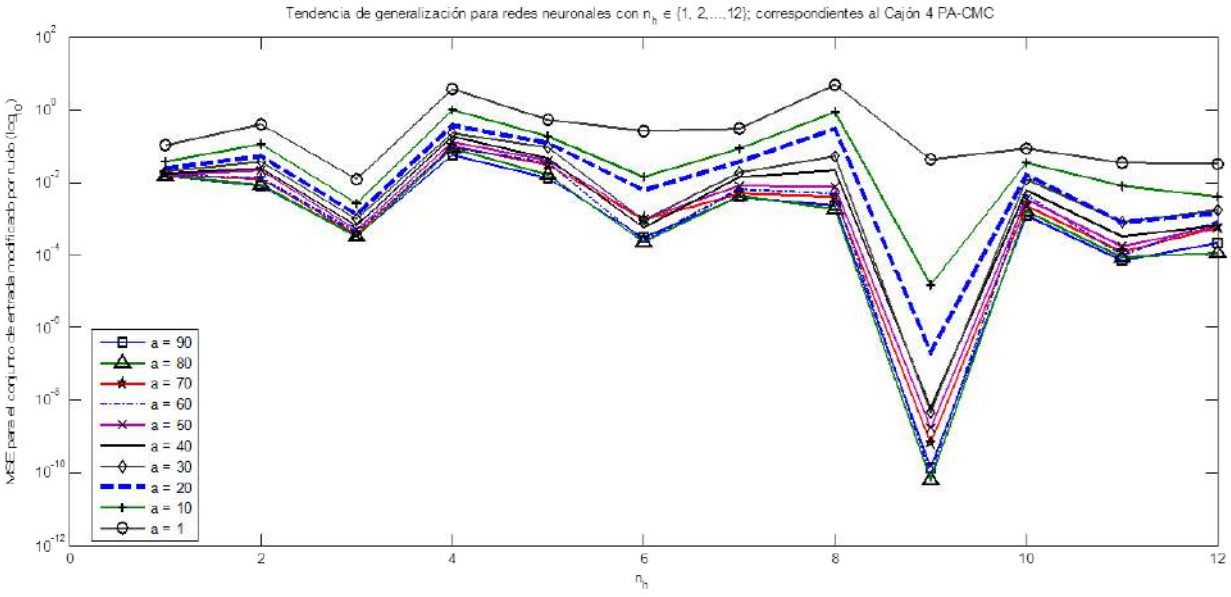


Figura 5.22: Curvas de tendencia de generalización.

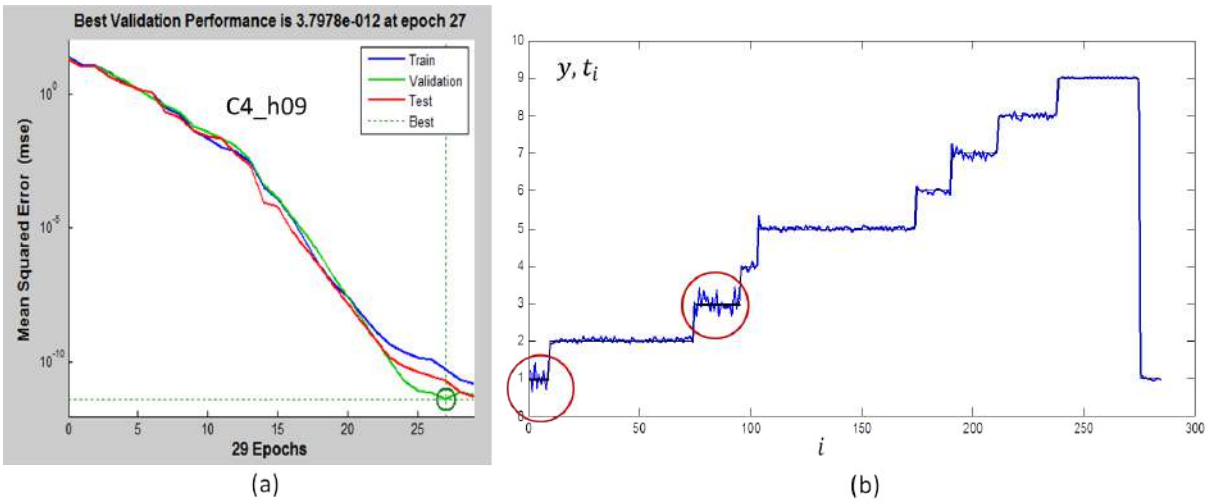


Figura 5.23: (a) Evolución del entrenamiento y (b) desempeño en el reconocimiento de patrones de la red 9. El algoritmo desconoce 8 muestras de las 285 (observe con mayor atención los patrones 1 y 3).

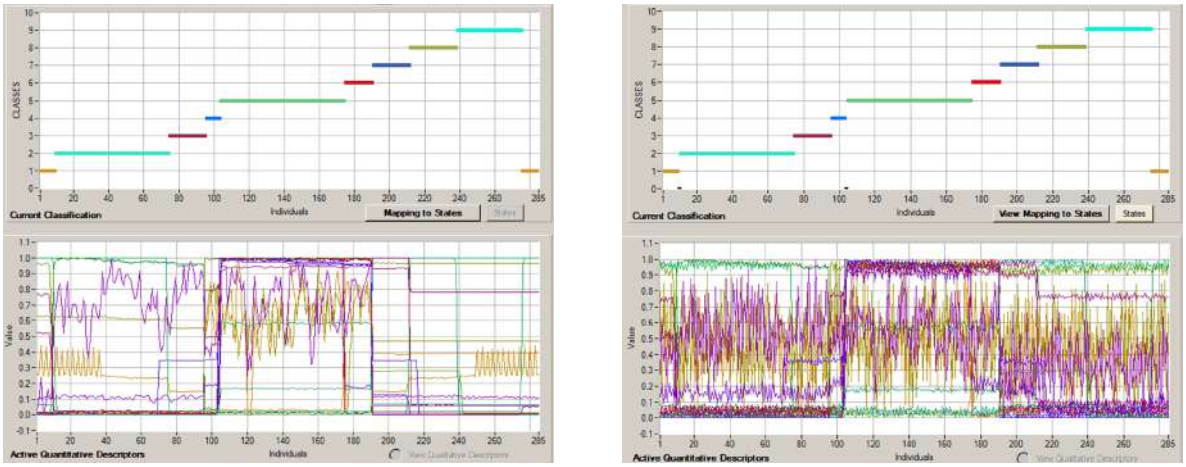
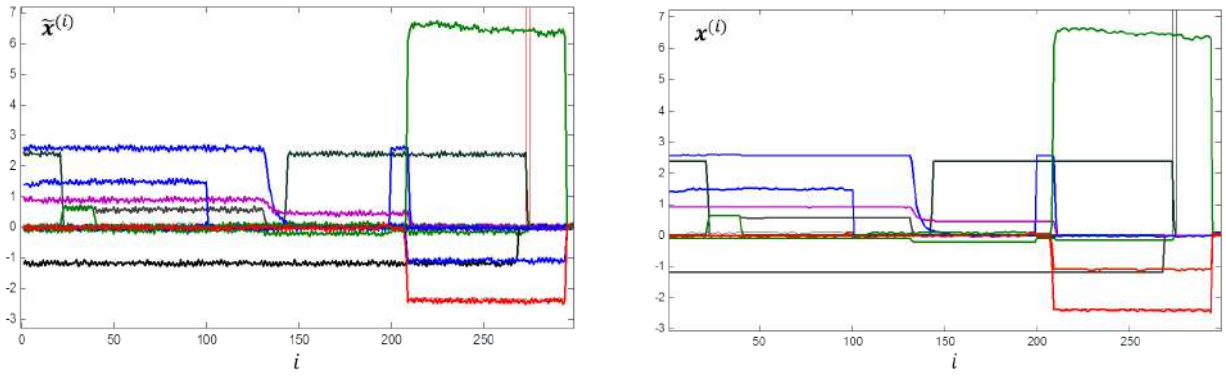


Figura 5.24: (a) Resultado de clasificación y (b) desempeño de reconocimiento de patrones de LAMDA.

LAMDA ha sido entrenado con el mismo conjunto de entrenamiento que las redes neuronales, también ha sido validado con el mismo conjunto de entrada modificado por ruido. Se tiene de dicha validación, que LAMDA desconoce únicamente dos muestras (0.07 %), la 10 y la 104, lo que da como resultado un 99.93 % de reconocimiento. En la Figura 5.24b se muestra gráficamente el resultado anterior.



(a) (b)

Figura 5.25: Ilustración del (a) conjunto de entrada usado para entrenamiento, prueba y validación y (b) el conjunto modificado por ruido, basado en el original.

5.6. Cajón I - CML/CMR

5.6.1. Redes neuronales artificiales

El Cajón I CML/CMR tiene 17 salidas que el sistema de diagnóstico desarrollado en este trabajo deberá registrar, de manera que se han entrenado 12 redes neuronales con el fin de elegir la que mejor se desempeñe en cuanto a capacidad de generalización. El resultado de dicho entrenamiento por cada red se encuentra en la Tabla 5.7. En la segunda columna de dicha tabla (tal como en experimentos anteriores), se ha registrado el mejor valor del MSE obtenido para cada arquitectura, después de quince inicializaciones aleatorias de los pesos sinápticos. Nuevamente dichos valores no brindan una base concluyente sobre el desempeño de cada red, de forma que se recurre a crear conjuntos de entrada modificados por ruido (Figura 5.25a) de distinta intensidad, basados en el conjunto original de la Figura 5.25b.

Tal como sucedió para el cajón anterior, la red número 9 resulta con mayor capacidad de generalización, lo que se puede observar claramente en la Figura 5.26, donde dicha red se elige como la mejor para el cajón que se estudia en esta sección por ser la que tiende a presentar el menor MSE; independientemente de la intensidad del ruido que presente el conjunto de entrada.

La red neuronal elegida, confunde un total de 33 muestras; es decir, el 11.03 % de un total de 299 muestras. Lo anterior implica que dicha red (red 9) brida un 88.97 % de reconocimiento. En la Figura 5.27 se pueden apreciar tanto la forma en que evolucionó su entrenamiento como la forma en que aproxima la función que define el vector de objetivos (ejemplos).

Red/ experimento	$MSE_{\text{mín}}$ (validación)	Arq. $(n_I/n_h/n_O)$
1	2.39×10^{-4}	[17/01/01] (pruning)
2	1.04×10^{-4}	[17/02/01] (pruning)
3	1.90×10^{-4}	[17/03/01] (pruning)
4	6.23×10^{-5}	[17/04/01] (pirámide)
5	1.24×10^{-5}	[17/05/01] (pruning)
6	6.77×10^{-6}	[17/06/01] (pruning)
7	7.83×10^{-5}	[17/07/01] (pruning)
8	9.19×10^{-5}	[17/08/01] (pruning)
9	1.30×10^{-5}	[17/09/01] (pruning)
10	8.66×10^{-6}	[17/10/01] (pruning
11	2.97×10^{-4}	[17/11/01] (pruning)
12	1.57×10^{-8}	[17/12/01] (pruning)

Tabla 5.7: Datos resultantes del entrenamiento de 12 redes neuronales con diferente enfoque de dimensionamiento, para el Cajón I CML/CMR. La información de la segunda columna se tomó del valor mínimo del MSE después de 15 (quince) inicializaciones aleatorias de los pesos sinápticos (y entrenamiento respectivo a partir de los mismos). El conjunto completo de entrada contiene 299 patrones (muestras), que se segmentan aleatoriamente en: Entrenamiento = 239 (80 %), Prueba = 30 (10 %) y Validación = 30 (10 %). Haciendo una inspección del conjunto de entrada, se identifican 11 clases que tendrían que reconocerse durante las pruebas 93, 94 y 95.

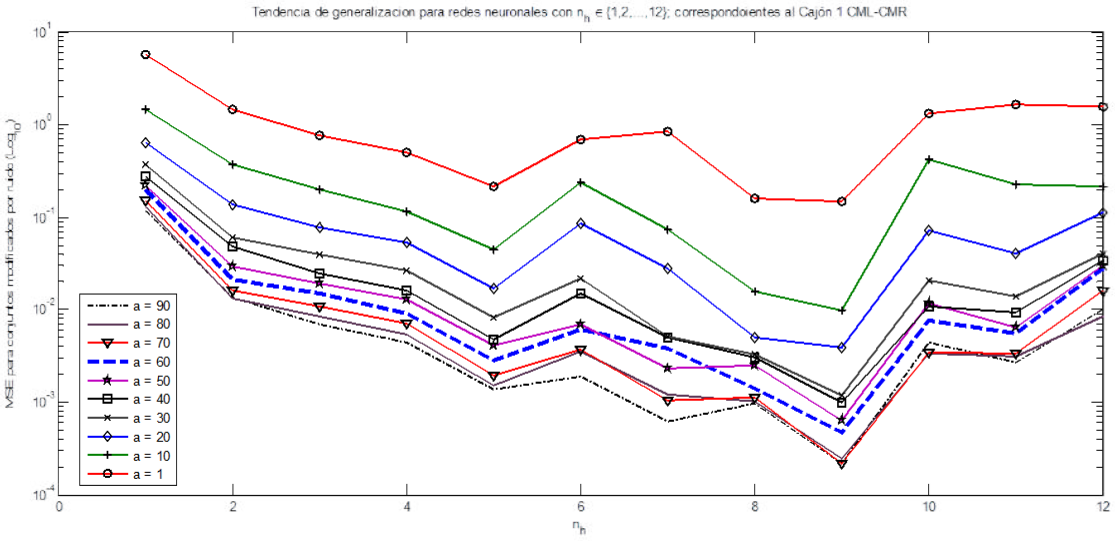


Figura 5.26: Curvas de tendencia de generalización.

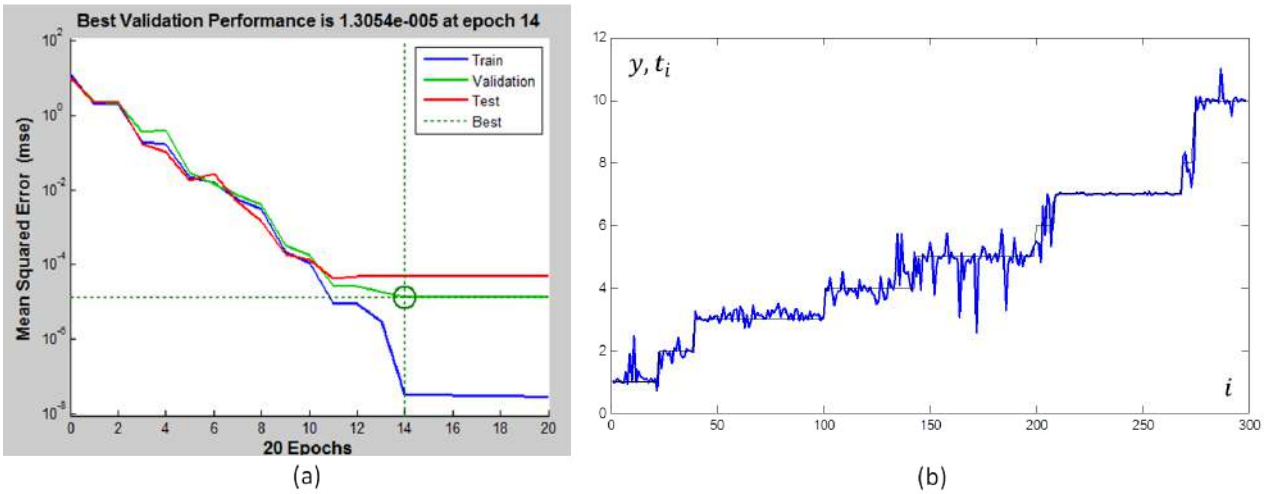


Figura 5.27: (a) Evolución del entrenamiento y (b) desempeño en el reconocimiento de patrones de la red 9. El algoritmo confunde 33 muestras de las 299, con $a = 1$.

5.6.2. Entrenamiento y validación de LAMDA (a través de SALSA)

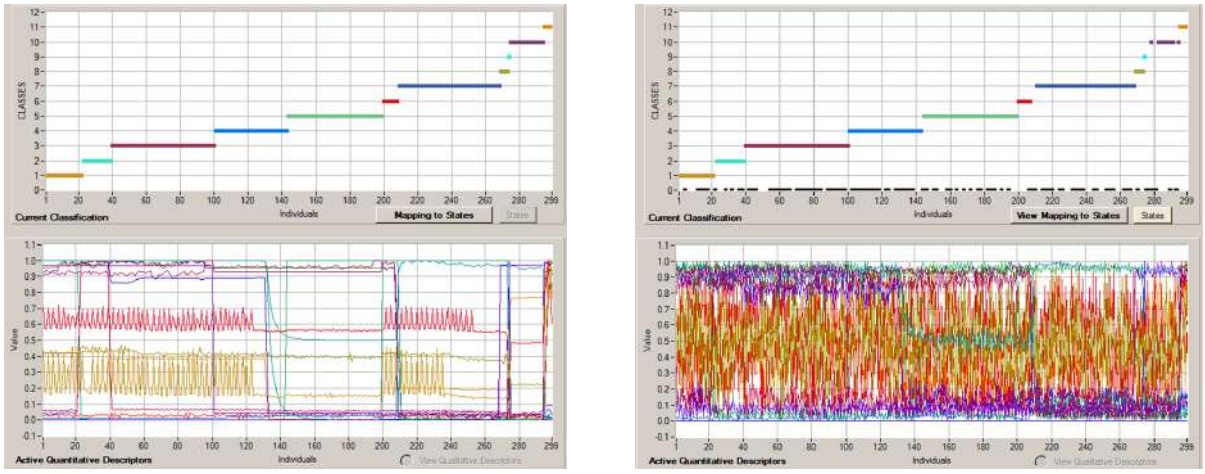


Figura 5.28: Ilustración del (a) resultado de clasificación y (b) desempeño de reconocimiento de patrones de LAMDA.

También se ha usado el conjunto de entrada de la Figura 5.25b para generar la partición difusa que se muestra en la Figura 5.28a, con un parámetro de exigencia igual a $\lambda = 0.56$. Esta vez, el desempeño de LAMDA es rebasado por el de la red neuronal correspondiente. Se muestra en la Figura 5.28b cómo el algoritmo equivoca gran parte de la clasificación; de hecho, desconoce 99 muestras ruidosas de las 299 que se le presentan en el espacio de entrada

con el fin de validar, lo cual indica que para el cajón bajo estudio se tiene un 66.89 % de reconocimiento.

5.7. Cajón II - CML/CMR

Se tiene ahora el último cajón del sistema de Piloto Automático. Se han obtenido resultados similares a los de secciones anteriores (excepto para el Cajón I CML/CMR, Sección 5.6), en el sentido de que la regla de la pirámide geométrica dicta un número de neuronas en capa oculta, igual o cercano al que brinda la mejor capacidad de generalización. La pirámide geométrica indica que $n_h = 3.87 \rightarrow 4$ para el cajón que se estudia en esta sección; sin embargo, la red con $n_h = 3$ es la que se desempeñó mejor.

5.7.1. Redes neuronales artificiales

El Cajón II - CML/CMR cuenta con quince salidas que el sistema de diagnóstico deberá registrar, de modo que las redes neuronales que se han probado en esta sección cuentan con quince nodos de entrada. En la Tabla 5.8, se tienen las mediciones realizadas al término del entrenamiento de las 12 redes con las que se realizaron experimentos para el cajón en cuestión. En este último conjunto de experimentos también se reafirma que no es una medida de capacidad de generalización el MSE del conjunto de validación.

A lo largo de este capítulo, se ha hechado mano de las curvas de tendencia para justificar cuantitativamente la preferencia sobre una red neuronal en particular, ya que se ha notado, en todos los experimentos sin excepción, que el MSE obtenido para una red neuronal después de procesar los conjuntos de entrenamiento, prueba y validación (sobre todo este último como lo establece la literatura) no es por sí sólo un valor concluyente sobre la capacidad de generalización de una red neuronal y su arquitectura. Con el fin de solventar la ambigüedad que ésto plantea, se ha usado un conjunto de patrones de entrada que han sido modificados mediante la adición de una función de ruido con intensidad variable ($a \in \{90, 80, ..., 10, 1\}$), lo cual lleva a situaciones críticas el desempeño de los dos algoritmos y permite verificar cuál de los dos es más robusto[15] y con mayor plasticidad. En la Figura 5.29, se muestran los patrones de entrada que representan en su mayoría los valores de frecuencia de las salidas del cajón.

Por lo tanto, en la Figura 5.30 se muestran las curvas de tendencia de generalización que

[15]No se tiene directamente una cuantificación sobre la robustez de RNAs o LAMDA; sin embargo, los resultados de algunos experimentos de este capítulo (secciones 5.4 y 5.7) sugieren que las condiciones que en particular se impusieron, lograron llevar al límite la robustez de ambos algoritmos.

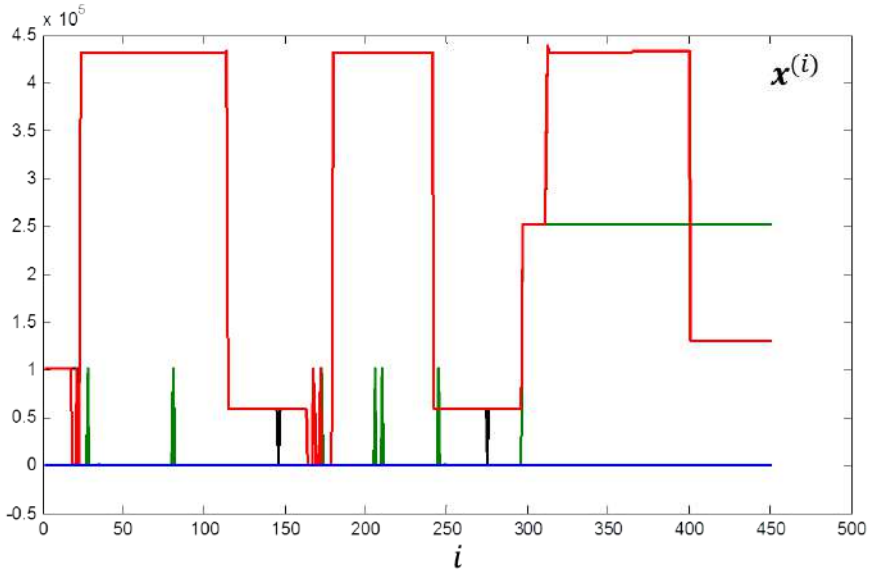


Figura 5.29: Patrones de entrada para el Cajón II CML/CMR. No se muestra esta vez el conjunto modificado por ruido, puesto que la mayoría de las señales que se estudian son de alta frecuencia y, dada la escala, las modificaciones son imperceptibles gráficamente.

Red/ experimento	$MSE_{\text{mín}}$ (validación)	Arq. $(n_I/n_h/n_O)$
1	6.50×10^{-3}	[15/01/01] (pruning)
2	7.75×10^{-4}	[15/02/01] (pruning)
3	6.30×10^{-11}	[15/03/01] (pruning)
4	4.33×10^{-7}	[15/04/01] (pirámide)
5	3.46×10^{-9}	[15/05/01] (pruning)
6	2.91×10^{-12}	[15/06/01] (pruning)
7	2.29×10^{-9}	[15/07/01] (pruning)
8	7.66×10^{-6}	[15/08/01] (pruning)
9	8.58×10^{-5}	[15/09/01] (pruning)
10	3.22×10^{-11}	[15/10/01] (pruning
11	1.74×10^{-7}	[15/11/01] (pruning)
12	5.86×10^{-7}	[15/12/01] (pruning)

Tabla 5.8: Datos resultantes del entrenamiento de 12 redes neuronales con diferente enfoque de dimensionamiento, para el Cajón II CML/CMR. La información de la segunda columna se tomó del valor mínimo del MSE después de 15 (quince) inicializaciones aleatorias de los pesos sinápticos (y entrenamiento respectivo a partir de los mismos). El conjunto completo de entrada contiene 451 patrones (muestras), que se segmentan aleatoriamente en: Entrenamiento = 361 (80 %), Prueba = 45 (10 %) y Validación = 45 (10 %). Haciendo una inspección del conjunto de entrada, se identifican 6 clases que tendrían que reconocerse durante las pruebas 79, 80 y 81.

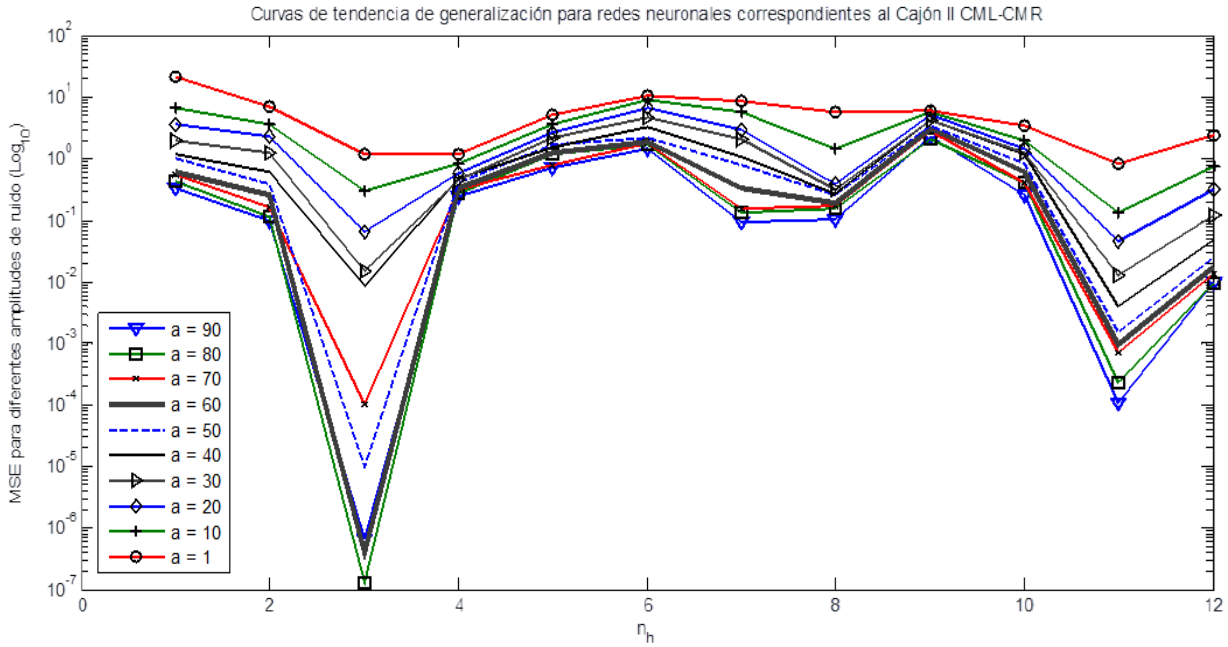


Figura 5.30: Curvas de tendencia de generalización.

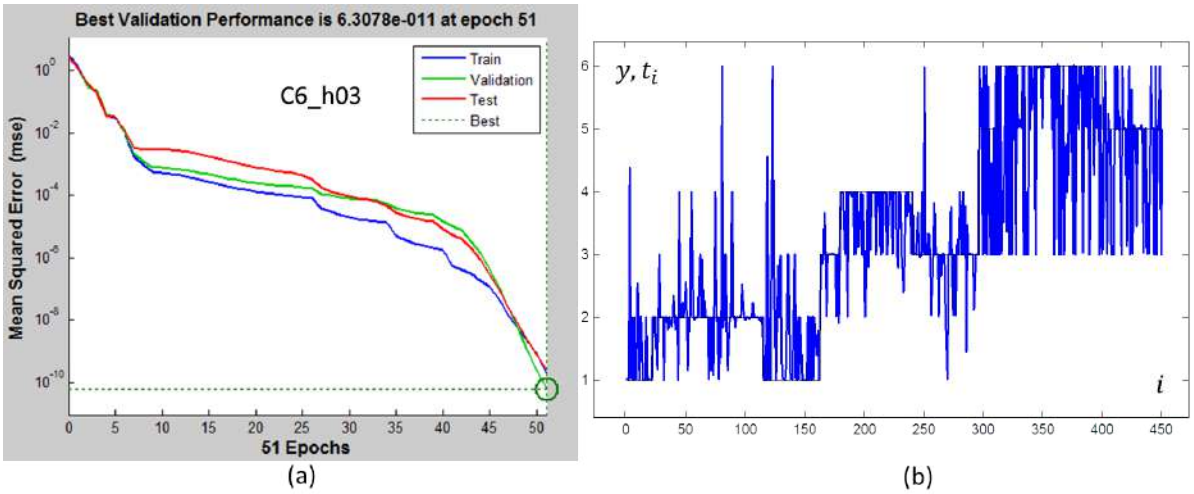
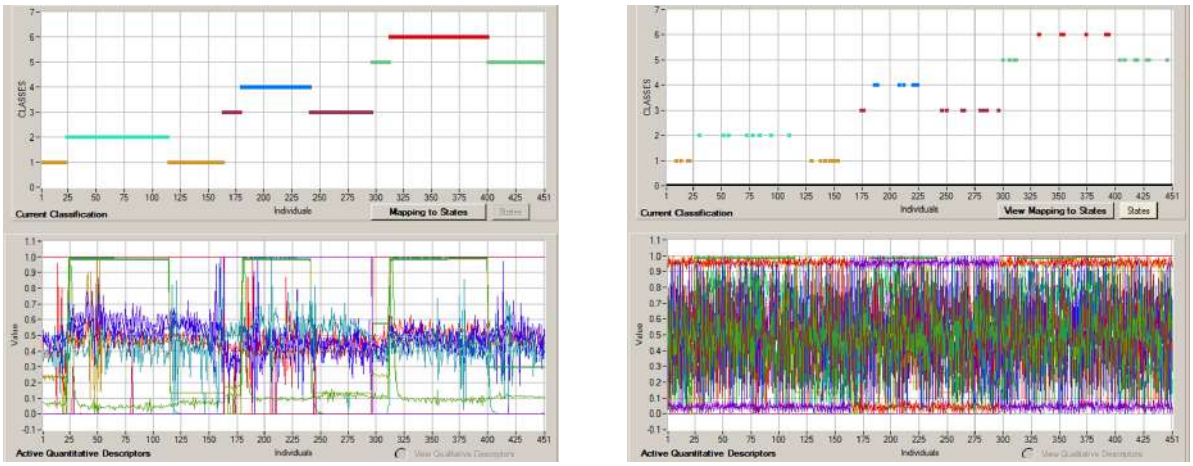


Figura 5.31: (a) Evolución del entrenamiento y (b) desempeño en el reconocimiento de patrones de la red 3. El algoritmo confunde 179 muestras de las 451, con $a = 1$.

revelan que la red número 3 tiende a cometer menos errores a medida que los patrones de entrada se vuelven más confusos. Adicionalmente, en la Figura 5.31a se muestra cómo varía el MSE conforme cambian los pesos sinápticos de la red durante el entrenamiento y en la Figura 5.31b se ve cómo dicha red aproxima la función definida por el vector de objetivos. De hecho confunde un total de 179 muestras (39.68 %) de las 451, lo que indica un 60.32 % de reconocimiento.

5.7.2. Entrenamiento y validación de LAMDA (a través de SALSA)



(a) (b)

Figura 5.32: Ilustración del (a) resultado de clasificación y (b) desempeño de reconocimiento de patrones de LAMDA.

Se tiene también el último experimento con LAMDA, en el cual se ha llevado a cabo un proceso de análisis de clusters dado un conjunto de datos igual que el empleado para redes neuronales. El resultado de dicho análisis se muestra en la Figura 5.32a, usando un parámetro de exigencia igual a 0.6, lo que ofrece una partición difusa adecuada. Por segunda ocasión (como en sección anterior), LAMDA resulta con un desempeño menor en fase de reconocimiento de patrones (Figura 5.32b), desconociendo el 69.62 % de las muestras.

5.8. Complejidad computacional

La complejidad computacional, es una medida usada para saber qué tan eficiente es un algoritmo, independientemente de la máquina en la que sea ejecutado, en términos tanto

de espacio en memoria (complejidad espacial) como de tiempo de ejecución (complejidad temporal). Desde un punto de vista generalizado, el último se basa en un análisis asintótico[16] del número de operaciones n, dependiendo de la cantidad de datos de entrada, que lleva a cabo un algoritmo, lo que repercute directamente en el tiempo que éste consume. Cabe señalar que en este trabajo se tiene un enfoque particularmente aplicativo, por lo que se comparan tiempos específicos de ejecución de los algoritmos que se estudian, de manera que se sepa cuál de ellos es más rápido. El punto de comparación que se plantea entonces, se basa en el número de operaciones básicas[17] que cada uno de ellos ejecuta para resolver el mismo problema[84].

En este trabajo de tesis, se verifica la complejidad temporal tanto de LAMDA como del perceptrón multicapa elegido para cada cajón del PA en fase de reconocimiento de patrones (no de entrenamiento), con la finalidad de tomarla en consideración como criterio de comparación entre estos dos algoritmos, cuyos enfoques a partir de los cuales se suscitan son muy distintos.

Para medir la complejidad computacional temporal es necesario plantear un esquema general de cada algoritmo, a efecto de etiquetar cada instrucción con el número de operaciones básicas que ejecuta, de modo que éstas sean contabilizadas.

5.8.1. Redes neuronales Artificiales

Se tienen arquitecturas diferentes de redes neuronales para cada cajón estudiado en las secciones anteriores de este capítulo; sin embargo, la complejidad temporal de cada una puede ser obtenida en términos del número de entradas que cada neurona tiene.

En el Algoritmo 1 se tiene la cantidad de operaciones que ejecuta cada instrucción, en términos tanto del número de nodos de entrenada como del número de neuronas en la capa oculta (n_h), de manera que se tiene la suma correspondiente:

$$f(n_I) = n_h + 1 + 3n_h + 1 + n_h(3n_h + 1) + n_h(7n_I) + 4n_h + 7n_h + 5n_h + 2 + 1$$

$$f(n_I) = 3n_h + 10n_h n_I + 5; \tag{5.2}$$

donde: n_I es el número de nodos de entrada y a su vez la dimensión m del espacio de entrada $X \supset \{\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_n\}$. Por lo tanto, sustituyendo n_h para cada arquitectura se tiene en la

[16]Se realiza un procedimiento que busca obtener una función que acote por arriba el tiempo de ejecución de un algoritmo, dado un cojunto de datos de entrada.

[17]Por operación básica se entiende toda aquella operación elemental del lenguaje en el que se esté programando (C y MATLAB), suponiendo que sus tiempos de ejecución son iguales. Se tienen las siguientes: suma (+), Resta (-), multiplicación (×), división (/), módulo (mod), potenciación (^), asignación (=), todos los operadores relacionales ($<, >, ==, \neq, \leq, \geq$), accesos a memoria y saltos.

Algoritmo 1 Esquema general del algoritmo de una red neuronal de dos capas (fase de reconocimiento de patrones)

```
1: procedure RedNeuronal(x[], wI[][], nI, nh, bh[], bO, wh[])
2:     int h, j, i; float yh[nh]={0}, mac = 0;                    ▷ n_h + 1
3:     for (h=0; h≤nh; h++) do                                    ▷ 3n_h + 1
4:         for (j=0; j≤nI; j++) do                                ▷ n_h(3n_I + 1)
5:             yh[h] = x[j]*wI[h][j] + yh[h];                     ▷ n_h(7n_I)
6:         end for
7:         yh[h] = yh[h] + bh[h];                                 ▷ 4n_h
8:         yh[h] = 1 / (1 + exp(-yh[h]));                         ▷ 7n_h
9:         mac = yh[h]*wh[h]+mac;                                 ▷ 5n_h
10:    end for
11:    mac = mac + bO;                                            ▷ 2
12:    return mac;                                                ▷ 1
13: end procedure
```

Cajón	Arq. $(n_I/n_h/n_O)$	Complejidad temp.
Cajón I PA/CMC	28/05/01	$f(n_I) = 50n_I + 20$
Cajón II PA/CMC	12/04/01	$f(n_I) = 40n_I + 17$
Cajón III PA/CMC	20/04/01	$f(n_I) = 40n_I + 17$
Cajón IV PA/CMC	19/09/01	$f(n_I) = 90n_I + 32$
Cajón I CML/CMR	17/09/01	$f(n_I) = 90n_I + 32$
Cajón II CML/CMR	15/03/01	$f(n_I) = 30n_I + 14$

Tabla 5.9: Complejidad computacional para cada red neurononal.

columna número tres de la Tabla 5.9 la complejidad temporal respectiva a partir de la función general de la Ecuación (5.2).

5.8.2. LAMDA

Se tiene para LAMDA el Algoritmo 2, donde, al igual que para RNAs, se etiqueta cada instrucción con el número de operaciones básicas que ejecuta. Separadamente se ha calculado la complejidad temporal para las funciones[18] `max ( )` y `min ( )`. Obsérvese el Algoritmo 2, de donde se tiene la complejidad temporal de LAMDA:

$$f(m) = N_{clases}m + N_{clases} + 3N_{clases} + 1 + N_{clases}(3m+1) + N_{clases}(14m) + 3N_{clases} + 1$$

$$+N_{clases}(8m+4) + N_{clases}(8m+4) + 6N_{clases} + (8N_{clases} + 4) + 1$$

[18]Se ha calculado como $8N + 4$; siendo N el tamaño del arreglo del que se busca encontrar el máximo o el mínimo.

$$= m(N_{clases} + 3N_{clases} + 14N_{clases}8N_{clases}) + 32N_{clases} + 7$$

$$f(m) = m(26N_{clases}) + 32N_{clases} + 7; \tag{5.3}$$

donde: m es la dimensión del espacio de entrada $X \supset \{\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_n\}$ (que cambia según el cajón del PA que se analice, dado su número de salidas), N_{clases} es el número de clases que el algoritmo haya aprendido durante el entrenamiento. En la línea número 15 del Algoritmo 2, lo que se regresa como valor es el índice del valor máximo de `GADs[]`.

Algoritmo 2 Esquema general del algoritmo LAMDA (fase de reconocimiento de patrones)

```
1:  procedure FUNCLAMDA(lambda, Nclases, nI, clases[][], x[])
2:      int i, j;
3:      float MADs[Nclases][nI]={0}, GADs[Nclases] = {0};      ▷ N_clases n_I + N_clases
4:      for (i = 0; i ≤ Nclases; i++) do                              ▷ 3N_clases + 1
5:          for (j = 0; j ≤ nI; j++) do                          ▷ N_clases(3n_I + 1)
6:              MADs[i][j]=(clases[i][j] ^ x[j])*((1-clases[i][j]) ^ (1-x[j]));
7:                                                               ▷ N_clases(14n_I)
8:          end for
9:      end for
10:     for (i = 0; i ≤ Nclases; i++) do                              ▷ 3N_clases + 1
11:         maximo = max(MADs[i], nI);                           ▷ N_clases(8n_I + 4)
12:         minimo = min(MADs[i], nI);                           ▷ N_clases(8n_I + 4)
13:         GADs[i] = lambda * minimo + (1-lambda)*maximo;                ▷ 6N_clases
14:     end for
15:     return max(GADs, Nclases);                               ▷ (8N_clases + 4) + 1
16: end procedure
```

5.9. Resultados de la comparación entre los algoritmos

En esta sección se tienen de manera condensada los datos obtenidos a lo largo del presente capítulo, de manera que se tengan disponibles como información precisa sobre el desempeño de las RNAs y LAMDA, con la finalidad de decidir cuál de los dos es idóneo para su implementación el Simulador de Pilotaje Automático de Taller del STC Metro de la Ciudad de México. Como criterios de comparación se han fijado el porcentaje de reconocimiento y la complejidad computacional temporal. El primero se ha elegido a efecto de conocer qué algoritmo es más robusto ante condiciones adversas que se puedan suscitar debido a factores externos (ruido principalmente) al sistema. Por otra parte, el segundo se origina de la necesidad de tener un segundo punto de comparación que pueda proporcionar información sobre la eficiencia de cada algoritmo, ya que éstos procesan grandes cantidades de datos,

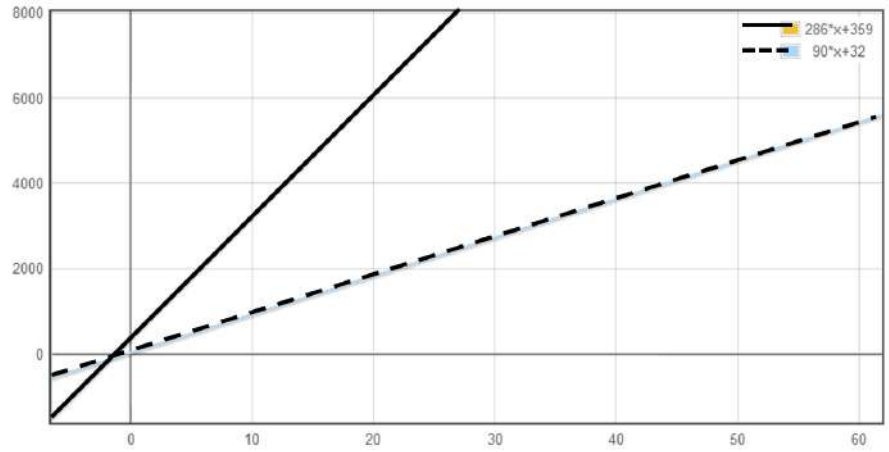


Figura 5.33: Funciones temporales máximas de LAMDA (línea continua) y RNAs (línea punteada). El eje horizontal representa la dimensión del espacio de entrada y el vertical el número de operaciones básicas.

lo cual puede resultar en notables diferencias en tiempo de procesamiento. El número de muestras (objetos) de entrada al sistema mediante la tarjeta de adquisición de datos, crece en proporciones de millones cada minuto.

Una diferencia importante entre los dos algoritmos bajo estudio, es que, si bien la complejidad computacional de ambos (RNAs entrenadas y LAMDA) crece en proporción lineal a la cantidad de objetos que reciban como entrada al avanzar el tiempo, la complejidad computacional de LAMDA crece además con la cantidad de clases que aprende o que haya aprendido (Véase la Ecuación (5.3)). A efecto de mostrar información lo más objetiva posible, se tomará en consideración el número de clases que se hayan reconocido para cada cajón, dado el número de pruebas que se analizaron. Así entonces, se tiene la complejidad computacional de LAMDA para cada cajón del PA en la Tabla 5.10. Nótese según los experimentos en esta sección, que así como la complejidad computacional de LAMDA puede variar según el valor que se escoja para λ y, por el ende, según el número de clases que haya aprendido el algoritmo en fase de entrenamiento, así también podría variar la complejidad de las RNAs si se entrenan con un conjunto de datos diferente; aunque no en las mismas proporciones. Lo último es claro si se compara la proporción en que varía la cota $f(m)$ desde $N_{clases} = 6$ a $N_{clases} = 11$ (para LAMDA, Tabla 5.10) con la proporción en que varía $f(n_I)$ desde $n_h = 3$ a $n_h = 9$ (para RNAs, Tabla 5.9). Para tener una idea más general de la diferencia en complejidad computacional entre los dos algoritmos, se muestran en la Figura 5.33 las funciones temporales máximas de cada algoritmo. El número de operaciones básicas que ejecuta LAMDA crece en una proporción de 2:1 con respecto de las que ejecutan las RNAs, a medida que objetos de entrada crecen en dimensión.

Ya se han obtenido los porcentajes de reconocimiento de patrones para los algoritmos estudiados en este trabajo de tesis, por lo que ahora estos resultados son concentrados en la

Cajón	N_{clases}	Complejidad temp.
Cajón I PA/CMC	11	$f(m) = 286m + 359$
Cajón II PA/CMC	6	$f(m) = 156m + 199$
Cajón III PA/CMC	5	$f(m) = 130m + 167$
Cajón IV PA/CMC	9	$f(m) = 234m + 295$
Cajón I CML/CMR	11	$f(m) = 286m + 359$
Cajón II CML/CMR	6	$f(m) = 156m + 199$

Tabla 5.10: Complejidad computacional para LAMDA, según N_{clases}.

Cajón	LAMDA (%)	RNAs (%)
Cajón I PA/CMC	99.7	98.8
Cajón II PA/CMC	97.75	63.14
Cajón III PA/CMC	51.03	41.84
Cajón IV PA/CMC	99.93	96.85
Cajón I CML/CMR	66.89	88.97
Cajón II CML/CMR	39.68	60.32
Promedio	75.83	74.98

Tabla 5.11: Porcentajes de reconocimiento para LAMDA y RNAs.

Tabla 5.11 donde se puede apreciar que, en la mayoría de los casos, LAMDA resultó con un mejor desempeño. Ya se ha descrito también la manera en que se ha llevado a situaciones críticas a LAMDA y a RNAs en fase de reconocimiento de patrones, por lo que ahora se presta atención en cuál de ellos se conportó mejor a lo largo del proceso de experimentación. Esto último también es posible vislumbrarlo en la Tabla 5.11 y será discutido en la siguiente sección.

A parte de las ventajas que se han obtenido cuantitativamente de LAMDA sobre las RNAs, existen otras que pueden proporcionar grandes facilidades a un usuario avanzado, al proveer mantenimiento al SPAT-135. Tales ventajas son:

- Es posible que el algoritmo genere clases de manera automática y en línea, ahorrando tiempo de mantenimiento.
- Es posible repetir el resultado de clasificación de manera determinística.
- Es posible analizar de manera explícita todo el proceso de particionamiento y sus resultados.
- Es posible entrenar a LAMDA en modo supervisado.

Se tienen también algunas desventajas:

- Existen otros algoritmos que han demostrado mejor rendimiento en diagnóstico de fallas[14]; tal es el caso de las SVMs, cuya aplicación es llevada a cabo para diagnosticar fallas en una red eléctrica de potencia.
- Cuando los datos se hacen muy variables en rangos pequeños[19], el algoritmo se confunde (como en la Sección 5.6), de modo que el enfoque del Aproximador Universal resulta mejor.

5.10. Conclusión de la comparación

Como primera conclusión se tiene que LAMDA es mejor, presentando en promedio un 75.83 % (contra 74.98 % de las RNAs) de reconocimiento y siendo mejor en el 66.66 % de los experimentos[20]. Pero es mucho mejor RNAs en cuanto a complejidad computacional, en una proporción de $3:1$. Se prefiere tomar como criterio dominante el hecho de que LAMDA ha sido mejor en la mayoría de los experimentos y con un mejor promedio en el porcentaje de reconocimiento, debido a que se tiene planeado para la implementación del sistema de diagnóstico, un equipo de cómputo lo suficientemente potente (en tiempo y recursos) como para solventar cualquier necesidad; siempre y cuando se tenga certeza del diagnóstico que se emite al usuario y una mayor facilidad de mantenimiento de la aplicación.

[19]El proceso de normalización hace más confusas las señales cuya variabilidad es frecuente y se da en intervalos muy pequeños.

[20]Se obtiene tal valor dado que en 6 experimentos globales, LAMDA fue mejor en 4 de ellos; es decir, $\frac{4}{6} = 0.66$.

Capítulo 6

Implementación y resultados

En este capítulo se mostrará el proceso de implementación del módulo de software de diagnóstico basado en una técnica de reconocimiento de patrones, según la conclusión del capítulo anterior: LAMDA. Para este fin, se ha usado *LabWindows/CVI* (de National instruments) como herramienta de desarrollo. Bajo la premisa de que la interfaz que se desarrolló aquí representa un trabajo no requerido como objetivo de este trabajo de tesis, se supone únicamente como parcial y que no está dirigida para ningún usuario. Dicho lo anterior, el lector deberá tomar como entendido que lo explicado a continuación, esté orientado de manera dominante a aspectos técnicos del software y no como una guía sobre el uso de tal interfaz.

LabWindows/CVI es un entorno de desarrollo gráfico en ANSI C, optimizado para aplicaciones de ingeniería, cuya finalidad son la pruebas y las medidas a través de hardware externo, mismo que es posible configurar de una manera muy transparente y eficaz. Integra bibliotecas especializadas para el procesamiento de señales y despliegue de información, además su modo de uso está debidamente documentado por el fabricante. Se ha elegido dicha herramienta en lugar de LabView (también de National Instruments), debido al mayor control que se tiene sobre los recursos del hardware (una tarjeta de adquisición para el caso de este trabajo). También se revisaron opiniones de programadores con experiencia en el uso de los productos de desarrollo de NI, verificándose que la industria tiene preferencia por el uso de LabWindows/CVI bajo el entendido simple de que al lenguaje de diagramas no es posible aplicarle métricas de calidad, en tanto que al código sí. Esto último, en efecto, implica que es posible optimizar las aplicaciones tanto en tiempo como en recursos[1], lo que plantea grandes ventajas con respecto al uso de LabView.

[1]Se sabe que en un entorno de desarrollo que cuenta con demasiados módulos, cada uno de éstos generalmente están pensados para un gran número de casos de uso, lo que implica demasiado código que no se utiliza en una aplicación en particular. De esta manera, una aplicación desarrollada en LabView es muchísimo más ‘‘pesada’’ que una que está desarrollada en LabWindows/CVI.

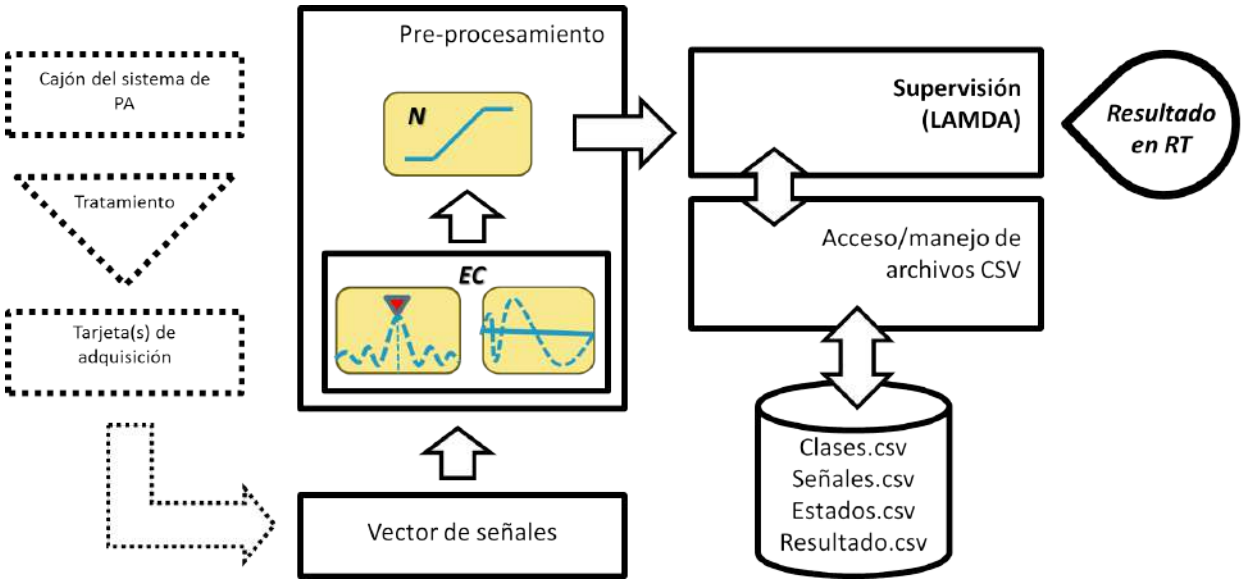


Figura 6.1: Diseño modular del software de diagnóstico (elementos marcados con líneas continuas).

La implementación del software, se ha llevado a cabo tomando como referencia un diseño modular. Puesto que básicamente se tiene como objeto de análisis a un sistema electrónico (un cajón del PA), el planteamiento general del diseño del módulo de software, ha de verse cómo un conjunto de componentes individuales, antes que como un algoritmo. En la Figura 6.1, se muestran de manera general los sub-módulos del diseño, mismos que en adelante serán expuestos de manera particular como secciones de este capítulo.

6.1. Pre-procesamiento

Para el desarrollo de este trabajo de tesis se ha realizado la recolección de información mediante la adquisición masiva de muestras de las señales de salida de cada cajón del PA del metro, usando hardware de National Instruments y LabView. Después de realizar una inspección exhaustiva de la información recopilada y por supuesto, después de las respectivas consultas interactivas con el personal que labora en el mantenimiento correctivo del Pilotaje Automático de STC Metro, se ha determinado que de las ocho técnicas mostradas en la Sección 2.9, sólo fue necesario el uso de dos de ellas[81]: *Análisis espectral por Transformada de Fourier*[2]; para la obtención de la componente de frecuencia de mayor energía en un conjunto muestral de entrada. También se usó una técnica simple de *filtrado* paso-bajas

[2]Otra técnica de análisis espectral muy usada es la Transformada Wavelet; sin embargo, dadas las características del análisis requerido, *las Wavelets* presentan la desventaja (para la aplicación aquí propuesta) de extraer información tanto temporal como espectral, lo que repercute en la exactitud que se requiere en el análisis espectral (refiérase al Principio de indeterminación o *Heisenberg uncertainty principle* [86]). Por otro lado, se tendría información redundante (correspondiente al dominio del tiempo).

(LPF), lo cual no es más que la obtención de la media aritmética de las muestras de voltaje en una ventana de achura fija.

El primer sub-módulo implementado es el de extracción de características ('EC' en la Figura 6.1), del cual su estructura se basa en parte del trabajo realizado por el equipo de desarrollo del SPAT-135, encargado del diseño y construcción de hardware y medición (elementos marcados con líneas punteadas en el diagrama de la Figura 6.1). Las funcionalidades que se describen en esta sección se hallan escritas en un archivo llamado `inputSignalTreating.c` incluido en el proyecto de la aplicación y en el Apéndice D.

Este sub-módulo también se encarga de actuar como interface de software con el exterior, almacenando tantos vectores de muestras le permita la(s) tarjeta(s) de adquisición ('Vector de señales' en la Figura). La tarea anterior es posible llamando a la función `DAQmxReadAnalogF64()`, incluida en la biblioteca NI-DAQmx de LabWindows/CVI. Dicha función recibe como parámetros principales los siguientes: `taskHandle` y `numSampsPerChan`. El primero se trata de un objeto que devuelve el driver de la tarjeta de adquisición a través de un *task*[3], para direccionar las propiedades de dicho hardware y el segundo es el número de muestras que serán adquiridas por cada canal. La función `DAQmxReadAnalogF64()`, devuelve por referencia los elementos de un arreglo de datos `readArray[]`, el cual contiene una muestra de cada canal por llamada[4]. De esta manera, `readArray[]` es el vector de señales que han de ser pre-procesadas antes de hacer el diagnóstico. Para esto último, el sub-módulo EC toma como referencia de pre-procesamiento seis vectores mapa: `mapaCajonI_PACMC[]`, `mapaCajonII_PACMC[]`, `mapaCajonIII_PACMC[]`, `mapaCajonIV_PACMC[]`, `mapaCajonI_CMLCMR[]` y `mapaCajonII_CMLCMR[]`, cuyos elementos son caracteres que codifican el tipo de señal que ha de pre-procesarse (Véase la Tabla 6.1) para cada cajón del PA. De esta manera, según sea el cajón y según sea la combinación de componentes de su respectivo vector mapa, el sub-módulo EC toma a `readArray[]` como entrada para calcular ya sea la frecuencia fundamental de la señal muestreada en un canal mapeado como señal de CA (códigos: `'b','c','d','e'` y `'f',`); o bien, el valor promedio para una señal muestreada en un canal mapeado como señal de CD (código: `'a'`). Se ha hecho mayor distinción en los códigos de las señales de CA, ya que la llamada a la función `SingleToneInfo()` recibe como parámetro un rango de frecuencia en el cual se espera encontrar la fundamental. Dicha función es llamada en cada adquisición, para medir la frecuencia fundamental del conjunto de muestras adquirido, usando como referencia el vector mapa. Específicamente, `SingleToneInfo()` recibe como parámetros un arreglo de datos (`waveform[]`) que serán objeto de procesamiento, el periodo de muestreo (`samplePeriodInSeconds`) y un rango de frecuencias entre las que

[3]Este y otros términos que puedan ser usados a lo largo del capítulo, se pueden consultar en la ayuda de LabWindows/CVI ya que son propios del entorno.

[4]Los datos en el arreglo, estarán dispuestos según lo indique el parámetro de modo de rellenado: si `fillMode`=1 entonces las muestras estarán intercaladas; es decir, una muestra de cada canal por cada elemento consecutivo de `readArray[]`, repitiendose este patrón tantas veces en el arreglo como número de canales se estén leyendo (`readArray[`$c_1, c_2, c_3, ..., c_1, c_2, c_3, ..., c_1, c_2, c_3$`]`). Si `fillMode`=0, entonces se colocan grupos de muestras de un canal en posiciones consecutivas del arreglo (`readArray[`$c_1, c_1, c_1, ..., c_2, c_2, c_2, ..., c_3, c_3, c_3$`]`).

código	tipo de seãnl
'a'	CD
'b'	100 KHz
'c'	23 KHz
'd'	1488 Hz
'e'	1188 Hz
'f'	1140 Hz
'g'	1040 Hz

Tabla 6.1: Código de las componentes de los vectores mapa.

debería encontrarse la frecuencia fundamental de la señal (`*searchType`). Por otro lado, devuelve por referencia la frecuencia fundamental (`frequency`), la amplitud y la fase de `waveform[]`. Para realizar el preoprocesamiento de las señales de CD se ha seguido el mismo procedimiento, sólo que en lugar de llamar a `SingleToneInfo()` se llama a `Median()`, que obtiene la media del conjunto de muestras que recibe como parámetro en forma de un arreglo de datos (`inputArray`) y devuelve por referencia el valor de la media (`*median`). Este último procesamiento es sumamente sencillo; sin embargo, es de mucha importancia porque básicamente se está implementando con ello un filtro pasa-bajas, cuya finalidad es la eliminación de ruido en las señales de CD.

El siguiente sub-módulo llamado 'N', recibe como entrada un vector de señales filtradas o que representan valores de frecuencias. Tiene como finalidad la normalización de los valores de entrada implementando la Ecuación (4.10). El pre-proceso de normalización es llevado a cabo en tiempo real, detectando en cada adquisición los valores máximo y mínimo de cada canal. Lo anterior, sugiere un reescalamiento dinámico de cada canal hacia valores difusos, lo que explica el porqué no es necesario hacer distinciones entre los diferentes rangos de valores de las señales de CD para los vectores mapa (todas las señales de CD son código `'a'`).

6.2. Acceso/manejo de archivos CSV

Se ha implementado un sub-módulo para el manejo de archivos, con tres finalidades: 1) acceder a archivos que contengan señales que hayan sido capturadas previamente, 2) almacenar los centros paramétricos de las clases identificadas durante el proceso de aprendizaje de LAMDA y 3) una vez almacenados (aprendidos) los centros paramétricos, acceder a ellos durante un proceso en fase de reconocimiento de patrones. Todos los archivos a los cuales accede la aplicación deberán estar en formato `*.csv` (valores separados por comas), puesto que este tipo de archivos es fácil de manipular con la librería usual para manejo de archivos en C (incluida también en LabWindows/CVI). Las funciones cuyo funcionamiento se describe a continuación, se hallan escritas en un archivo de código fuente llamado `fileDriving.c`

y la explicación sobre los valores que reciben como parámetros y los que devuelven ya sea como referencia o por valor, está documentada en las definiciones respectivas en código y en el Apéndice C.

Se ha diseñado una estructura única y simple que deberá tener el contenido de estos archivos, tanto para los centros paramétricos (clases) como para archivos producto de la captura de señales. Dicha estructura se muestra en la Figura 6.2. El primer renglón del archivo, es el *encabezado* y define las dimensiones de la matriz de datos implícitamente escrita en dicho archivo. Los primeros tres dígitos del encabezado, le dicen a la aplicación cuántos descriptores tiene cada objeto de entrada y los siguientes cinco, le indican cuántas muestras (o centros paramétricos) contiene el archivo. Lo anterior, a efecto de que la función encargada de cargar en memoria RAM el contenido del archivo, sepa dónde empezar y donde terminar. La función encargada de tratar con el encabezado, extraer y asegurar la integridad de la información contenida en él, se llama `matrixClasesConfig()`. El siguiente renglón se trata del número de elementos que han sido asignados a cada clase. Cada columna es una clase (centros paramétricos $\boldsymbol{\rho}_k$ de cada clase C_k). Véase además, que todos los elementos de la primera columna son iguales a 0.5 (sólo en el caso de un archivo de clases), esto porque dicha columna representa a la clase NIC o clase 0. Los renglones siguientes, son los descriptores x_j. Para un archivo de señales, las columnas son las muestras $\mathbf{x}^{(i)}$. El segundo renglón del archivo no es tomado en cuenta y se rellena con ceros. La función implementada para llevar a cabo la carga de archivos de centros paramétricos y de señales (en la Figura 6.1 `clases.csv` y `señales.csv` respectivamente), se ha llamado `loadFile()`. La columna de ceros del archivo es necesaria como *token* para asegurar la integridad de los datos[5]; en caso de omitirla, habrá errores. Dentro de este mismo conjunto de funciones implementadas se tiene a `storeKnoledgeFile()`, cuya finalidad es manejar un archivo bajo el mismo esquema que se ha descrito, pero para almacenar el conocimiento adquirido por LAMDA en fase de aprendizaje (también en `clases.csv`).

Se implementó también una función llamada `loadStates()`, cuya finalidad es cargar en memoria un conjunto de cadenas de texto, con las que ha sido etiquetada cada clase aprendida por LAMDA; dicho de otra forma: el archivo de estados asociados a cada clase (`estados.csv`). El llenado de dicho archivo debe realizarse manualmente, puesto que es información que el usuario de mantenimiento del PA proveerá al usuario de mantenimiento del SPAT-135. La aplicación interpreta cada renglón consecutivo del archivo como una etiqueta para cada clase consecutiva; es decir, la cadena escrita en el renglón número 1 del archivo, corresponde a la clase $k = 0$, el renglón número 2 del archivo, corresponde a la clase $k = 1$ y así sucesivamente. La longitud de cada cadena escrita en el archivo por renglón, puede ser cualquiera; sin embargo, `loadStates()` usa el valor de la constante global `MAX_STATE_STRING_LENGTH`, definida al principio del archivo en el que se describe la función, para establecer el tamaño máximo de dichas cadenas. Aunque el usuario escriba más caracteres en un renglón del archivo, sólo serán tomados en cuenta para la etiqueta de estado,

[5]Se tienen por separado funciones especializadas en la validación de archivos de la aplicación.

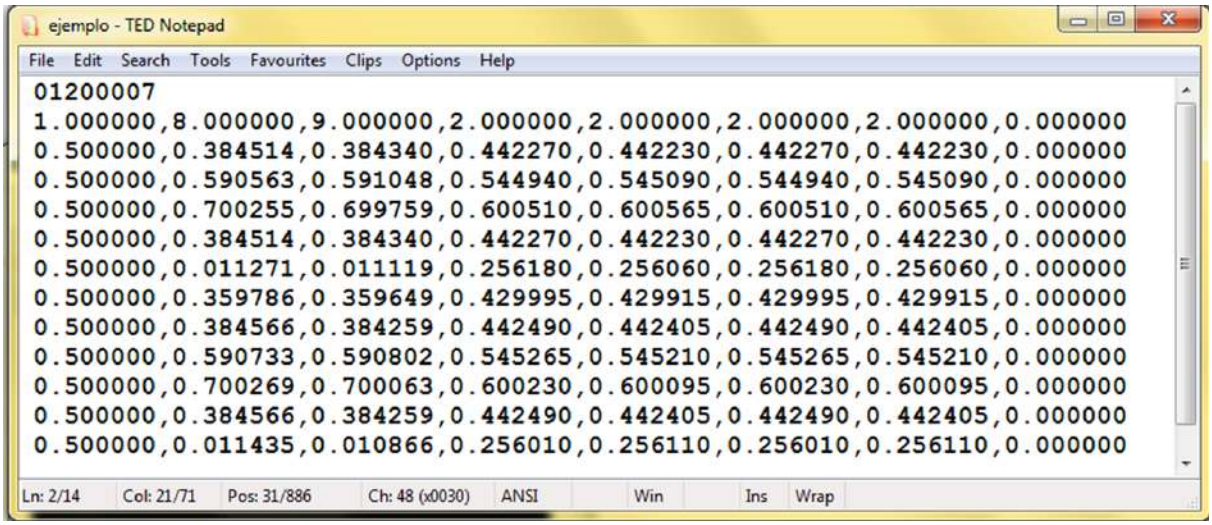


Figura 6.2: Diseño de la estructura de un archivo `*.csv`, en el que se almacenan centros paramétricos y señales adquiridas que carga la aplicación. Nótese que los valores se encuentran normalizados.

una cantidad de caracteres igual o menor a `MAX_STATE_STRING_LENGTH`. En la Figura 6.3, se tiene un ejemplo de la estructuración del archivo de estados.

6.3. Supervisión (LAMDA)

La implementación de LAMDA, es el resultado de todo el análisis hecho en este trabajo de tesis. Este sub-módulo, ha sido el que se ha tornado más complejo de implementar, ya que al principio se tenía poco conocimiento sobre los detalles que engloba el algoritmo. Por otra parte, el hecho de tener la característica de clasificación automática, lo hace complicado de implementar debido a que es necesario realizar manejos intrincados de memoria dinámica, suscitados por el hecho de que los archivos de clases y señales, pueden tener cualquier dimensión no esperada. En cuanto a la dimensión de los vectores de entrada, se pudieron haber manejado dimensiones fijas; sin embargo, esto volvería difícil de mantener la aplicación y, aparte, sin escalabilidad y sin posibilidades de uso para trabajos de investigación futuros. En el archivo que contiene la descripción de funciones que implementan LAMDA (`lamda.c`), están escritas tres funciones llamadas `LAMDA_LFF(), LAMDA_OLL()` y `LAMDA_ROL()`. El código puede ser consultado en el Apéndice E.

La primera (`LFF`: *Learning From File*) es la más compleja de todas, pues al tiempo de ejecutar el algoritmo de LAMDA, emula las condiciones en las que se encontraría la aplicación en una situación real de diagnóstico. Lo anterior es así, debido a que se hace el costoso manejo

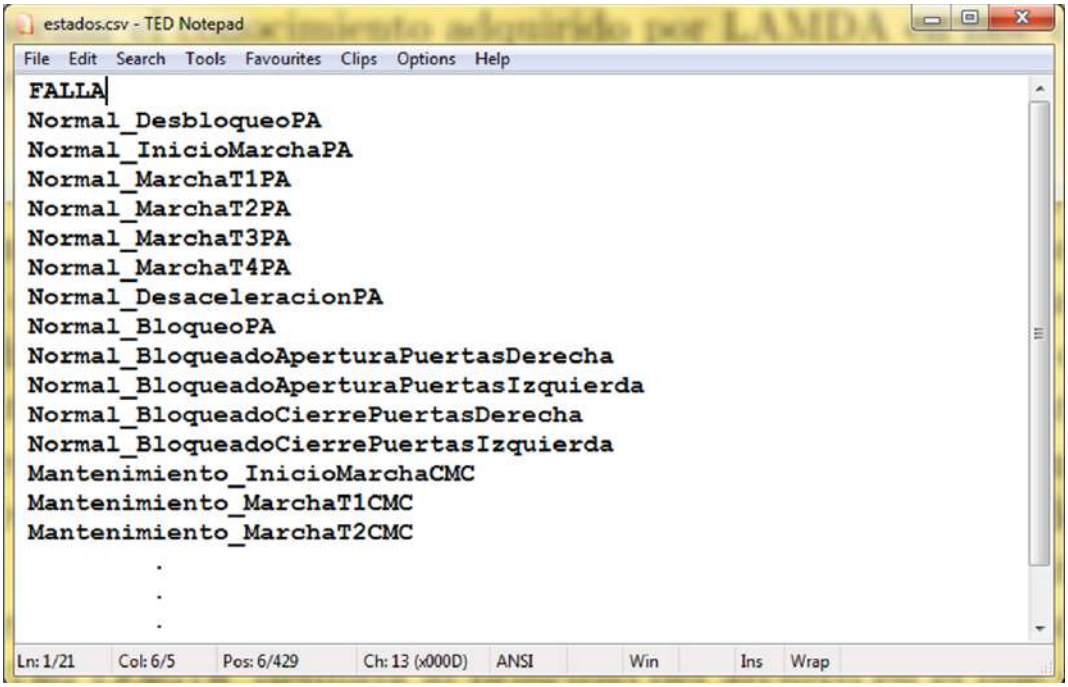


Figura 6.3: Ejemplo de un archivo de estados (`estados.csv`).

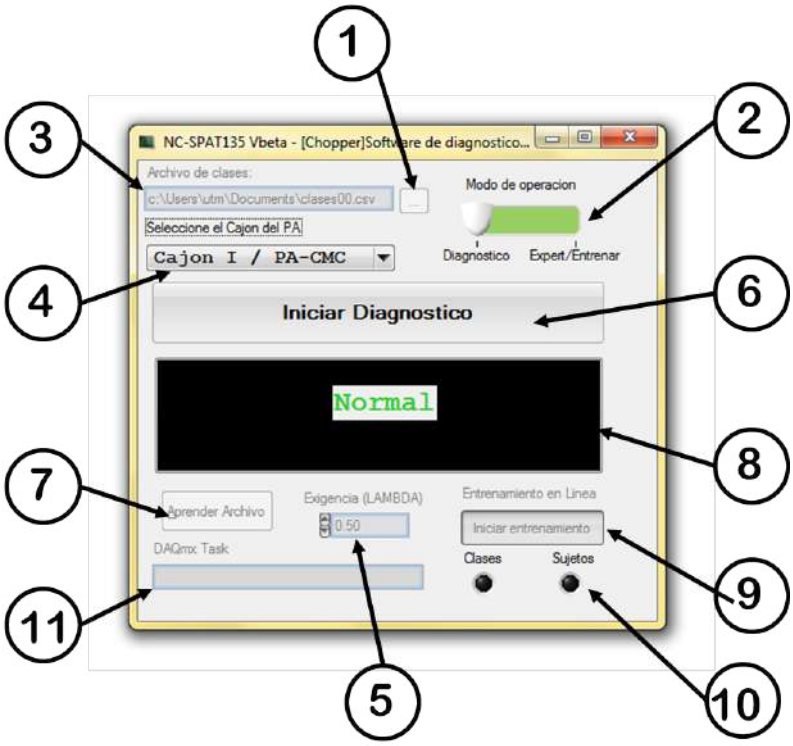


Figura 6.4: GUI experimental del NC-SPAT135. Véase cada elemento de manera detallada en el Apéndice A.

de archivos[6] de señales capturadas previamente; señales que, desde luego, son genuinas. Cada muestra tomada (componente de un objeto de entrada) desde el archivo, con la ayuda de la función `loadFile()`, es tratada como si fuera una muestra proveniente del exterior, sólo que ya se encuentra normalizada y es procesada directamente por LAMDA. `LAMDA_LFF()` ha sido implementada con fines puramente de investigación, ya que tanto el usuario final como el usuario de mantenimiento al SPAT135, no se verán en absoluto envueltos en un caso de uso de dicha función, lo que implica que el botón que la llama también queda reservado como herramienta para propósitos de investigación en este trabajo de tesis (botón de control marcado con el número 7 en la Figura 6.4).

Antes de continuar, veremos un esquema de la interfaz gráfica de usuario diseñada para propósitos experimentales[7], a efecto de exponer en adelante las funcionalidades de cada función implementada para LAMDA. En la Figura 6.4, se encuentra la GUI y se pueden apreciar cada uno de sus componentes. En la Apéndice A, se tiene una tabla con el identificador que define a cada uno de ellos en el código fuente[8]. De momento, se tiene a continuación una lista simplificada:

1. *Selector de archivos.*
2. Selector del *modo de operación.*
3. *Monitor de archivo cargado.*
4. *Selector del cajón de PA.*
5. *Parámetro de exigencia.*
6. *Botón de inicio de diagnóstico.*
7. *Botón de aprendizaje desde archivo.*
8. *Monitor de estado.*
9. *Botón de aprendizaje en línea.*
10. *Confirmador de archivos cargados.*
11. *Identificador de task.*

[6]Dichos archivos pueden llegar ser de dimensiones cercanas a los 30MBytes, conteniendo caracteres en código ASCII que representan cada muestra capturada y normalizada.

[7]Ya se ha hecho mención en la Sección 1.2, que existe una parte del equipo de desarrollo del SPAT135 (usabilidad) quienes se encargarán del diseño e implementación de la GUI del producto final.

[8]Los elementos 5, 7, 9 10 y 11 sólo se activan y son de utilidad en modo `Experto`.

La llamada a `LAMDA_LFF()`, sólo es posible hacerla a través del botón marcado con el número 7 en la Figura 6.4, lo que implica que el usuario debió haber seleccionado el modo de operación `Experto`. La primera funcionalidad para la que está programada `LAMDA_LFF()`, es darle al usuario la opción de elegir la ubicación de `clases.csv`, a través de un cuadro de diálogo usual de Windows[9]. Una vez elegido dicho archivo, `LAMDA_LFF()` lanza otro cuadro de diálogo para elegir el archivo de señales. Por defecto, el parámetro de exigencia esta seleccionado en 0.5 (elemento número 5), de manera que si el usuario desea otro valor deberá ajustarlo antes de pulsar el botón `Aprender archivo`, ya que una vez pulsado no es posible ajustar λ sino hasta que se termine el proceso de clasificación. Cuando el usuario haya elegido el archivo de señales en el cuadro de diálogo, pulsando el botón `Load`, el proceso de entrenamiento inicia en segundo plano. Una vez terminado dicho proceso, la interfaz lanza un mensaje indicando que la operación fue exitosa; dicho de otra manera, el entrenamiento a partir del archivo de señales capturadas ha finalizado y podrá verificarse que éste ya contiene datos (o los modificó en su caso).

La segunda función implementada para LAMDA, se llama `LAMDA_OLL()` y tiene la misión de hacer que la aplicación aprenda automáticamente en línea (*OLL: On-Line Learning*[10]). La aplicación la llama pulsando el botón `Iniciar entrenamiento` (marcado con el número 9 en la Figura 6.4). Antes de usar dicho botón, es necesario definir el archivo de clases sobre el que trabajará la función, lo cual se hace pulsando el botón número 1, que es el selector de archivos. Al pulsar dicho botón, es lanzado un cuadro de diálogo donde se escoge el archivo de clases. Después de pulsar `Load` en el cuadro de diálogo, éste se cierra y la ruta del archivo que será cargado queda mostrada en el indicador número 3 de la interfaz. Ya que se ha elegido el archivo de clases, el usuario pulsa `Iniciar entrenamiento`, el método del botón (`onLineLearning()`) carga en RAM el archivo de clases, reserva la cantidad de memoria que indique el vector mapa del cajón seleccionado (selector número 4 en la Figura 6.4) y se activa inmediatamente *timer*, llamando a su método `realTimeR_Recognition()` el cual ejecuta el código en su cuerpo (incluyendo a `LAMDA_OLL()`) cada vez que ocurre un `EVENT_TIMER_TICK`. La función `LAMDA_OLL()` recibe como parámetro un objeto de entrada (un arreglo llamado `norData[]`) en cada disparo del *timer*, dicho objeto proviene del sub-módulo de pre-procesamiento (ya no desde un archivo como en el caso de `LAMDA_LFF()`); específicamente desde la función `ProcessingProfile()`, incluida también en el cuerpo del método de *timer* y llamada justamente antes que `LAMDA_OLL()`. Mientras el algoritmo aprende clases nuevas o reconoce las que ya existen, el resultado se irá desplegando en indicador número 8 de la interfaz.

La función `LAMDA_ROL()` (*ROL: Recognition On-Line*), es la más sencilla de todas, pues se limita a cargar el archivo de clases seleccionado por el botón número 1 de la interfaz y

[9]El usuario deberá evitar el uso del botón ***bibliotecas*** que se presenta a la izquierda de cualquier cuadro de diálogo para selección de archivos, ya que LabWindows no obtiene datos a partir de ese modo de navegación y por lo tanto, se producirá un error en tiempo de ejecución.

[10]Entiéndase el término *en línea*, como la acción continua de la aplicación de adquirir muestras desde el exterior (desde la tarjeta de adquisición de datos) y procesarlas en tiempo real.

exclusivamente a reconocer las clases que le permita `clases.csv`. Esta función representa solamente la fase de reconocimiento de patrones de LAMDA y no es posible entrenar a la aplicación, además será la de uso más frecuente, ya que el botón que la llama (`Iniciar Diagnostico`) estaría activo en modo `Diagnostico`; es decir, el modo de operación usual para el usuario final. La descripción de su funcionamiento y contexto son los mismo que para `LAMDA_OLL()`, pero con la diferencia interna de que mientras `LAMDA_ROL()` se está ejecutando, la aplicación no aprende y se limita a desplegar ya sea clases o estados reconocidos (incluyendo los no reconocidos: `FALLA`), a través del indicador número 8 de la interfaz.

6.4. Resultados

En esta sección, se presentan primeramente los resultados que se obtuvieron después de entrenar al NC-SPAT 135 con un conjuto de datos que representan a un *registro de comportamiento*[11] en estado de funcionamiento correcto del PA; posteriormente, se muestran resultados que el software implementado brinda, a partir de los registros de comportamiento, como información interpretable acerca del funcionamiento correcto de cada cajón del PA, a tal información se le ha llamado *modelo de comportamiento*; así mismo, se ha obtenido información a la que se ha llamado *modelo de comportamiento en falla*, misma que puede ser intepretada para describir el funcionamiento incorrecto de cada cajón del PA. Por útimo, se expone la interpretación de los modelos de comportamiento obtenidos. En la Figura 6.5a se tiene una fotografía de los elementos usados para llevar a cabo el procedimiento anterior.

Se ha obtenido un registro de comportamiento caracteriza cuantitativamente a un cajón del PA, apartir de conjuntos de señales de voltaje adquiridas en tiempo real desde cada cajón que se sabe a priori que funciona correctamente, de manera que los centros paramétricos que LAMDA ha aprendido a partir de cada vector de muestras de estas señales, se presentan como evidencia del entrenamiento. Dichos registros de comportamiento son producto de una, dos o tres pruebas de cajón seleccionadas en base a la experiencia del personal experto en el Banco Simulador de Pruebas, de manera que se sepa que dichas pruebas presentarán comportamientos diferentes cuando el PA funcione correctamente y cuando el PA presente diferentes falla conocidas. Las fallas son inducidas en cada cajón haciendo uso de una tarjeta electrónica dañada que reemplaza a una que funciona correctamente. Cabe señalar que sólo se cuenta con una tarjeta dañada que corresponde a cada cajón del PA (véase la Figura 6.5b), de manera que las pruebas fueron realizadas mediante el siguiente procedimiento:

1. Para cada cajón del PA, se identificaron pruebas de cajón en las cuales se manifiesten fallas debido al reemplazo de una tarjeta que funciona correctamente, por la correspondiente tarjeta dañada. Sépase que la manifestación de fallas se identifica en el Banco

[11] Llámese *registro de comportamiento* a la matriz de centros paramétricos $\boldsymbol{\rho} = \{\boldsymbol{\rho}_1, \boldsymbol{\rho}_2, .., \boldsymbol{\rho}_k, .., \boldsymbol{\rho}_l\}$, representativa de la partición $C = \{C_1, C_2, ..., C_k, ..., C_l\}$ del espacio de entrada.

Simulador de Pruebas de PA y que se ha buscado que la cantidad máxima de prubas seleccionadas, aportan información suficiente para presentar resultados de diagnóstico. A las pruebas seleccionadas en este paso, se les ha llamado *pruebas piloto*.

2. Se configuró la aplicación en modo de aprendizaje. Para cada cajón se verificó la conveniencia de la clasificación de LAMDA con diferentes valores del parámetro de exigencia, bucando obtener una partición difusa lo sufientemente tolerante a la variabilidad de los descriptores de entrada. Lo anterior, será llevado a cabo para las pruebas piloto.

3. Para cada cajón, se realizó el entrenamiento de LAMDA en tres ocasiones[12] con las pruebas piloto mientras el PA funcionaba correctamente, de manera que el registro de comportamiento respectivo se genere y se estabilice lo suficiente.

4. Para cada cajón, la herramienta de software se configuró en modo de reconocimiento de patrones, de manera que usara el registro de comportamiento almacenado para procesar datos generados desde el cajón, mientras se llevaban a cabo las pruebas piloto. La secuencia de índices de clases a las cuales fueron asociados los patrones de entrada en cada adquisición de datos, fue almacenada en un archivo etiquetado con el número de cajón y prueba en la que fue generado.

5. Para cada cajón, se hizo el intercambio correspondiente de la tarjeta electrónica dañada. El registro de comportamiento, fue usado nuevamente para procesar datos generados desde el cajón, mientras se llevaban a cabo las pruebas piloto. La secuencia de índices de clases a las cuales fueron asociados los patrones de entrada en cada adquisición de datos, fue almacenada en un archivo etiquetado con el número de cajón y prueba en la que fue generado; indicando además, que ahora existen fallas y cuáles son.

6. Las secuencias de obtenidas en los pasos 4 y 5, son extensas[13], de longitud variable y son variantes en cada vez que una prueba se realiza; por lo que fueron analizadas con más detalle y se determinó que existe en ellas una distribución frecuencial de reconocimiento de clases, específica para cada prueba realizada y que dicha distribución se mantenía independientemente del número de veces que cada prueba se repitiera. Por lo tanto, la distribución frecuencial de reconocimiento en una prueba de cajón, pudo proveer un modelo de comportamiento para cada cajón, según la prueba que se le estubiese realizando y según la falla que se estubiese induciendo. De esta manera, se obtuvieron modelos de comportamiento por prueba en estado de funcionamiento correcto y modelos de comportamiento por prueba en falla.

[12]Se ha encontrado que si se lleva a cabo el paso 2, la partición difusa adquiere suficiente estabilidad en el tercer ciclo de entrenamiento; es decir, repetir cada prueba piloto en tres ocasiones, hasta que el entrenamiento se torne suficientemente estable, de manera que LAMDA ya no modifique la cantidad de conjuntos en la partición del espacio de entrada.

[13]Su tamaño es igual al número de adquisiciones de datos que se hayan realizado en una prueba de cajón.

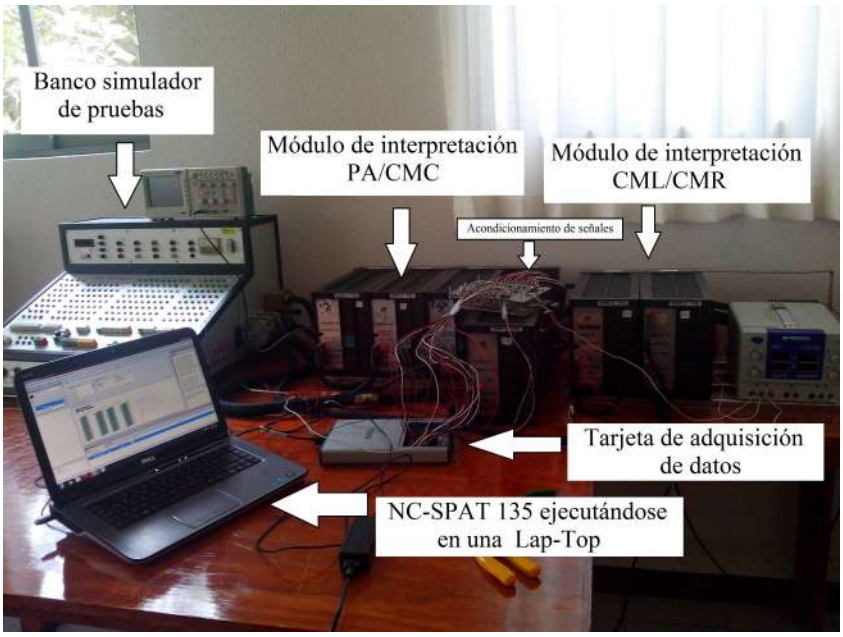
7. Como último paso, se expone una interpretación de los modelos de comportamiento, a efecto de reportar sus características y cómo usarlos para determinar que existen fallas y de qué se tratan éstas.

Un ejemplo de la interpretación de un modelo de comportamiento, se puede plantear con la Figura 6.6a. Obsérvese que el eje horizontal representa el índice de cada clase que LAMDA ha reconocido durante la prueba 3, hecha al cajón I PA/CMC. Y el eje vertical, representa la frecuencia normalizada con que cada una de estas clases es reconocida a lo largo de dicha prueba. En la figura, se ve que la clase 14 es la más frecuentemente reconocida durante la prueba, ya que su frecuencia normaliazada es 1. Además se puede ver que las clases 6 y 7, son reconocidas 70 % menos veces que la clase más frecuente (la clase 14); así mismo, la clase 12 es reconocida por LAMDA con una frecuencia normalizada de 0.4; 60 % menos que la clase más frecuente.

Ahora, supóngase que se requiere verificar si el Cajón I PA/CMC está fallando y se ha sometido a la prueba 3. Al término de dicha prueba, la aplicación generó el modelo de comportaiento de la Figura 6.6b, de manera que es suficiente verificar que existan diferencias significativas entre el modelo de comportamiento de referencia (Figura6.6a) y el obtenido: véase que en principio, la clase más frecuente en la Figura 6.6b no es la 14, como en el caso de la referencia; lo que ya representa una diferencia significativa. El hecho de que una o algunas clases sean reconocidas con mucha mayor frecuencia que las demás, significa que durante la prueba dichas clases representan un comportamiento constante en intervalos de tiempo prolongados, o bien un estado estacionario de comportamiento; de manera que si dichos comportamientos prolongados o estacionarios cambian por completo, con respecto a la referencia, significa categóricamente un estado de falla.

Una vez que se ha expuesto el análisis anterior, nótese que en general, la diferencia que separa a los modelos que representan un funcionamiento correcto (lado derecho de las figuras) de los que representan fallas (lado izquierdo), es que la mayoría de las clases que se reconocen con frecuencias bajas (e. g. 0.1-0.35) en un modelo de funcionamiento correcto, se reconocen con mucha menor frecuencia o no se reconocen en el modelo de comportamiento en falla y viceversa. Por otro lado en algunos casos, las clases más frecuentemente reconocidas coinciden como en las figuras 6.6e y 6.6f, pero es fácil determinar que existe una falla observando las demás clases, por ejemplo: la clase 17 tiene una frecuencia normalizada de 0.6 en el modelo de comportamiento, pero en el modelo que representa una falla, dicha clase no se reconoce en absoluto. Existen pocos casos en los que no es tan claro, como en el caso de las Figuras 6.6c y 6.6d; sin embargo, las clases menos frecuentes (clase 5), siguen aportando información sufciente.

En las Tablas 6.3 a 6.8, se tienen los registros de comportamiento obtenidos mediante el NC-SPAT 135. Las columnas representan cada centro paramétrico $\boldsymbol{\rho}_k$ obtenido para un cajón, N es el número de elementos asignados a cada clase durante el entrenamiento y los



(a)

(b)

Figura 6.5: (a) Montaje de elementos para realizar pruebas de cajón, entrenamiento y pruebas finales del NC-SPAT 135 y (b) tarjetas electrónicas dañadas para obtener modelos de comportamiento en falla. El STC Metro, ha proporcionado una tarjeta para cada cajón, con la finalidad de caracterizar la falla más frecuente.

renglones representan las componentes ρ_j de cada centro paramétrico; nótese que la primera columna de cada tabla, corresponde a la clase NIC.

En las Figuras 6.6a a 6.6f, se tienen los modelos de comportamiento de las pruebas 3, 5 y 7 para el Cajón `I` - PA/CMC que caracterizan el funcionamiento correcto del equipo; así mismo se tienen los modelos de comportamiento en falla de las mismas pruebas para caracterizar las fallas {*''Programa no alimentado'', ''piloto no disponible'', ''imposible traccionar''*}, mismas que se inducen al reemplazar la tarjeta de `Circuitos Anexos`.

En las Figuras 6.7a a 6.7f, se tienen los modelos de comportamiento de las pruebas 38, 40 y 49 para el Cajón `II` - PA/CMC que caracterizan el funcionamiento correcto del equipo; así mismo se tienen los modelos de comportamiento en falla de las mismas pruebas para caracterizar las fallas {*''Piloto no disponible'', ''energía cable CMC no disponible''*}, mismas que se inducen al reemplazar la tarjeta `Anexo I`.

En las Figuras 6.8a a 6.8d, se tienen los modelos de comportamiento de las pruebas 52 y 57 para el Cajón `III` - PA/CMC que caracterizan el funcionamiento correcto del equipo; así mismo se tiene tienen los modelos de comportamiento en falla de las mismas pruebas para caracterizar la falla {*''No hay tracción debido a ecuación de partida incorrecta''*}, misma que se induce al reemplazar la tarjeta de `Decoder II`.

En las Figuras 6.9a y 6.9b, se tienen los modelos de comportamiento de la prueba 63 para el Cajón `IV` - PA/CMC que caracteriza el funcionamiento correcto del equipo; así mismo se tiene el modelo de comportamiento en falla de la misma prueba para caracterizar las fallas {*''No hay captación'', ''programa no alimentado''*}, mismas que se inducen al reemplazar la tarjeta de `Captación AR`.

En las Figuras 6.10a a 6.10d, se tienen los modelos de comportamiento de las pruebas 95 y 96 para el Cajón `I` - CML/CMR que caracterizan el funcionamiento correcto del equipo; así mismo tienen los modelos de comportamiento en falla de las mismas pruebas para caracterizar las fallas {*''Falla en captación AR'', ''no hay conmutación de captores*''}, mismas que se inducen al reemplazar la tarjeta de `Interfaz de captores AR`.

En las Figuras 6.11a a 6.11f, se tienen los modelos de comportamiento de las pruebas 79, 81 y 83 para el Cajón `II` - CML/CMR que caracterizan el funcionamiento correcto del equipo; así mismo se tienen los modelos de comportamiento en falla de las mismas pruebas para caracterizar la falla {*''Bloqueo de simulación de partida''*}, misma que se induce al reemplazar la tarjeta de `SD2-CMLCMR`.

En la Tabla 6.2, se tienen los nombres de las pruebas cuyas etiquetas se han asignado para agregar simplicidad a la redacción de este tranajo de tesis.

Cabe resaltar que por cuestiones demostrativas, en algunos casos se está haciendo un

Tabla 6.2: Pruebas de cajón, sus etiquetas y nombres.

Prueba de cajón (etiqueta)	Nombre de prueba
03. `tracPA-AV`	Tracción PA dirección AV
05. `tracCMC-AV`	Tracción CMC dirección AV
07. `lampPNA-AV`	Lámpara PNA, dirección AV
38. `umbr255AV_OD`	Umbral 255, dirección AV, puertas abiertas en OD
40. `umbr197AV_OD`	Umbral 197, dirección AV, puertas abiertas en OD
49. `CMC_AV`	Conducción Manual Controlada en dirección AV
52. `iniPA-AV`	Inicio del PA en dirección AV
57. `83Q-AVOD`	Lámpara 83Q en dirección AV, puertas abiertas en OD
63. `captrsVal`	Validación de captores
79. `CML25-AV`	Conducción Manual Libre, tracción 25 en dirección AV
81. `CML50-AV`	Conducción Manual Libre, tracción 50 en dirección AV
83. `CMR`	Conducción Manual Restringida
95. `CML-50-AV`	Conducción Manual Libre, tracción 50 en dirección AV
96. `CML-50-AR`	Conducción Manual Libre, tracción 50 en dirección AR

análisis de tres pruebas[14] de cajón para observar el comportamiento de una falla (por ejemplo para el Cajón I PA/CMC); sin embargo, es claro que una sola prueba de cajón que no se comporte de acuerdo a lo que sugiere la referencia respectiva (como en el caso del Cajón IV PA/CMC, Figuras 6.9a y 6.9b), aporta suficiente evidencia para determinar que el cajón en cuestión está fallando.

[14]En los casos en que sólo se analizaron una o dos pruebas, fue debido a que las fallas provocadas por las tarjetas reemplazadas, sitúan al PA en un estado de inactividad forzada. Este comportamiento es constante independientemente de si se intenta seguir manipulando el banco de pruebas para continuar con más pruebas, de manera que resulta redundante seguir capturando datos.

Tabla 6.3: Registro de comportamiento obtenido para el Cajón I - PA/CMC, con $\lambda = 0.60$.

ρ	ρ_0	ρ_1	ρ_2	ρ_3	ρ_4	ρ_5	ρ_6	ρ_7	ρ_8	ρ_9	ρ_{10}	ρ_{11}	ρ_{12}	ρ_{13}	ρ_{14}	ρ_{15}	ρ_{16}	ρ_{17}	ρ_{18}	ρ_{19}	ρ_{20}	ρ_{21}	ρ_{22}	ρ_{23}	ρ_{24}	ρ_{25}	ρ_{26}
N	1	20	6	21	9	58	43	25	17	45	3	39	12	106	63	10	3	15	16	6	11	10	3	2	5	2	4
ρ_1	0.5	0.413232	0.266194	0.947721	0.915456	0.954095	0.956466	0.947186	0.936478	0.954655	0.181165	0.949084	0.926925	0.045828	0.047236	0.922052	0.814482	0.932452	0.934662	0.884491	0.921962	0.869269	0.81239	0.735007	0.874161	0.734924	0.852769
ρ_2	0.5	0.50208	0.475543	0.489143	0.566321	0.171302	0.502567	0.852591	0.911891	0.924751	0.178985	0.817724	0.896211	0.714073	0.918237	0.481778	0.502469	0.470749	0.478203	0.853492	0.185916	0.453543	0.192525	0.704449	0.858463	0.643155	0.70889
ρ_3	0.5	0.480761	0.486688	0.489214	0.487243	0.488571	0.488457	0.491914	0.489965	0.492236	0.495019	0.491734	0.492947	0.488625	0.489611	0.491148	0.494984	0.490039	0.4895	0.491815	0.487008	0.489434	0.494044	0.496407	0.490237	0.494472	0.491113
ρ_4	0.5	0.491125	0.281422	0.911242	0.870348	0.91783	0.910839	0.909127	0.91027	0.923406	0.343253	0.916992	0.887014	0.921563	0.917591	0.891721	0.799149	0.963762	0.911912	0.849231	0.876649	0.841811	0.770043	0.719242	0.830343	0.698504	0.797365
ρ_5	0.5	0.956335	0.084853	0.061939	0.087591	0.0256	0.038678	0.02576	0.956598	0.988852	0.169345	0.987108	0.042306	0.994809	0.00852	0.107247	0.193008	0.07895	0.968749	0.91662	0.066121	0.95	0.833333	0.250122	0.9	0.75	0.678776
ρ_6	0.5	0.483675	0.422351	0.433649	0.32807	0.505219	0.40647	0.517177	0.539389	0.536775	0.503086	0.529358	0.524574	0.534238	0.537479	0.442113	0.547929	0.430006	0.535362	0.525262	0.44549	0.529967	0.50834	0.504835	0.506101	0.452219	0.478825
ρ_7	0.5	0.41589	0.244069	0.918664	0.895901	0.026609	0.023792	0.898819	0.896372	0.901342	0.194839	0.904023	0.878902	0.902675	0.894922	0.893979	0.787158	0.045152	0.882704	0.840906	0.056888	0.844798	0.176705	0.679408	0.847703	0.721529	0.133681
ρ_8	0.5	0.409384	0.35361	0.289722	0.298424	0.197297	0.954184	0.924022	0.935638	0.950765	0.260002	0.954379	0.085202	0.042883	0.045178	0.500857	0.464094	0.893653	0.924093	0.881617	0.11024	0.864048	0.811203	0.732885	0.144195	0.439605	0.847391
ρ_9	0.5	0.430263	0.357369	0.323925	0.372241	0.443162	0.329912	0.388939	0.269672	0.298637	0.415968	0.461309	0.366436	0.18017	0.334249	0.920198	0.810098	0.897863	0.931466	0.39123	0.454131	0.873582	0.439146	0.530694	0.488082	0.463342	0.443785
ρ_{10}	0.5	0.480463	0.241895	0.922405	0.913038	0.956517	0.936229	0.927033	0.931733	0.940411	0.302008	0.920146	0.879433	0.934965	0.911066	0.912801	0.804238	0.927801	0.92838	0.867199	0.918715	0.866444	0.807812	0.725014	0.865884	0.73245	0.847843
ρ_{11}	0.5	0.378541	0.340338	0.043032	0.071725	0.117688	0.031694	0.078712	0.05132	0.034827	0.182679	0.123044	0.062091	0.926434	0.923184	0.498959	0.536713	0.502631	0.515517	0.102144	0.104869	0.495215	0.180966	0.260532	0.117058	0.258236	0.142282
ρ_{12}	0.5	0.485945	0.251594	0.941266	0.897235	0.944623	0.94652	0.930785	0.929322	0.945343	0.307738	0.94629	0.919101	0.952025	0.949451	0.914643	0.808977	0.929247	0.929444	0.876682	0.904334	0.860334	0.789279	0.736558	0.849138	0.704772	0.806699
ρ_{13}	0.5	0.474642	0.255139	0.897055	0.878503	0.915615	0.910014	0.897019	0.903838	0.916055	0.297695	0.892465	0.86451	0.895808	0.886061	0.885953	0.786838	0.901371	0.899906	0.848362	0.885079	0.842735	0.787796	0.716305	0.840158	0.713535	0.820562
ρ_{14}	0.5	0.479699	0.26277	0.909697	0.876891	0.913888	0.91721	0.902959	0.902119	0.915587	0.288623	0.906636	0.885518	0.905722	0.910036	0.888333	0.787922	0.903135	0.904096	0.849718	0.883688	0.844981	0.783575	0.712521	0.837286	0.706367	0.809995
ρ_{15}	0.5	0.522977	0.601639	0.494032	0.601909	0.549035	0.543409	0.548656	0.543644	0.51202	0.506753	0.519652	0.431969	0.545455	0.526056	0.527404	0.542105	0.584646	0.529746	0.473454	0.580949	0.545037	0.531859	0.515432	0.526455	0.480769	0.50484
ρ_{16}	0.5	0.483086	0.293072	0.906128	0.884637	0.948642	0.948995	0.974656	0.968575	0.953331	0.468376	0.964375	0.896322	0.954006	0.966279	0.89075	0.799276	0.970094	0.971273	0.125258	0.081725	0.12406	0.196345	0.269455	0.132166	0.713151	0.15341
ρ_{17}	0.5	0.445727	0.443946	0.452243	0.385107	0.2546	0.450916	0.797222	0.812584	0.826481	0.271846	0.696834	0.601491	0.790959	0.782401	0.538675	0.443557	0.533918	0.785161	0.648996	0.266133	0.78379	0.519199	0.667136	0.234687	0.429487	0.405191
ρ_{18}	0.5	0.416752	0.313804	0.919996	0.895351	0.856963	0.919728	0.864568	0.894413	0.905511	0.171383	0.793187	0.885061	0.669764	0.901004	0.35634	0.493597	0.367069	0.414642	0.84985	0.864689	0.370307	0.779623	0.708778	0.856269	0.723372	0.824618
ρ_{19}	0.5	0.459763	0.72706	0.047833	0.078714	0.0364	0.038105	0.048078	0.056032	0.040094	0.683989	0.04881	0.071788	0.036845	0.043855	0.073723	0.184512	0.062427	0.072139	0.106599	0.07073	0.123647	0.187519	0.26231	0.122778	0.261569	0.149961
ρ_{20}	0.5	0.975	0.960958	0.054232	0.066256	0.059073	0.037588	0.034908	0.032283	0.01774	0.166795	0.023138	0.074282	0.010861	0.011113	0.07083	0.199153	0.044197	0.040209	0.629912	0.072656	0.091155	0.714409	0.515882	0.14992	0.250187	0.219864
ρ_{21}	0.5	0.4543	0.43427	0.498211	0.445202	0.271226	0.492202	0.799915	0.819169	0.842396	0.245373	0.741574	0.653142	0.82031	0.827677	0.452803	0.456079	0.452742	0.824332	0.678542	0.283464	0.781498	0.514895	0.677039	0.24711	0.448093	0.453255
ρ_{22}	0.5	0.473734	0.439051	0.474526	0.515073	0.473936	0.485091	0.447603	0.352296	0.44869	0.344515	0.456904	0.510901	0.486936	0.464178	0.491413	0.479448	0.528504	0.483128	0.500463	0.477882	0.487489	0.512366	0.508413	0.503399	0.484807	0.503034
ρ_{23}	0.5	0.411654	0.487201	0.323435	0.43876	0.40883	0.37152	0.43799	0.413312	0.437522	0.439114	0.870843	0.731522	0.924838	0.920607	0.32984	0.689477	0.253047	0.887351	0.376595	0.390243	0.237521	0.485583	0.308807	0.296685	0.387175	0.364247
ρ_{24}	0.5	0.975	0.908204	0.062394	0.67192	0.016905	0.151157	0.231721	0.855696	0.073118	0.166868	0.100161	0.109265	0.070049	0.132602	0.95	0.196877	0.966667	0.105594	0.097312	0.825452	0.937223	0.267392	0.250212	0.100179	0.75	0.260879
ρ_{25}	0.5	0.466163	0.559881	0.562088	0.36406	0.493473	0.498436	0.451492	0.436009	0.503441	0.445513	0.536286	0.526992	0.541973	0.49624	0.610199	0.635064	0.593579	0.642268	0.462687	0.373767	0.581249	0.317623	0.450723	0.311438	0.297552	0.21576
ρ_{26}	0.5	0.461271	0.347625	0.797924	0.789527	0.814115	0.808701	0.78901	0.829763	0.813644	0.376549	0.797913	0.790238	0.777733	0.796485	0.694956	0.759629	0.675772	0.762144	0.766699	0.786106	0.645078	0.724705	0.632069	0.737634	0.665348	0.747627
ρ_{27}	0.5	0.551442	0.251941	0.918836	0.894959	0.936526	0.91939	0.8998	0.894565	0.90496	0.285794	0.907783	0.88486	0.906146	0.901379	0.895892	0.794543	0.904684	0.886682	0.849694	0.905783	0.838959	0.779968	0.700341	0.855647	0.720903	0.82315
ρ_{28}	0.5	0.479457	0.274328	0.803024	0.795394	0.822649	0.815594	0.804231	0.81358	0.812661	0.80695	0.791386	0.865174	0.803878	0.812002	0.736652	0.81608	0.807179	0.769596	0.7993308	0.74379	0.721923	0.660459	0.757859	0.670263	0.755658

Tabla 6.4: Registro de comportamiento obtenido para el Cajón `II` - PA/CMC, con $\lambda = 0.60$.

$\boldsymbol{\rho}$	$\boldsymbol{\rho}_0$	$\boldsymbol{\rho}_1$	$\boldsymbol{\rho}_2$	$\boldsymbol{\rho}_3$	$\boldsymbol{\rho}_4$	$\boldsymbol{\rho}_5$	$\boldsymbol{\rho}_6$	$\boldsymbol{\rho}_7$	$\boldsymbol{\rho}_8$	$\boldsymbol{\rho}_9$	$\boldsymbol{\rho}_{10}$	$\boldsymbol{\rho}_{11}$	$\boldsymbol{\rho}_{12}$	$\boldsymbol{\rho}_{13}$	$\boldsymbol{\rho}_{14}$
N	1	63	372	403	222	580	309	151	18	722	44	170	11	14	3
ρ_1	0.5	0.465723	0.090995	0.394472	0.362573	0.369487	0.363477	0.343809	0.692365	0.343467	0.42869	0.391325	0.497605	0.542808	0.534759
ρ_2	0.5	0.992055	0.040606	0.972162	0.01296	0.997679	0.039075	0.050421	0.029207	0.99881	0.031411	0.995167	0.651456	0.808225	0.662701
ρ_3	0.5	0.861402	0.046386	0.107474	0.077291	0.018954	0.119684	0.881387	0.043627	0.023742	0.852221	0.868604	0.498799	0.656105	0.170834
ρ_4	0.5	0.462151	0.15135	0.486589	0.239306	0.735484	0.408308	0.422672	0.686359	0.822649	0.306668	0.438206	0.427961	0.526437	0.526803
ρ_5	0.5	0.460126	0.152753	0.393909	0.341343	0.414766	0.379373	0.465542	0.560104	0.390378	0.452288	0.46041	0.470502	0.51269	0.540066
ρ_6	0.5	0.477964	0.150817	0.351028	0.460533	0.379058	0.237107	0.28336	0.326561	0.396321	0.348272	0.46843	0.533425	0.497844	0.515334
ρ_7	0.5	0.475986	0.140594	0.395075	0.400645	0.399897	0.447587	0.465363	0.414712	0.415961	0.469351	0.461031	0.506159	0.523551	0.529304
ρ_8	0.5	0.485485	0.144789	0.443465	0.443684	0.442308	0.428553	0.332946	0.570698	0.424464	0.504982	0.377802	0.545437	0.543141	0.518519
ρ_9	0.5	0.748192	0.035767	0.036973	0.182051	0.997226	0.027773	0.020437	0.049565	0.286436	0.187565	0.049051	0.923319	0.069367	0.62596
ρ_{10}	0.5	0.373378	0.103755	0.396311	0.385447	0.42234	0.419241	0.346146	0.670736	0.412033	0.373486	0.363601	0.480472	0.515567	0.517544
ρ_{11}	0.5	0.413236	0.108465	0.443125	0.49686	0.467669	0.438784	0.493795	0.703158	0.456132	0.468856	0.463171	0.562998	0.556452	0.558824
ρ_{12}	0.5	0.237698	0.105998	0.06967	0.067268	0.073624	0.062285	0.061706	0.782622	0.063638	0.125544	0.080494	0.520713	0.48234	0.528571

Tabla 6.5: Registro de comportamiento para el Cajón III - PA/CMC, con $\lambda = 0.60$.

ρ	ρ_0	ρ_1	ρ_2	ρ_3	ρ_4	ρ_5	ρ_6	ρ_7	ρ_8	ρ_9	ρ_{10}	ρ_{11}	ρ_{12}	ρ_{13}	ρ_{14}	ρ_{15}	ρ_{16}	ρ_{17}	ρ_{18}	ρ_{19}	ρ_{20}	ρ_{21}	ρ_{22}	ρ_{23}	ρ_{24}	ρ_{25}	ρ_{26}	ρ_{27}	ρ_{28}	ρ_{29}	ρ_{30}	ρ_{31}	ρ_{32}	ρ_{33}	ρ_{34}	ρ_{35}	ρ_{36}	ρ_{37}	ρ_{38}	ρ_{39}	ρ_{40}	ρ_{41}	ρ_{42}	ρ_{43}
N	1.000	32.000	196.000	199.000	52.000	66.000	4.000	26.000	63.000	80.000	194.000	9.000	2.000	23.000	19.000	10.000	11.000	4.000	11.000	10.000	88.000	10.000	249.000	20.000	5.000	4.000	3.000	11.000	46.000	8.000	2.000	12.000	4.000	5.000	2.000	9.000	2.000	13.000	3.000	2.000	12.000	8.000	3.000	2.000
ρ_1	0.500	0.905	0.894	0.840	0.016	0.485	0.875	0.301	0.008	0.006	0.836	0.944	0.251	0.978	0.294	0.865	0.046	0.307	0.046	0.700	0.889	0.861	0.881	0.885	0.841	0.734	0.806	0.651	0.689	0.589	0.681	0.487	0.125	0.887	0.652	0.893	0.750	0.758	0.893	0.250	0.042	0.929	0.827	0.252
ρ_2	0.500	0.821	0.800	0.922	0.331	0.336	0.440	0.411	0.536	0.669	0.926	0.361	0.419	0.203	0.492	0.299	0.367	0.281	0.769	0.888	0.924	0.886	0.913	0.895	0.685	0.811	0.401	0.397	0.359	0.294	0.449	0.341	0.832	0.327	0.310	0.276	0.367	0.412	0.292	0.322	0.458	0.883	0.765	0.397
ρ_3	0.500	0.800	0.894	0.895	0.831	0.947	0.361	0.392	0.516	0.668	0.899	0.882	0.420	0.260	0.420	0.308	0.348	0.292	0.702	0.890	0.901	0.873	0.887	0.865	0.882	0.794	0.402	0.324	0.336	0.279	0.432	0.333	0.818	0.364	0.310	0.286	0.405	0.328	0.334	0.338	0.379	0.855	0.762	0.397
ρ_4	0.500	0.894	0.882	0.000	0.041	0.126	0.762	0.061	0.009	0.987	0.712	0.662	0.404	0.115	0.497	0.218	0.650	0.192	0.941	0.985	0.941	0.989	0.966	0.894	0.868	0.564	0.401	0.447	0.064	0.750	0.475	0.793	0.227	0.146	0.475	0.186	0.251	0.040	0.716	0.347	0.123	0.929	0.827	0.251
ρ_5	0.500	0.548	0.606	0.589	0.549	0.579	0.488	0.660	0.619	0.519	0.592	0.631	0.528	0.486	0.602	0.642	0.583	0.334	0.620	0.676	0.683	0.663	0.673	0.665	0.599	0.594	0.545	0.611	0.633	0.573	0.274	0.608	0.502	0.600	0.519	0.634	0.611	0.622	0.634	0.572	0.591	0.621	0.621	0.520
ρ_6	0.500	0.935	0.922	0.940	0.111	0.055	0.875	0.098	0.064	0.086	0.940	0.530	0.251	0.123	0.404	0.353	0.051	0.852	0.664	0.899	0.965	0.950	0.002	0.025	0.101	0.125	0.168	0.143	0.289	0.064	0.750	0.175	0.875	0.781	0.584	0.088	0.750	0.940	0.237	0.687	0.880	0.884	0.167	0.464
ρ_7	0.500	0.900	0.901	0.905	0.154	0.105	0.757	0.082	0.257	0.069	0.906	0.112	0.558	0.064	0.092	0.426	0.679	0.484	0.993	0.946	0.990	0.946	0.994	0.971	0.814	0.872	0.327	0.370	0.024	0.779	0.750	0.443	0.750	0.818	0.690	0.566	0.251	0.771	0.711	0.250	0.128	0.907	0.830	0.257
ρ_8	0.500	0.804	0.862	0.594	0.093	0.092	0.329	0.115	0.881	0.910	0.598	0.814	0.750	0.102	0.657	0.310	0.940	0.552	0.740	0.684	0.951	0.950	0.996	0.925	0.899	0.875	0.813	0.047	0.300	0.064	0.252	0.043	0.717	0.822	0.750	0.087	0.252	0.231	0.168	0.614	0.872	0.765	0.658	0.730
ρ_9	0.500	0.454	0.933	0.454	0.468	0.421	0.463	0.333	0.457	0.454	0.429	0.451	0.514	0.571	0.429	0.168	0.457	0.507	0.482	0.301	0.487	0.449	0.411	0.317	0.443	0.293	0.325	0.338	0.427	0.493	0.569	0.426	0.519	0.365	0.520	0.534	0.314	0.398	0.267	0.329	0.304	0.482	0.431	0.520
ρ_{10}	0.500	0.366	0.383	0.268	0.310	0.269	0.531	0.223	0.438	0.193	0.279	0.514	0.534	0.242	0.392	0.364	0.457	0.530	0.420	0.259	0.333	0.309	0.300	0.317	0.443	0.293	0.325	0.338	0.504	0.507	0.496	0.519	0.353	0.385	0.520	0.534	0.314	0.398	0.267	0.329	0.304	0.482	0.431	0.520
ρ_{11}	0.500	0.977	0.045	0.009	0.187	0.097	0.759	0.098	0.915	0.994	0.996	0.143	0.750	0.264	0.049	0.858	0.897	0.131	0.936	0.089	0.949	0.089	0.046	0.066	0.127	0.158	0.767	0.931	0.133	0.832	0.251	0.065	0.126	0.317	0.252	0.098	0.251	0.467	0.254	0.250	0.497	0.063	0.196	0.750
ρ_{12}	0.500	0.867	0.942	0.963	0.934	0.398	0.539	0.359	0.955	0.966	0.966	0.554	0.538	0.414	0.893	0.522	0.740	0.545	0.931	0.079	0.028	0.071	0.024	0.046	0.306	0.536	0.545	0.727	0.611	0.601	0.593	0.658	0.855	0.464	0.519	0.541	0.427	0.700	0.409	0.737	0.957	0.915	0.182	0.521
ρ_{13}	0.500	0.513	0.140	0.920	0.536	0.530	0.676	0.468	0.562	0.506	0.927	0.667	0.538	0.571	0.309	0.536	0.525	0.529	0.602	0.863	0.888	0.863	0.119	0.138	0.283	0.204	0.587	0.340	0.597	0.599	0.306	0.669	0.559	0.517	0.514	0.662	0.517	0.396	0.514	0.493	0.359	0.571	0.769	0.590
ρ_{14}	0.500	0.894	0.965	0.008	0.945	0.102	0.875	0.108	0.972	0.988	0.068	0.753	0.717	0.224	0.827	0.175	0.909	0.127	0.945	0.056	0.009	0.057	0.008	0.028	0.892	0.866	0.688	0.404	0.049	0.115	0.750	0.510	0.866	0.141	0.251	0.136	0.619	0.857	0.652	0.743	0.952	0.927	0.171	0.251
ρ_{15}	0.500	0.400	0.338	0.351	0.478	0.372	0.433	0.383	0.411	0.304	0.371	0.466	0.508	0.484	0.600	0.594	0.575	0.536	0.369	0.280	0.276	0.301	0.283	0.297	0.395	0.360	0.492	0.029	0.447	0.525	0.496	0.406	0.351	0.566	0.521	0.427	0.556	0.617	0.575	0.577	0.639	0.894	0.348	0.521
ρ_{16}	0.500	0.427	0.428	0.419	0.373	0.388	0.502	0.377	0.409	0.389	0.410	0.506	0.515	0.396	0.363	0.425	0.445	0.507	0.471	0.495	0.479	0.485	0.460	0.459	0.473	0.451	0.533	0.361	0.449	0.491	0.506	0.467	0.452	0.451	0.514	0.487	0.421	0.356	0.400	0.420	0.312	0.440	0.481	0.493
ρ_{17}	0.500	0.460	0.487	0.507	0.331	0.326	0.419	0.319	0.401	0.396	0.511	0.407	0.539	0.459	0.270	0.406	0.398	0.573	0.419	0.511	0.497	0.495	0.498	0.498	0.590	0.495	0.524	0.238	0.294	0.271	0.484	0.327	0.472	0.557	0.782	0.406	0.558	0.291	0.550	0.529	0.291	0.479	0.514	0.365
ρ_{18}	0.500	0.175	0.081	0.087	0.107	0.094	0.175	0.085	0.106	0.072	0.086	0.223	0.514	0.161	0.175	0.210	0.295	0.524	0.114	0.098	0.069	0.106	0.069	0.089	0.249	0.178	0.362	0.228	0.187	0.373	0.501	0.147	0.184	0.340	0.513	0.219	0.286	0.215	0.215	0.289	0.151	0.135	0.211	0.320

Tabla 6.6: Registro de comportamiento para el Cajón `IV` - PA/CMC, con $\lambda = 0.60$.

$\boldsymbol{\rho}$	$\boldsymbol{\rho}_0$	$\boldsymbol{\rho}_1$	$\boldsymbol{\rho}_2$	$\boldsymbol{\rho}_3$	$\boldsymbol{\rho}_4$	$\boldsymbol{\rho}_5$	$\boldsymbol{\rho}_6$	$\boldsymbol{\rho}_7$	$\boldsymbol{\rho}_8$	$\boldsymbol{\rho}_9$	$\boldsymbol{\rho}_{10}$	$\boldsymbol{\rho}_{11}$	$\boldsymbol{\rho}_{12}$	$\boldsymbol{\rho}_{13}$	$\boldsymbol{\rho}_{14}$
N	1	43	22	108	32	37	126	31	5	256	14	35	23	55	24
ρ_1	0.5	0.483349	0.471529	0.50784	0.836763	0.852117	0.866945	0.859585	0.796332	0.548037	0.519137	0.504886	0.85023	0.644647	0.563897
ρ_2	0.5	0.750658	0.76352	0.810781	0.687824	0.779815	0.681707	0.660245	0.617075	0.817849	0.838916	0.807313	0.870194	0.756939	0.700221
ρ_3	0.5	0.211709	0.226388	0.073487	0.906142	0.909452	0.50729	0.067826	0.144985	0.092805	0.099457	0.092255	0.931216	0.876866	0.880586
ρ_4	0.5	0.408941	0.397977	0.383486	0.833895	0.858484	0.85568	0.848662	0.790591	0.465433	0.287536	0.403842	0.868561	0.650452	0.585918
ρ_5	0.5	0.505657	0.517481	0.548256	0.476461	0.288655	0.911731	0.88183	0.815504	0.630251	0.570996	0.591683	0.12196	0.795515	0.778326
ρ_6	0.5	0.473447	0.422909	0.601125	0.095486	0.060133	0.107309	0.11564	0.184157	0.337129	0.189509	0.439158	0.054017	0.278622	0.346732
ρ_7	0.5	0.509339	0.435307	0.889546	0.04363	0.024246	0.015998	0.920592	0.849458	0.661226	0.902098	0.754702	0.030844	0.026857	0.043343
ρ_8	0.5	0.505219	0.427459	0.892352	0.04523	0.034782	0.023574	0.921006	0.849677	0.661999	0.902363	0.760132	0.048388	0.036313	0.047952
ρ_9	0.5	0.504651	0.516384	0.540945	0.483252	0.335201	0.908141	0.873293	0.808409	0.62371	0.559035	0.585377	0.215236	0.791338	0.780653
ρ_{10}	0.5	0.554307	0.554474	0.587736	0.808983	0.793777	0.861519	0.859502	0.801188	0.654149	0.639659	0.657503	0.77123	0.675908	0.633666
ρ_{11}	0.5	0.902025	0.169706	0.206239	0.05927	0.364482	0.1537	0.074709	0.521225	0.039418	0.317667	0.317955	0.104833	0.090607	0.663197
ρ_{12}	0.5	0.903048	0.142375	0.159922	0.084381	0.357348	0.152827	0.077913	0.540051	0.04058	0.282902	0.304144	0.100409	0.083631	0.622277
ρ_{13}	0.5	0.506539	0.464644	0.522853	0.472078	0.645598	0.532226	0.498901	0.46831	0.552571	0.464619	0.503504	0.820176	0.623183	0.54446
ρ_{14}	0.5	0.452954	0.435011	0.435964	0.254408	0.237441	0.179536	0.180943	0.236086	0.393205	0.436372	0.398836	0.2097	0.377459	0.45533
ρ_{15}	0.5	0.527254	0.573413	0.516408	0.562483	0.515316	0.512371	0.46252	0.46469	0.528649	0.510413	0.537015	0.410388	0.620409	0.661459
ρ_{16}	0.5	0.490945	0.5066	0.491666	0.075082	0.076006	0.067464	0.072091	0.143059	0.431474	0.437764	0.455515	0.100655	0.389451	0.520881
ρ_{17}	0.5	0.577761	0.54544	0.554804	0.699091	0.545982	0.667778	0.693587	0.688335	0.536189	0.456799	0.493531	0.383521	0.57626	0.644494
ρ_{18}	0.5	0.23826	0.225361	0.296265	0.418348	0.382204	0.399425	0.462396	0.478282	0.253263	0.367314	0.263614	0.338694	0.21079	0.171642
ρ_{19}	0.5	0.532231	0.513929	0.504107	0.246924	0.264657	0.518874	0.563375	0.570297	0.501963	0.459637	0.541727	0.32363	0.450605	0.508781

Tabla 6.7: Registro de comportamiento para el Cajón I - CML/CMR, con $\lambda = 0.53$.

ρ	ρ_0	ρ_1	ρ_2	ρ_3	ρ_4	ρ_5	ρ_6	ρ_7	ρ_8	ρ_9	ρ_{10}	ρ_{11}	ρ_{12}	ρ_{13}	ρ_{14}	ρ_{15}	ρ_{16}	ρ_{17}	ρ_{18}	ρ_{19}	ρ_{20}	ρ_{21}	ρ_{22}	ρ_{23}
N	1	4	3	2	2	11	19	2	2	3	109	39	12	88	79	625	4	364	155	2	2	79	2	2
ρ_1	0.5	0.535283	0.566171	0.621984	0.621984	0.669218	0.611557	0.621984	0.621984	0.663177	0.667253	0.706905	0.483258	0.682593	0.721926	0.727918	0.538552	0.716166	0.70289	0.608603	0.486364	0.679468	0.479508	0.46875
ρ_2	0.5	0.875	0.249813	0.75	0.254359	0.116268	0.222023	0.551009	0.75	0.173729	0.178665	0.087961	0.223848	0.089701	0.032162	0.059025	0.494886	0.037748	0.067701	0.75	0.498377	0.224183	0.75	0.250697
ρ_3	0.5	0.557388	0.555163	0.614606	0.614388	0.657289	0.596447	0.616006	0.616006	0.655	0.694599	0.70802	0.488421	0.71669	0.721634	0.725849	0.527599	0.728863	0.714678	0.578429	0.486842	0.674342	0.482759	0.46875
ρ_4	0.5	0.875	0.675902	0.75	0.75	0.099647	0.350208	0.25707	0.75	0.615719	0.203458	0.057742	0.103339	0.032133	0.026008	0.055783	0.826427	0.037485	0.044124	0.75	0.75	0.075106	0.75	0.614234
ρ_5	0.5	0.560245	0.567962	0.613695	0.613481	0.645827	0.586657	0.613481	0.613481	0.651784	0.688602	0.711788	0.492759	0.721067	0.71985	0.723642	0.539739	0.725	0.718044	0.592903	0.495614	0.667107	0.47807	0.492188
ρ_6	0.5	0.546371	0.579761	0.617627	0.619005	0.591264	0.533945	0.619595	0.619595	0.660233	0.687802	0.734339	0.507477	0.55573	0.656819	0.557946	0.502623	0.557677	0.555948	0.608904	0.491071	0.549138	0.477273	0.496154
ρ_7	0.5	0.875	0.796963	0.749996	0.75	0.888981	0.128193	0.749993	0.749992	0.833213	0.598452	0.979625	0.518118	0.007263	0.123731	0.001409	0.242693	0.071305	0.846041	0.75	0.717363	0.038415	0.75	0.251227
ρ_8	0.5	0.530109	0.572351	0.61588	0.615665	0.672776	0.614711	0.61588	0.61588	0.65465	0.70049	0.732621	0.5063	0.726388	0.733704	0.73146	0.550824	0.731054	0.727488	0.60543	0.504098	0.689045	0.487705	0.5
ρ_9	0.5	0.875	0.179454	0.266863	0.268627	0.05659	0.143213	0.75	0.75	0.171539	0.224055	0.08584	0.829065	0.105082	0.054319	0.111709	0.127407	0.079883	0.117005	0.75	0.252627	0.103501	0.72434	0.75
ρ_{10}	0.5	0.875	0.459711	0.450534	0.250235	0.254288	0.027013	0.300015	0.250264	0.556091	0.383785	0.186398	0.327811	0.090711	0.262944	0.221266	0.351589	0.173008	0.179497	0.251533	0.75	0.228974	0.75	0.25081
ρ_{11}	0.5	0.875	0.833333	0.75	0.719672	0.0864	0.357752	0.606706	0.250855	0.786214	0.29715	0.266389	0.246112	0.129585	0.295852	0.26279	0.366013	0.178446	0.25719	0.250674	0.75	0.185342	0.251034	0.666719
ρ_{12}	0.5	0.875	0.682597	0.75	0.75	0.36728	0.26728	0.749994	0.749995	0.839075	0.993431	0.987027	0.453841	0.008898	0.980582	0.001293	0.125527	0.001438	0.003284	0.349361	0.251545	0.017121	0.373673	0.471605
ρ_{13}	0.5	0.875	0.444277	0.75	0.75	0.182128	0.27579	0.610186	0.250375	0.489933	0.262053	0.266179	0.419761	0.365273	0.081267	0.087204	0.456295	0.11485	0.24559	0.250755	0.25171	0.134398	0.75	0.250419
ρ_{14}	0.5	0.555163	0.570487	0.620659	0.620659	0.668362	0.613448	0.622021	0.622021	0.662695	0.626435	0.725607	0.489144	0.623798	0.816691	0.805111	0.54664	0.75041	0.717007	0.628697	0.487288	0.655996	0.474138	0.472222
ρ_{15}	0.5	0.559033	0.572868	0.622234	0.62234	0.539324	0.428978	0.62234	0.622567	0.663422	0.697303	0.582861	0.508954	0.752942	0.493417	0.588768	0.549745	0.571123	0.585381	0.395974	0.486842	0.525123	0.495763	0.491935
ρ_{16}	0.5	0.875	0.269825	0.250215	0.250215	0.646182	0.186328	0.749964	0.250019	0.833145	0.714657	0.03678	0.274721	0.042728	0.030602	0.013094	0.664	0.427767	0.018216	0.75	0.75	0.230784	0.646575	0.644517
ρ_{17}	0.5	0.875	0.412326	0.250569	0.250542	0.352463	0.211378	0.673241	0.250607	0.167236	0.334848	0.261575	0.184371	0.109659	0.273726	0.314593	0.800376	0.175063	0.281209	0.250828	0.75	0.378071	0.655254	0.250843

Tabla 6.8: Registro de comportamiento para el Cajón II - CML/CMR, con $\lambda = 0.60$.

ρ	ρ_0	ρ_1	ρ_2	ρ_3	ρ_4	ρ_5	ρ_6	ρ_7	ρ_8	ρ_9	ρ_{10}	ρ_{11}	ρ_{12}	ρ_{13}	ρ_{14}	ρ_{15}	ρ_{16}	ρ_{17}	ρ_{18}
N	1	173	218	536	140	390	34	184	13	88	23	508	18	4	267	2	2	2	3
ρ_1	0.5	0.752896	0.782055	0.725291	0.743884	0.428552	0.375879	0.521296	0.361271	0.941152	0.421277	0.365995	0.481334	0.325651	0.372827	0.370984	0.691851	0.508621	0.365476
ρ_2	0.5	0.886285	0.930773	0.923113	0.914461	0.943934	0.512243	0.97654	0.556476	0.902831	0.540743	0.965849	0.528469	0.176917	0.970706	0.333399	0.702605	0.560345	0.525884
ρ_3	0.5	0.622107	0.779504	0.772477	0.764247	0.550856	0.484964	0.886373	0.497127	0.889799	0.471007	0.834167	0.43635	0.29711	0.808386	0.314759	0.35	0.466667	0.443768
ρ_4	0.5	0.585205	0.721587	0.4823	0.557741	0.764598	0.53084	0.629077	0.509977	0.807374	0.516554	0.741735	0.534338	0.545639	0.671545	0.656738	0.543651	0.5	0.534483
ρ_5	0.5	0.901009	0.933997	0.85764	0.873571	0.928649	0.549727	0.319561	0.503384	0.509514	0.520745	0.67896	0.579905	0.541009	0.715415	0.519824	0.396968	0.535714	0.548535
ρ_6	0.5	0.98796	0.006974	0.05353	0.945339	0.01273	0.13301	0.32556	0.863019	0.472376	0.061916	0.054287	0.889195	0.127528	0.781782	0.254203	0.252737	0.577076	0.554752
ρ_7	0.5	0.649355	0.642462	0.650335	0.662806	0.619853	0.483645	0.666184	0.527761	0.675373	0.491977	0.679245	0.500192	0.647608	0.647564	0.549721	0.586778	0.524194	0.512821
ρ_8	0.5	0.520338	0.518754	0.477168	0.488529	0.491632	0.56564	0.658924	0.525583	0.577753	0.543557	0.423521	0.492274	0.486929	0.517889	0.405113	0.558576	0.519231	0.544147
ρ_9	0.5	0.932695	0.970869	0.073847	0.083619	0.917065	0.052129	0.097558	0.144522	0.140491	0.799196	0.146138	0.915384	0.126931	0.506875	0.702219	0.75	0.480207	0.568489
ρ_{10}	0.5	0.57238	0.679724	0.435269	0.439622	0.469071	0.55865	0.888688	0.419266	0.646297	0.524279	0.295564	0.432344	0.268733	0.228669	0.318705	0.720317	0.506757	0.522222
ρ_{11}	0.5	0.85757	0.894723	0.9184	0.903449	0.899867	0.201644	0.931975	0.193441	0.931133	0.248938	0.928775	0.369025	0.715927	0.931233	0.657065	0.655294	0.55	0.217024
ρ_{12}	0.5	0.590726	0.813779	0.625561	0.57164	0.523889	0.539338	0.802367	0.48638	0.614925	0.495857	0.331366	0.449498	0.541335	0.358545	0.292816	0.715983	0.546875	0.519608
ρ_{13}	0.5	0.289019	0.147376	0.268159	0.27179	0.451707	0.517052	0.276442	0.59227	0.313981	0.534359	0.572441	0.573243	0.646761	0.559841	0.722803	0.403553	0.516667	0.571429
ρ_{14}	0.5	0.501166	0.654838	0.560188	0.522564	0.225509	0.500241	0.461062	0.447856	0.484708	0.534937	0.235632	0.461998	0.343302	0.151546	0.308508	0.417628	0.516667	0.528571
ρ_{15}	0.5	0.403341	0.245093	0.294901	0.362851	0.670734	0.499137	0.41518	0.555572	0.040856	0.499387	0.608771	0.511372	0.612778	0.710368	0.594715	0.43589	0.535714	0.48716

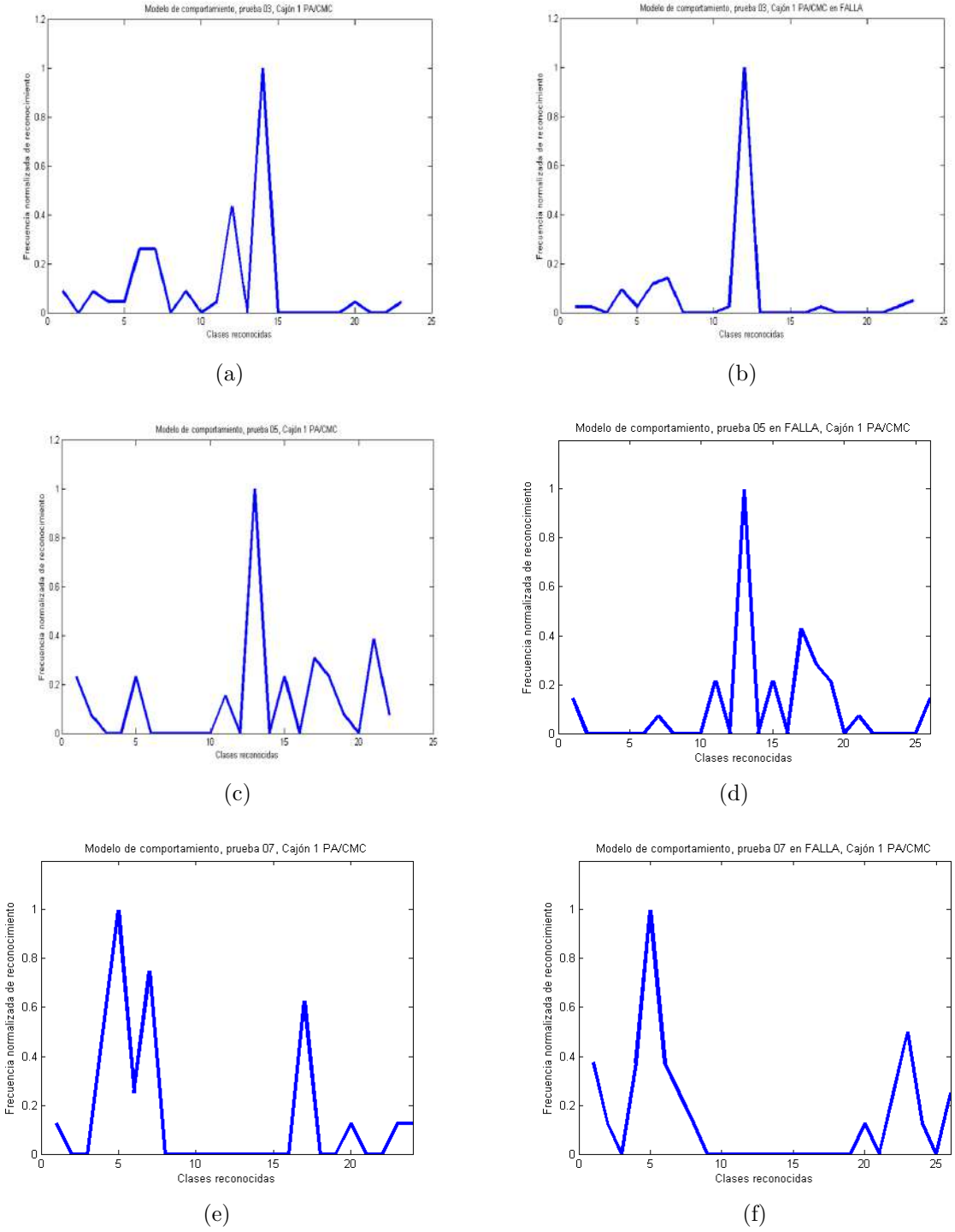


Figura 6.6: Para el Cajón I PA/CMC, prueba 3 (`tracPA-AV`): (a) Modelo de comportamiento en funcionamiento correcto, (b) modelo de comportamiento en falla. Prueba 5 (`tracCMC-AV`): (c) Modelo de comportamiento en funcionamiento correcto, (d) modelo de comportamiento en falla. Prueba 7 (`lampPNA-AV`): (e) Modelo de comportamiento en funcionamiento correcto, (f) modelo de comportamiento en falla.

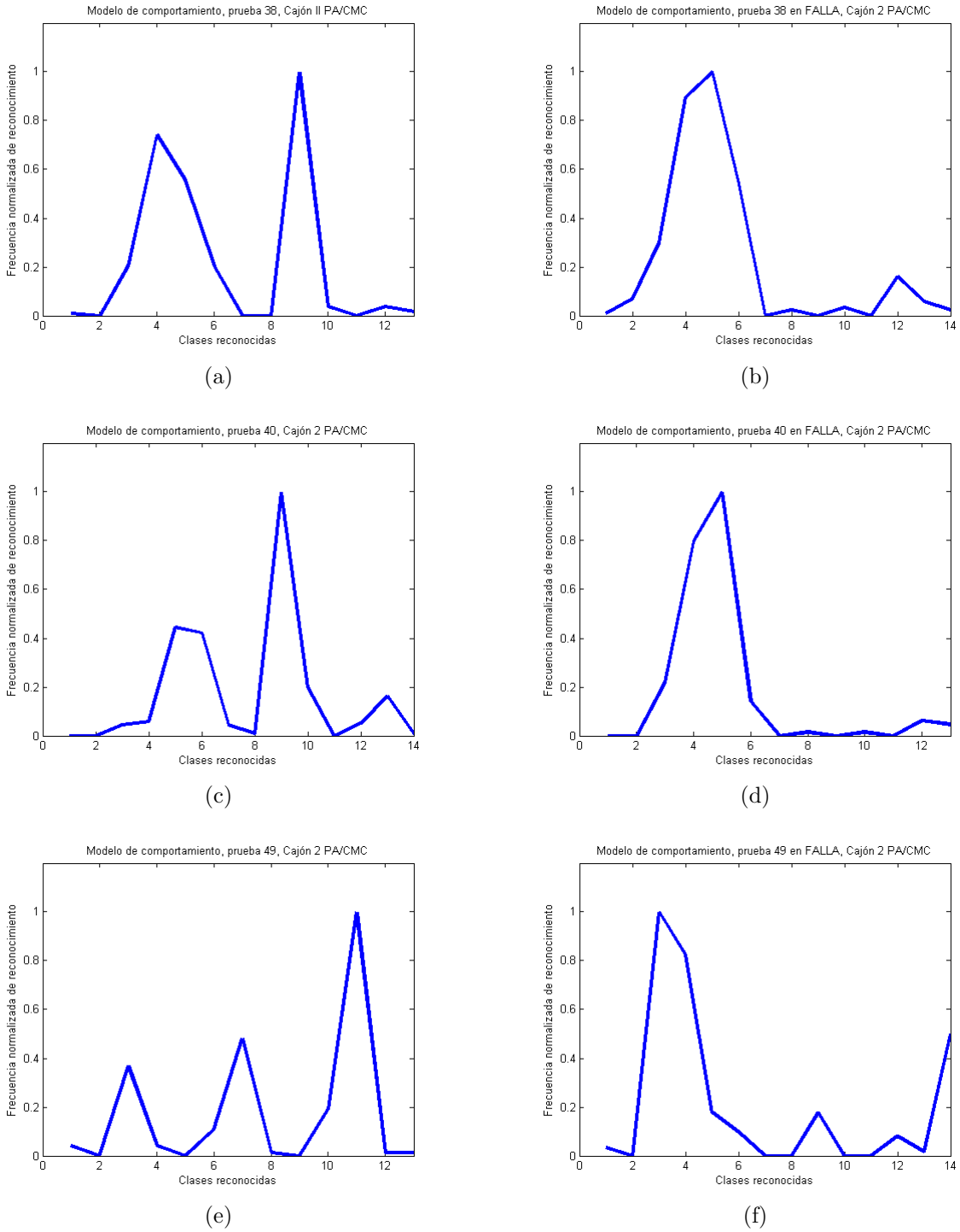


Figura 6.7: Para el Cajón II PA/CMC, prueba 38 (`umbr255AV_OD`): (a) Modelo de comportamiento en funcionamiento correcto, (b) modelo de comportamiento en falla. Prueba 40 (`umbr197AV_OD`): (c) Modelo de comportamiento en funcionamiento correcto, (d) modelo de comportamiento en falla. Prueba 49 (`CMC_AV`): (e) Modelo de comportamiento en funcionamiento correcto, (f) modelo de comportamiento en falla.

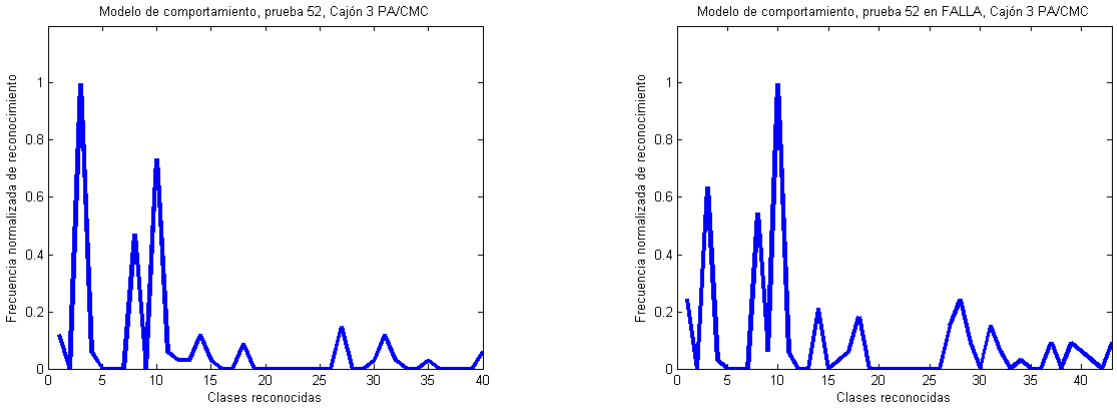
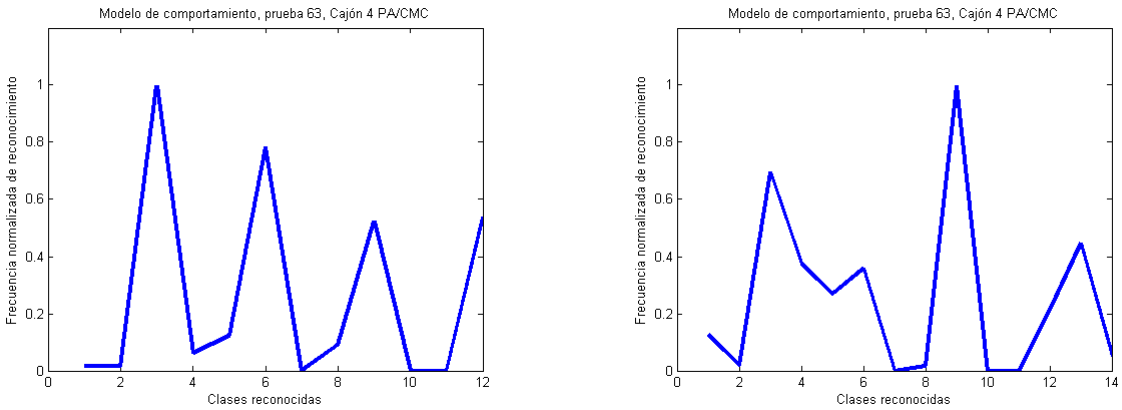


Figura 6.8: Para el Cajón III PA/CMC, prueba 52 (`iniPA-AV`): (a) Modelo de comportamiento en funcionamiento correcto, (b) modelo de comportamiento en falla. Prueba 57 (`83Q-AVOD`): (c) Modelo de comportamiento en funcionamiento correcto, (d) modelo de comportamiento en falla.



(a) (b)

Figura 6.9: Para el Cajón IV PA/CMC, prueba 63 (`captrsVal`): (a) Modelo de comportamiento en funcionamiento correcto, (b) modelo de comportamiento en falla.

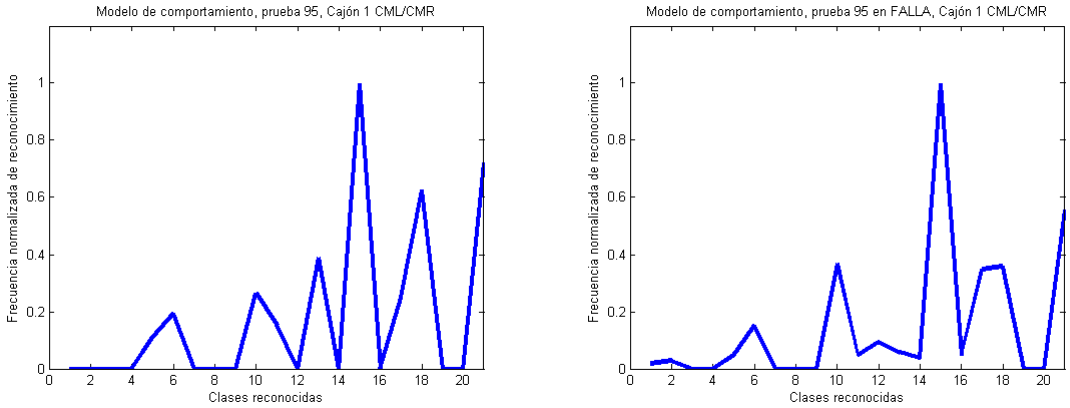


Figura 6.10: Para el Cajón I CML/CMR, prueba 95 (`CML-50-AV`): (a) Modelo de comportamiento en funcionamiento correcto, (b) modelo de comportamiento en falla. Prueba 96 (`CML-50-AR`): (a) Modelo de comportamiento en funcionamiento correcto, (b) modelo de comportamiento en falla.

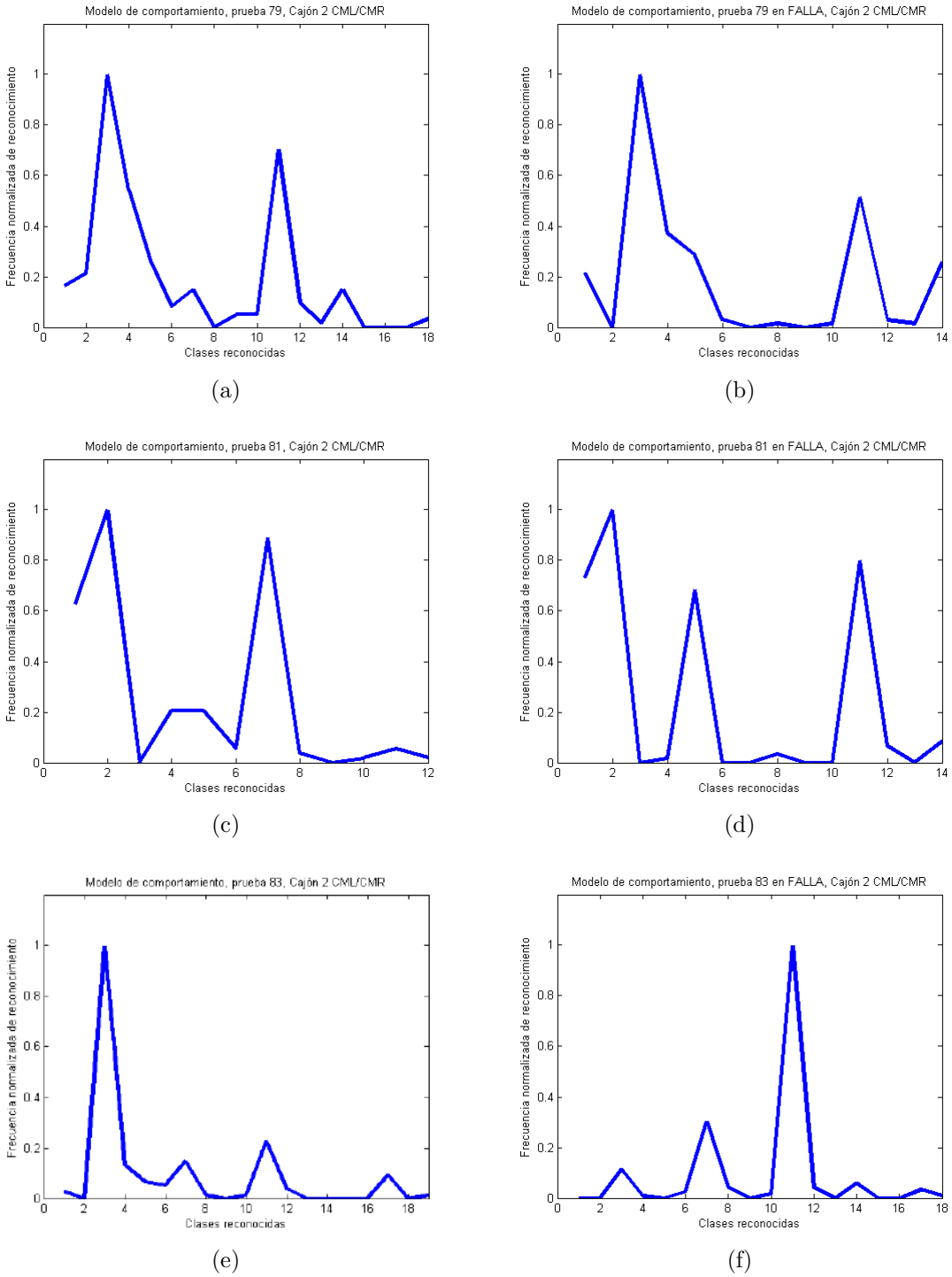


Figura 6.11: Para el Cajón II CML/CMR, prueba 79 (`CML25-AV`): (a) Modelo de comportamiento en funcionamiento correcto, (b) modelo de comportamiento en falla. Prueba 81 (`CML50-AV`): (a) Modelo de comportamiento en funcionamiento correcto, (b) modelo de comportamiento en falla. Prueba 83 (`CMR`): (a) Modelo de comportamiento en funcionamiento correcto, (b) modelo de comportamiento en falla.

Capítulo 7

Conclusiones y trabajo futuro

En este capítulo se presentan las conclusiones que este trabajo de investigación ha proporcionado, así como los trabajos futuros que se tiene estimado que podrán realizarse a partir del mismo.

Conclusiones

1. El software, desarrollado, llamado *Núcleo Chopper del SPAT 135 (NC-SPAT 135)*, es totalmente funcional para diagnosticar por separado el funcionamiento de los seis cajones del sistema electrónico de Pilotaje Automático PA 135 Chopper (Material NM-79 Chopper).

2. El estudio del estado del arte del diagnóstico automático de fallas basado en técnicas de reconocimiento de patrones, ha proporcionado las referencias necesarias para considerar que los dos algoritmos analizados en este trabajo, pueden ser tomados en cuenta para diagnosticar el sistema electrónico de Piloto Automático del metro.

3. Los estudios realizados en el marco teórico, han brindado la oportunidad de conocer a un nivel más allá del esperado por un lado a las RNAs y por otro a LAMDA. RNAs cuenta con amplia literatura y aplicaciones. Mucha de esta literatura aborda lo esencial, que en algunos casos no es suficiente. Se ha profundizado y se ha alcanzado conocimiento invaluable sobre dicho algoritmo. En cuanto a LAMDA, no se cuenta con mucha literatura que profundice en cada aspecto de su estructura; sin embargo, ahora se cuenta, sin lugar a dudas, con un conocimiento avanzado en el tema.

4. Se deja abierta a cualquier persona, una fuente de información fiable de conocimiento a través de este trabajo de investigación, en cual se abarcan aspectos generales y particulares que son muy difíciles de encontrar en la actualidad sobre LAMDA.

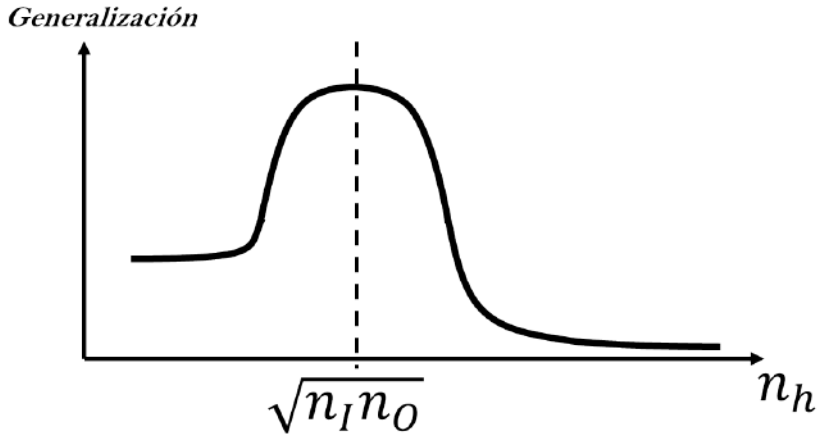


Figura 7.1: Patrón estimado de capacidad de generalización para las redes neuronales estudiadas.

5. Se ha probado la hipótesis principal, la cual estimaba que es posible diagnosticar el sistema de PA implementando un núcleo de software, basado en una técnica de reconocimiento de patrones.

6. Se ha probado la hipótesis secundaria, la cual plantea que, dados los resultados de la investigación sobre las características de LAMDA, sería idóneo para ser implementado en el producto final.

7. Al menos para la aplicación que se desarrolla en este trabajo, se ha probado que la *regla de la pirámide geométrica* es un enfoque que se aproxima en gran medida, a obtener el mayor rendimiento posible de una red neuronal.

8. Al menos para la aplicación que se estudia en este trabajo y en la mayoría de los casos estudiados, se ha determinado que una red neuronal con una capa oculta y una capa monolítica de salida, pierde plasticidad a medida que el número de neuronas en la capa oculta (n_h) se aleja de $\sqrt{n_I n_O}$. En otra minoría de casos (Sección 5.5 y Sección 5.6) el resultado no se aleja mucho del anterior, pero resulta evidente que si n_h crece más allá de este rango de valores, la red cae en un sobre ajuste cada vez peor. Se estima en general, un patrón muy similar al de la Figura 7.1.

9. Dado un conjunto de entrada modificado por ruido, se tiene que en fase de reconocimiento de patrones LAMDA es mejor al presentar en promedio un 75.83 % de reconocimiento, además fue mejor en el 66.66 % de los experimentos; sin embargo, es mucho mejor RNAs en cuanto a complejidad computacional temporal, en una proporción de 3:1.

10. En base a los resultados obtenidos de la implementación del NC-SPAT 135, los modelos de comportamiento propuestos para caracterizar pruebas de cajón, proporcionan información suficiente para realizar diagnósticos precisos de fallas.

Trabajo futuro

1. Se plantea el mejoramiento de la interfaz y las opciones con las que la aplicación desarrollada cuenta; por ejemplo: la implementación de una interfaz enfocada a la investigación, en la que se muestren todos los resultados obtenidos a lo largo del procesamiento de los datos de entrada, con capacidad de exportar dichos resultados en formatos usuales de archivos. Se propone, dentro de este mejoramiento, que los vectores mapa sean configurables tanto en dimensión como en el contenido de sus elementos, haciendo configurable el tipo de pre-procesamiento que se aplicaría a cada variable de entrada, mediante un módulo de dicha interfaz.

2. Se propone incorporar en la aplicación, módulos basados en metodologías para la selección de características[1] en tiempo real y que los resultados que éstos procedimientos generen, sean también almacenados para su análisis posterior, o como perfiles de análisis preestablecidos en experimentos posteriores.

3. Dado que se propone que la aplicación escale a un ámbito de investigación, se propone también integrar más algoritmos que puedan ser comparados en la misma herramienta, enfáticamente incluyendo aquellos que han mostrado los resultados más competitivos en un plazo no anterior a cinco años (el caso de SVMs por ejemplo). Dicha integración, para propósitos de comparación, deberá estar debidamente cotejada con los criterios de comparación que sean posibles de tomar en cuenta según las características de los algoritmos bajo estudio. Lo anterior requiere un trabajo de investigación exhaustivo de la literatura.

4. Se propone también hacer los experimentos vistos en este trabajo para RNAs, pero ahora para arquitecturas de dos capas ocultas, con objeto de verificar el comportamiento de la plasticidad a medida que crece desde cero el número de neuronas en cada capa oculta.

5. Se propone realizar un estudio completo sobre los algoritmos más competitivos que existan en la literatura, a efecto de generar una guía global de rendimiento según el contexto[2]. Lo anterior requiere establecer corpora de datos de entrada que sirvan de referencia, así como establecer también un conjunto patrón de experimentos que impliquen situaciones que abarquen la mayoría de los contextos (o por lo menos los más usuales).

[1] Algunas de las metodologías más comunes en la literatura son: *análisis discriminante, componentes principales, entropía de Shannon y transformación de Fisher*[85].

[2] Entiéndase por *contexto* el entorno y la planta (o sistema observado), con los que un algoritmo de reconocimiento de patrones en particular debe interactuar.

Apéndice A

Tabla de componentes de la GUI del NC-SPAT135

Id	Descripción	Identificador en el código	Método
1	*Selector de archivos.* Si el *modo de operación* está seleccionado como `Diagnóstico`, selecciona exclusivamente archivo de clases (`clases.csv`). Si está seleccionado como `Experto`, selecciona el archivo de clases sólo si el usuario va a entrenar en línea la aplicación. En caso de ser desde un archivo, este botón no tiene utilidad.	`PANEL_SELECT_FILE`	`fileSelectionButton()`

2	Selector del *modo de operación*. Si está seleccionado como `Experto`, el usuario avanzado podrá darle mantenimiento al software, realizando entrenamientos del algoritmo ya sea desde un archivo de senñales (`señales.csv`), o en línea. Si está seleccionado como `Diagnóstico`, el usuario en general podrá hacer diagnósticos en fase de reconocimiento de patrones (sin entrenamiento alguno).	`PANEL_MODO`	N/A
3	*Monitor de archivo cargado*. Es un campo de texto no editable que muestra la ruta del archivo que ha sido seleccionado por el *Selector de archivos* o por el botón de *Aprender Archivo*.	`PANEL_CLASES_FILE`	N/A
4	*Selector del cajón de PA*. Selecciona el cajón del PA que será objeto de pruebas.	`PANEL_CAJON`	N/A
5	*Parámetro de exigencia*. Es un selector numérico en el que se puede seleccionar el valor del parámetro de exigencia con el que ha de trabajar LAMDA. Sólo es editable en modo `Experto` y acepta valores reales exclusivamente $\in [0, 1]$.	`PANEL_LAMBDA`	N/A

6	*Botón de inicio de diagnóstico*. Es un botón conmutador que inicialmente esta suelto (no oprimido). Al hacer "click" en él (se queda oprimido), inicia el proceso de diagnóstico, activando inmediatamente a `PANEL_TIMER`, mismo que sólo se interrumpe si se vuelve a pulsar el botón, liberándolo. El timer llama a `LAMDA_ROL()`.	`PANEL_TOGGLE_DIAGNOSIS`	`onlineDiagnosisButton()`
7	*Botón de aprendizaje desde archivo*. Este es un botón de control sencillo que llama directamente a la función `LAMDA_LFF()` que se ha implementado en este trabajo. `LAMDA_LFF()` entrena a LAMDA desde un archivo de señales.	`PANEL_APRENDER_FROM_FILE`	`aprenderArchivoButton()`
8	*Monitor de estado*. Es un indicador textual del estado que la aplicación está reconociendo en tiempo real. Si no hay un estado asociado al reconocimiento de un patrón y que este sea asignado a alguna clase existente en el archivo de clases, el indicador muestra una etiqueta no personalizada (Por ejemplo: `C_1`, `C_2`, etc.). Si LAMDA determina que no hay información suficiente para clasificar algún patrón de entrada, el indicador muestra una `FALLA`.	`PANEL_COLORNUM`	N/A

9	*Botón de aprendizaje en línea*. Este botón tiene la misma funcionalidad que el *Botón de inicio de diagnóstico*; a excepción de que sólo se puede usar en modo `Experto` y que al llamar a su método, éste activa el timer el cual a su vez, llama a `LAMDA_LOL()` para ejecutar el algoritmo de aprendizaje y clasificación en línea.	`PANEL_TOGGLE_ONLINE_LEARN`	`onLineLearning()`
10	*Confirmador de archivos cargados*. Este indicador consta de dos indicadores boleanos en color rojo cuando hay confirmación positiva de que se ha cargado un archivo y negro cuando hay confirmación negativa.	`PANEL_LED_CLASES_FILE` y `PANEL_LED_SUJETOS_LOADED`	N/A
11	*Identificador de task*. Esta caja de texto no es editable e indica el nombre del *task*, que se encarga de comunicarse con el driver de la tarjeta de adquisición de datos. Existe configurado un *task* para cada cajón del PA. Dicha configuración se realiza mediante la interfaz que ofrece el módulo DAQmx de LabWindows, donde se establece el número de canales y la asignaciones de pines que se utlizan en la tarjeta.	`PANEL_TASKID`	N/A

	timer. Este componente sólo es visible en el panel de diseño de la GUI, pero está presente en el código de la aplicación en todo momento que se haga un procesamiento de datos en línea. Si algún otro método de la aplicación (por ejemplo un botón de control) direcciona el atributo boleano `ATTR_ENABLED` y lo pone a 1 (`SetCtrlAttribute (panelHandle, PANEL_TIMER, ATTR_ENABLED, 1);`), *timer* llama a su método `realTimeR_Recognition()` en cada `EVENT_TIMER_TICK` de manera indefinida, hasta que se ponga nuevamente `ATTR_ENABLED = 0`, desde algún método o función llamados en la aplicación.	`PANEL_TIMER`	`realTimeR_Recognition()`

Apéndice B

Código fuente para curvas de tendencia

```
%=========================================================================
% Title:          plasticNetTest.m
% Purpose:        Funcion que calcula y grafica las curvas de
%                 tendencia de generalizacion para N redes
%                 neuronales contenidas en el cell-array nets{}
%                 que se recibe como parametro. PinB es la ma-
%                 triz que contiene los patrones de entrada
%                 originales, objVal es el vector de objetivos
%                 para validar la red despues de la adicion de
%                 ruido, aMax es el maximo valor de a para el
%                 que se desea obtener las curvas.
%                 PinR es un conjunto ya modificado por ruido
%                 para algun experimento previo con el cual
%                 solo se desea hacer la validacion de nets y
%                 ruido es un valor booleano para indicarle a la
%                 funcion si se quiere hacer el experimento com-
%                 pleto (ruido=1) o solo se hara validacion
%                 usando PinR. La funcion regresa el porcentaje
%                 de muestras no reconocidas o confundidas en
%                 porConfusas, regresa el indice de la red de
                  mejor performance (mejor) y el conjunto modi-
%                 ficado por ruido (PinN).
% Created on:     07/08/2013 at 09:07 a.m. by I. Arroyo-Fernandez.
% Copyright:      All Scientific Community. No Rights Reserved.
%=========================================================================
```

```
function [porConfusas mejor PinN] = plasticNetTest(nets, PinB,
                                    objVal, aMax, PinR, ruido)
a = 100;
if a == aMax
    disp('Division por cero!!');
    return;
end
[r c] = size(PinB);

if ruido
 PinN = zeros(r, c);
 x = zeros(r, c);
 si = -1;
else
    PinN = PinR;
end

mses = zeros(8,length(nets));
ee = zeros(length(nets), length(objVal));
ss = zeros(length(nets), length(objVal));
l = 1;

for k=0:10:aMax
  if ruido
    for i=1:r
        for j=1:c
            x=rand(r,c)/(a-k);
            si=si*(-1);
            PinN(i,j)=PinB(i,j)+(si)*x(i,j);
        end
    end
  end

  for i = 1:length(nets)
     ss(i,:)=sim(nets{i},PinN');
  end

  for i=1:length(nets)
     ee(i,:)=ss(i,:)-objVal';
     mses(l,i)=(1/length(objVal))*sum(ee(i,:).^2);
  end

 l=l+1;
```

```
end

 minimo = min(min(mses));
 restas = zeros(length(nets), l-1);
 sumas = zeros(1, length(nets));

  for i = 1:length(nets)
      for j = 1:l-1
          restas(i,j) = mses(j,i)-minimo;
      end
      sumas(i) = sum(restas(i,:));
  end

 [v mejor] = min(sumas);
 clear v;
 confusas = 0;
 objVal = objVal';

for i=1:length(objVal)
     if abs(ss(mejor,i)-objVal(i)) > 0.5
         confusas = confusas + 1;
     end
end
t = 1:length(objVal);
figure; plot(t, ss(mejor,:), t, objVal);
porConfusas = (confusas/length(objVal))*100;
figure;semilogy(mses');
```

Apéndice C

Código fuente para manejo de archivos

```
//==============================================================================
// Title:       fileDriving.c
// Purpose:     This file holds functions designed for accessing
//              files that LAMDA reads  or writes.
// Created on:  11/03/2013 at 12:07 p.m. by I. Arroyo-Fernandez.
// Copyright:   All Scientific Community. No Rights Reserved.
//==============================================================================
//======= Include files lies here ==============================================
#include "toolbox.h"
#include <analysis.h>
#include <ansi_c.h>
#include <cvirte.h>
#include <userint.h>
#include "lamda.h"
#include "fileDriving.h"
//========================= Constants ==========================================
#define MAX_STATE_STRING_LENGTH 50

/// HIFN  Esta funcion jala (por referencia) los parametro del
//        archivo de clases, mismos que estan contenidos en la
//        cabecera del mismo. Dichos parametros son unicamente
//        el numero de descriptores y el numero clases y el
//        numero de descriptores
/// HIPAR path/ Es una cadena que contiene la ruta del archivo
//        que se va a accesar.
/// HIPAR clases/ Con este apuntador se regresa por referencia
```

```
//          el numero de clases que se encuentran en el archivo
//          accesado.
/// HIPAR descriptores/ Con este apuntador se regresa por refe-
//          rencia el numero de descriptores mas uno (por el ren-
//          glon de elementos) que se encuentran en el archivo
//          accesado.
/// HIRET Regresa el codigo de error (>= 0) en caso de haberlo;
//          -1 si no hay errores.
int matrixClasesConfig(char *path,int *clases, int *descriptores)
{         //Header example: 05100000053
          //                3 |   8    = 11
        FILE *archivo;
        char cabecera[12] = {0}, r[4] = {0}, c[9] = {0};
        int i;

        archivo = fopen(path, "r");
        //Error Handling
        if (archivo == NULL || path == "")
                return 0;
//      12 caracteres que incluyen 11 digitos mas el fin de cadena:
        fgets(cabecera, 12, archivo);
//      Verificar integridad de la cabecera, en caso negativo :
        if (strlen(cabecera) != 11)
                return 2;     //terminar funcion y regresar error.
        for (i = 0; i < 3; i++)
                r[i] = cabecera[i];
        r[i] = '\0';
        for (i = 0; i < 8; i++)
                c[i] = cabecera[i + 3];
        c[i] = '\0';
        *descriptores = atoi(r);
        *clases       = atoi(c);
        fclose(archivo);
        return -1;            // -1 = there weren't errors
}
//=============================================================
/// HIFN    This function almacena la matriz con clases nuevas mas
//          las ya existentes reforzadas en el archivo cuya ruta
//          sea path.
/// HIPAR   path/Es una cadena que contiene la ruta del archivo que
//          se va a accesar.
/// HIPAR   clusterMatrix/Apunta la matriz que contiene el cono-
//          cimiento existente mas el nuevo (en caso de haberlo).
```

```
//          Ademas de una columna de ceros que solo cumple con el
//          formato establecido.
/// HIPAR   Ncls/Numero de clases que contiene la matriz de cono-
//          cimiento.
/// HIPAR   Ndtrs/Numero de descriptores que contiene la matriz
//          de cono cimiento.
/// HIRET   Regresa el codigo de error (>= 0) en caso de haberlo;
//          -1 si no hay errores.
int storeKnoledgeFile(char *path, double **clusterMatrix, int Ncls,
                      int Ndtrs)
{
        FILE *archivo;
        char bufferA[13] = {0}, *bufferB, bufF[9]= "0000" ,
             bufG[9]= "000", bufH[9]= "00000" ,bufI[9]= "000000" ,
             bufJ[9]= "0000000" , bufA[9]= "00", bufB[9] = "0",
             bufC[4] = "", bufD[13] = "", bufE[9] = "";
        int i = 0, j = 0;

        bufferB = (char *) malloc ((((Ncls * 13)+13) + Ncls + 1)
                                        * sizeof(char));

        archivo = fopen (path, "w");

        if (archivo == NULL || path == "")
                return 2;

        sprintf (bufC, "%d", Ndtrs);
        int L = strlen (bufC);
        switch (L)
        {
                case 1:
                        strcat (bufA, bufC);
                        strcpy (bufD, bufA);
                        break;
                case 2:
                        strcat (bufB, bufC);
                        strcpy (bufD, bufB);
                        break;
                case 3:
                        strcpy (bufD, bufC);
                        break;
        }
```

```
        sprintf (bufC, "%d", Ncls);
        sprintf (bufA, "00");
        sprintf (bufB, "0");
        L = strlen (bufC);
        switch (L)
        {
                case 1:
                        strcat (bufJ, bufC);
                        strcpy (bufE, bufJ);
                break;
                case 2:
                        strcat (bufI, bufC);
                        strcpy (bufE, bufI);
                break;
                case 3:
                        strcat (bufH, bufC);
                        strcpy (bufE, bufH);
                break;
                case 4:
                        strcat (bufF, bufC);
                        strcpy (bufE, bufF);
                        break;
                case 5:
                        strcat (bufG, bufC);
                        strcpy (bufE, bufG);
                        break;
                case 6:
                        strcat (bufA, bufC);
                        strcpy (bufE, bufA);
                        break;
                case 7:
                        strcat (bufB, bufC);
                        strcpy (bufE, bufB);
                        break;
                case 8:
                        strcpy (bufE, bufC);
                        break;
        }
        strcat (bufD, bufE); // Se forma y se imprime el Header.
        fputs (bufD, archivo);
        fputc (10, archivo);
//=================================================================
        bufferB[0] = '\0';
```

```
        for (i = 0; i < Ndtrs; i++)
        {
                for(j=0, bufferB[0] = '\0'; j <= Ncls; j++)
                {  // (1):
                   sprintf (bufferA, "%f,", clusterMatrix[j][i]);
                   strcat (bufferB, bufferA);
                }
//matrixCluster[][] is treated row by row in this loop.
                bufferB[strlen (bufferB)-1] = 0;
                fputs (bufferB, archivo);
                fputc (10, archivo);
        }

        fclose (archivo);
        return -1;
}
//=========================================================
/// HIFN  Esta funcion sirve para cargar el contenido del archivo
//        de las cases en la matriz de clases. Dicho archivo debe
//        estar en formato CSV (Coma Separated Values).
/// HIPAR path/Es una cadena que contiene la ruta del archivo que
//        se va a accesar.
/// HIPAR matrizClases/Apunta a la matriz donde sera cargado el
//        contenido del archivo accesado.
/// HIPAR Ncl/Indica a la funcion el numero de clases que contiene
//        el archivo que se va accesar.
/// HIPAR Nds/Indica a la funcion el numero de descriptores que
//        contiene el archivo que se va accesar (considerando
//        claseNIC y sin considerar la columna de ceros).
/// HIRET Regresa el codigo de error (>= 0) en caso de haberlo;
//        -1 si no hay errores.
int loadFile(char *path, double **matrizClases, int Ncl, int Nds)
{
        FILE *archivo;
        char *numero, **nums, num[14], c = 0, listo = 0;
        int i = 0, j = 0, k = 0;
        numero = (char *) malloc ((((Ncl * 13)+13) + Ncl + 1)
                                        *sizeof(char));

        nums = (char **) malloc ((Nds + 1) * sizeof(char *));
        for (i = 0; i < Nds + 1; i++)
           nums[i] = (char *) malloc((((Ncl * 13)+13) + Ncl + 1)
                                        * sizeof(char));
```

```
        archivo = fopen(path, "r");

        if (archivo == NULL)
                return 0;

        fgets(numero, 13, archivo);
        for (i = 0; i < Nds; i++)
        {
            fgets(numero, ((Ncl * 13)+13) + Ncl + 1, archivo);
            for (j = 0; ; j++)
            {
                if((numero[j] != 10) && (numero[j] != 0))
                    nums[i][j] = numero[j];
                else
                    break;
             }
             nums[i][j] = '\0';
        }
        int m = 0;
        for (i = 0; i < Nds; i++)
        {
                k = 0;
                for (j = 0; j < strlen (nums[i]) + 1; j++)
                {
                    if (nums[i][j]!=44 && nums[i][j]!=0)
                    {
                          num[m] = nums[i][j];
                          m++;
                    }
                    else
                          {
                             num[m] = '\0';
                             matrizClases[k][i] = atof(num);
                             k++;
                             m = 0;
                          }
                }

        }
        fclose(archivo);
        return -1;
}
//=================================================================
```

```
/// HIFN  Esta funcion carga el archivo de etiquetas para cada clase
//        para que el clasificador asocie en tiempo real.
char **loadStates (char *path, int *point)
{
    FILE *archivo;
    int i = 0;
    char **stateMatrix;

    stateMatrix = (char **) malloc (Nclases * sizeof(char *));
    for (i = 0; i < Nclases; i++)
       stateMatrix[i] = (char *)malloc (MAX_STATE_STRING_LENGTH
                                        * sizeof(char));
    archivo = fopen(path, "r");

    for (i = 0; !feof(archivo) && i < Nclases; i++)
       fgets (stateMatrix[i], MAX_STATE_STRING_LENGTH, archivo);

    stateMatrix =(char **)realloc(stateMatrix,i*sizeof(char*));
    fclose (archivo);
    *point = i;
    return  stateMatrix;
}
```

Apéndice D

Código fuente para procesamiento de señales

```
//==============================================================================
// Title:       inputSignalTreating.c
// Purpose:     This code is attempting to acquire and processing
//              signals coming from each cajon del PA in order to
//              perform pattern recognition via LAMDA. It will be
//              applyed to fail diganosis of PA system at STC
//              Metro, in Mexico City.
// Created on:  11/04/2013 at 02:01 p.m. by I. Arroyo-Fernandez.
// Copyright:   All Scientific Community. No Rights Reserved.
//==============================================================================
// Include files
#include <cviauto.h>
#include <analysis.h>
#include <NIDAQmx.h>
#include <ansi_c.h>
#include <cvirte.h>
#include <userint.h>
#include "inputSignalTreating.h"
#include "User Interface Application.h"
#include "toolbox.h"
#include <cvitdms.h>
// Constants
#define CENTER_FREQ_23k 1E3//23.0E3
#define WIDTH_FREQ_23k  2000//5.00E3
#define CENTER_FREQ_100k 100.0E3
#define WIDTH_FREQ_100k  10.00E3
```

```
#define samplesPerChannel 16384
//==========Profile mapping for each PA module==========
#define mapaCajonI_PACMC    "XgAcXXXXjAmXXXAXXXXXXXXXXgXXXXXXXgX\\
                             XXXXXgCgXeXeXAXXXXXXXXXXeXXXkXXXXXb\\
                             XbXXXgXXXmXXXXXXXkXfhXXmAfhgAj"
#define mapaCajonII_PACMC   "XXXXXeXXXXpXXXXXXXXXXXXXXXaXXXXXXXX\\
                             XXXXXXXXXXXXXXXXXgsXXXXXXXtXXXXXXXiX\\
                             XXXXXgyXXfXXXXXXXfXXXXXXXXbXXX"
#define mapaCajonIII_PACMC  "mgXgmXXXXXbXXXXXXXXXXnXXXXXXXozXXAD\\
                             XXXzXXAXXXXXXXXXXXXXXXXXXiXXXXXXXXXX\\
                             XXXXXXXXXXzXXXXXXXbXXXXXXXbkdXX"
#define mapaCajonIV_PACMC   "XXXXXXXXXXXjXegXXXXeXXuXXXXXXXXXXXXXe\\
                             XXiXiXXXXXXubXrqAXAXXgXXXAXXXgXXXXXX\\
                             XgXXXXXXXgXXXgXXXggXXXfgAXvgbX"
#define mapaCajonI_CMLCMR   "XXXXXXXXXXXXXXhBhXBXXXXXXXXXXXYXYXBeB\\
                             XXXXXXXXXXXBXXXBXXXXXXXXXBXXXXBXXXXX\\
                             XXXXXfXfXBXXXBXXXbXbXXBXXXBYXY"
#define mapaCajonII_CMLCMR  "YXXYXXYYXgFXXXYXXXXXXXXXXXYXXXXXXXXX\\
                             XXXGXXXXXXXXXXYXXXXXgXXXYXXXXXXXXYXX\\
                             XXXXXXXXXXXXXXXXXXiXXXXXXXXXgXX"
//==========Sampling Periods==========
#define sampCajonI_PACMC    0.0001
#define sampCajonII_PACMC   0.00025
#define sampCajonIII_PACMC  12.04E-6
#define sampCajonIV_PACMC   12.04E-6
#define sampCajonI_CMLCMR   12.04E-6
#define sampCajonII_CMLCMR  12.04E-6
//==========Amplitude limits==========
#define ampINF23 0.35
#define ampSUP23 0.75
#define ampINF100 0.45
#define ampSUP100 0.85
//====Type variable for frequecy limits searching==========
SearchType span;
//==========Static global variables==========
static TaskHandle daqTask;
//==========Global variables==========
float64 ***dataFile, *dataIn, **signals, **processed,
        **mapaMaxsMins;
double **normalized;
//==========UI callback function prototypes==========
void signalSorter(int *numCanales, int muestrasPorCanal, float64
        *datosEntrada, float64 **canales);
```

```
void signalProcessing (void);
void memoryReservation (void);
void dataNormalization(void);
void dataAutoNormalization(void);
void stopTask(void);
//=============================================================
/// HIFN  Esta funcion carga y configura el Task que se vaya
//        utilizar  segun el cajon seleccionado , pueden ir mas
//        cases , en caso de agregar JH(NM-78) al programa.
/// HIPAR cajon/Numero de cajon que se va a diagnosticar (1-6).
int taskSelectingAndSignalMapping (int cajon)
{
  char *tarea = "                              ";
  int error = 0;

  if (!perfil.running)
  {
    switch (cajon)
    {
      case 0:
        errChk(DAQmxLoadTask("cajon1_PACMC0",&daqTask));
        errChk (DAQmxGetTaskAttribute (daqTask,
                                       DAQmx_Task_Name, tarea , 20));
        SetCtrlVal (panelHandle , PANEL_TASKID, tarea);
        perfil.perfilMap = mapaCajonI_PACMC;
        perfil.sampling = sampCajonI_PACMC;
        break;
      case 1:
        errChk(DAQmxLoadTask("cajon2_PACMC0",&daqTask));
        errChk (DAQmxGetTaskAttribute (daqTask,
                                       DAQmx_Task_Name, tarea , 20));
        SetCtrlVal (panelHandle , PANEL_TASKID, tarea);
        perfil.perfilMap = mapaCajonII_PACMC;
        perfil.sampling = sampCajonII_PACMC;
        break;
      case 2:
        errChk(DAQmxLoadTask("cajon3_PACMC0",&daqTask));
        errChk (DAQmxGetTaskAttribute (daqTask,
                                       DAQmx_Task_Name, tarea , 20));
        SetCtrlVal(panelHandle , PANEL_TASKID, tarea);
        perfil.perfilMap = mapaCajonIII_PACMC;
        perfil.sampling = sampCajonIII_PACMC;
        break;
```

```
        case 3:
            errChk(DAQmxLoadTask("cajon4_PACMC0",&daqTask));
            errChk (DAQmxGetTaskAttribute (daqTask,
                                DAQmx_Task_Name, tarea, 20));
            SetCtrlVal(panelHandle, PANEL_TASKID, tarea);
            perfil.perfilMap = mapaCajonIV_PACMC;
            perfil.sampling = sampCajonIV_PACMC;
            break;
        case 4:
            errChk(DAQmxLoadTask("cajon5_CMLCMR0",&daqTask));
            errChk(DAQmxGetTaskAttribute (daqTask,
                                DAQmx_Task_Name, tarea, 20));
            SetCtrlVal (panelHandle, PANEL_TASKID, tarea);
            perfil.perfilMap = mapaCajonI_CMLCMR;
            perfil.sampling = sampCajonI_CMLCMR;
            break;
        case 5:
            errChk(DAQmxLoadTask("cajon6_CMLCMR0",&daqTask));
            errChk (DAQmxGetTaskAttribute (daqTask,
                                DAQmx_Task_Name, tarea, 20));
            SetCtrlVal (panelHandle, PANEL_TASKID, tarea);
            perfil.perfilMap = mapaCajonII_CMLCMR;
            perfil.sampling = sampCajonII_CMLCMR;
            break;

    }
    errChk (DAQmxGetTaskAttribute (daqTask, DAQmx_Task_NumChans,
                                &perfil.numChannels));
    perfil.dataSize = perfil.numChannels * samplesPerChannel;
    dataIn =(float64*)malloc(perfil.dataSize*sizeof(float64));
    errChk (DAQmxStartTask (daqTask));
    perfil.running = 1;
  }
  return 0;
  Error:
        if (error == -201003)
            MessagePopup ("Error de hardware", "El dispositivo\\
                  de Adquisicion no esta disponible o se\\
                  encuentra desconectado");
        else
            DAQmxClearTask (daqTask);
        SetCtrlVal (panelHandle, PANEL_TOGGLE_ONLINE_LEARN, 0);
        return error;
```

```
}
//=====================================================================
/// HIFN  Rutina para llamada a funciones de preprocesamiento
int ProcessingProfile (void)
{
  int error = 0;
  errChk(DAQmxReadAnalogF64 (daqTask, samplesPerChannel, 10.0,
    DAQmx_Val_GroupByChannel, dataIn, perfil.dataSize, NULL, 0));
  memoryReservation ();
  signalSorter(&perfil.numChannels, samplesPerChannel, dataIn,
                                                        signals);
  signalProcessing ();
  dataAutoNormalization ();
  return 0;
Error:
       return error;
}
//=====================================================================
/// HIFN The next fuction takes the globaly acquired dataIn[] array
//       provided by DAQmxReadAnalogF64(), in ...
///HIPAR datosEntrada/ which has the samples of each channel in a
//       inconvenient way to treat them. Each channel is sorted in
//       a better form to be treated.
///HIPAR canales/ points to a matrix which number of rows is
///HIPAR numCanales/ the number of channels and ...
///HIPAR muestrasPorCanal/ Is the number of columns, it is number
//       of samples per channel and per acquirement.
void signalSorter(int *numCanales, int muestrasPorCanal,
                                  float64 *datosEntrada, float64 **canales)
{
   uInt32 i, j = 0, k;

   for (i = 0; i < *numCanales; i++)
   {
       for (j = 0; j < muestrasPorCanal; j++)
           canales[i][j] = datosEntrada[(muestrasPorCanal * i) + j];
   }
}
//=====================================================================
/// HIFN The next fuction gives the preprocessed signals depending
//       on each component of mapping vectors: perfil.perfilMap[i].
//       Parameters are taken from global variables.
void signalProcessing (void)
```

```
{
  double prom, frec, amp, fase, f0;
  int i, j;
  for (i = 0; i < perfil.numChannels; i++)
  {    //TODAS las de CD.
      if(perfil.perfilMap[i] != 'a' && perfil.perfilMap[i] != 'c')
      {
          Median(signals[i], samplesPerChannel, &prom);
          for (j = 0; j < samplesPerChannel; j++)
          processed[i][j] = prom;
      }
      if (perfil.perfilMap[i] == 'a')  //23KHZ
      {
          span.centerFrequency = CENTER_FREQ_23k;
          span.frequencyWidth = WIDTH_FREQ_23k;
          SingleToneInfo (signals[i], samplesPerChannel,
                          perfil.sampling, &span, &frec, &amp, &fase);
          SetCtrlVal (panelHandle, PANEL_FREC, frec);
          if (amp > ampINF23 && amp < ampSUP23)
              f0 = 1;
          else
              f0 = 0;
          for (j = 0; j < samplesPerChannel; j++)
              processed[i][j] = frec*f0;
      }
      if (perfil.perfilMap[i] == 'c')  //100KHZ
      {
          span.centerFrequency = CENTER_FREQ_100k;
          span.frequencyWidth = WIDTH_FREQ_100k;
          SingleToneInfo (signals[i], samplesPerChannel,
                          perfil.sampling, &span, &frec, &amp, &fase);
          if (amp > ampINF100 && amp < ampSUP100)
              f0 = 1;
          else
              f0 = 0;
          for (j = 0; j < samplesPerChannel; j++)
              processed[i][j] = frec*f0;
      }

  }
}
//==========================================================================
/// HIFN This rutine allocates dynamic memory for all variables
```

```
void memoryReservation (void)
{
   signals = (float64 **) malloc (perfil.numChannels
                                  * sizeof(float64 *));
   processed    = (float64 **) malloc (perfil.numChannels
                                  * sizeof(float64 *));
   normalized   = (double **) malloc (perfil.numChannels
                                  * sizeof(double *));
   for (int i=0; i<perfil.numChannels; i++)
   {
      signals[i] = (float64 *) malloc(samplesPerChannel
                                  *sizeof(float64));
      processed[i] = (float64 *) malloc(samplesPerChannel
                                  *sizeof(float64));
      normalized[i]= (double *) malloc(samplesPerChannel
                                  *sizeof(double));
   }
}
//=============================================================
/// HIFN This rutine normalizes data given by signalProcessing()
//       which are in processed[][]
void dataAutoNormalization(void)
{

  int i=0, j=0, mai=0, mii=0;
  double maxi=0, mini=0;

  for (i=0; i<perfil.numChannels; i++)
  {
     if (perfil.perfilMap[i]=='a' || perfil.perfilMap[i]=='c')
     {

        if (mapaMaxsMins[i][0] < processed[i][0])
           mapaMaxsMins[i][0] = processed[i][0];
        mapaMaxsMins[i][1] = 0;
     }
     else
     {
        MaxMin1D (signals[i], samplesPerChannel, &maxi, &mai,
                                                 &mini, &mii);
        if (mapaMaxsMins[i][0] <maxi)
           mapaMaxsMins[i][0] =maxi;
        if (mapaMaxsMins[i][1] >mini)
```

```
            mapaMaxsMins[i][1] = mini;

        }
    }

    for (i = 0; i < perfil.numChannels; i++)
    {
        for (j = 0; j < samplesPerChannel; j++)
            normalized[i][j] = (processed[i][0] - mapaMaxsMins[i][1])
                               / (mapaMaxsMins[i][0] - mapaMaxsMins[i][1]);
    }
    SetCtrlVal (panelHandle, PANEL_FMAX, mapaMaxsMins[0][0]);
    SetCtrlVal (panelHandle, PANEL_FMIN, mapaMaxsMins[0][1]);
}
//==============================================================================
/// HIFN This rutine calls a DAQ stop function which can't be
//       called from several parts of the code.
void stopTask (void)
{
    DAQmxStopTask (daqTask);
}
```

Apéndice E

Código fuente LAMDA

```
//==========================================================================
// Title:        lamda.c
// Purpose:      This source code file holds functios that
//               implements LAMDA pattern recognition algorithm
//               in different versions:
//               * Learning From File (LAMDA_LFF),
//               * Learning On-Line (LAMDA_LOL) and
//               * Recognizing On-line (LAMDA_ROL).
// Created on:   11/03/2013 at 11:39 by Ignacio Arroyo-Fernandez.
// Copyright:    All Scientific Community. No Rights Reserved.
//==========================================================================
// Include files
#include "toolbox.h"
#include <analysis.h>
#include <ansi_c.h>
#include <cvirte.h>
#include <userint.h>
#include "fileDriving.h"
#include "lamda.h"
#include "User Interface Application.h"
//==========================================================================
/// HIFN  LAMDA_ Learn Fron File  (aprendizaje desde un archivo,
//        no desde la tarjeta de adquisicion).
/// HIRET Function returns error code to map in errorMap[][].
int LAMDA_LFF (void)
{
    int maxI = 0, minI = 0, i = 0, j = 0, k = 0, err = -1;
    double max = 0, min = 0, L = 0.6, **dataSubjects;
```

```
FILE *clasesFile, *classLogFile;
char pathFile2[260] = {0}, bufer[6] = {0}, *clasesCadena;

if (FileSelectPopup ("c:\\Users\\utm\\Documents", "*.csv", "",
                  "Seleccione el archivo de CLASES existentes",
                        VAL_LOAD_BUTTON, 0, 1, 1, 0, pathFile))
{
     if (err = matrixClasesConfig (pathFile, &Nclases,
                                          &Ndescriptors) >= 0)
     {
        MessagePopup ("Error de acceso", errorMap[err]);
        return 0;
     }

     if (FileSelectPopup ("c:\\Users\\utm\\Documents", "*.csv",
          "", "Seleccione el archivo de OBJETOS a observar",
                        VAL_LOAD_BUTTON, 0, 1, 1, 0,pathFile2))
     {
        if (err = matrixClasesConfig (pathFile2, &Nsubjects, &j)
                                                          >= 0)
        {
             MessagePopup ("Error de acceso", errorMap[err]);
             return 0;
        }
     }
     else
             return 0;
}
else
     return 0;
if (j == Ndescriptors)
     k = 1;
else k = 0;
switch (k)
{
   case 1:
     dataSubjects = (double **)malloc((Nsubjects + 1)
                                          *sizeof(double *));
     for (j = 0; j < Nsubjects + 1; j++)
        dataSubjects[j] = (double *) malloc (Ndescriptors
                                           * sizeof(double));
     clasesCadena  = (char *) malloc(5 * sizeof(char));
     clasesCadena[0] = 0;
```

```
clasesROs = (double **) malloc((Nclases + 1)
                                        * sizeof(double *));
for (i = 0; i < Nclases + 1; i++)
    clasesROs[i] = (double *) malloc(Ndescriptors
                                        * sizeof(double));
MADs = (double **) malloc(Nclases * sizeof(double *));
for (i = 0; i < Nclases; i++)
    MADs[i] = (double *) malloc((Ndescriptors - 1)
                                        * sizeof(double));
GADs = (double *) malloc(Nclases * sizeof(double));
if (err = loadFile(pathFile, clasesROs, Nclases,
                                        Ndescriptors) >= 0)
{
    MessagePopup ("Error de acceso", errorMap[err]);
    return 0;
}
if (err=loadFile(pathFile2, dataSubjects, Nsubjects,
                                        Ndescriptors)>=0)
{
    MessagePopup ("Error de acceso", errorMap[err]);
    return 0;
}
for (i = 0; i < Nsubjects; i++)
{
    for (j = 1; k < Nclases + 1; j++)
    {
        if (clasesROs[j][0] == 0)
        {
            vacio = j;
            break;
        }
    }
    for (j = 0; j < vacio; j++)
        for (k = 0; k < Ndescriptors - 1; k++)
          MADs[j][k] = pow(clasesROs[j][k + 1],
          dataSubjects[i][k+1])*pow((1-clasesROs[j][k+1]),
                                1 - dataSubjects[i][k+1]);
    GetCtrlVal (panelHandle, PANEL_LAMBDA, &L);
    for (j = 0; j < vacio; j++)
    {
        MaxMin1D (MADs[j], Ndescriptors - 1, &max, &maxI,
                                        &min, &minI);
        GADs[j] = L * min + (1 - L) * max;
```

```
        }
        MaxMin1D (GADs, vacio, &max, &maxI, &min, &minI);
        if (maxI == nic)
        {
            clasesROs[vacio][0] = 2;
            for (j = 1; j < Ndescriptors; j++)
                clasesROs[vacio][j]=0.5+
                                ((dataSubjects[i][j]-0.5)/ 2);
            clasesROs = (double **) realloc (clasesROs,
                              (vacio + 2) * sizeof(double *));
            clasesROs[vacio+1]= 0;
            clasesROs[vacio+1]=(double*)realloc(
                          clasesROs[vacio + 1], Ndescriptors
                                            * sizeof(double));
            for (j = 0; j < Ndescriptors; j++)
                clasesROs[vacio + 1][j] = 0;
            MADs = (double **) realloc (MADs, (vacio + 1)
                                           * sizeof(double *));
            MADs[vacio] = 0;
            MADs[vacio] = (double *) realloc (MADs[vacio],
                          (Ndescriptors - 1) * sizeof(double));
            for (j = 0; j < Ndescriptors-1; j++)
                MADs[vacio][j] = 0;
            GADs = (double *) realloc (GADs, (vacio + 1)
                                            * sizeof(double));
            GADs[vacio] = 0;
            vacio++;
         }
         else
            {
               clasesROs[maxI][0]++;
               for (j = 1; j < Ndescriptors; j++)
                  clasesROs[maxI][j] = clasesROs[maxI][j] +
                    ((dataSubjects[i][j] - clasesROs[maxI][j])/
                                          clasesROs[maxI][0]);

            }
         sprintf(bufer, "%d, ", maxI);
         clasesCadena = (char *) realloc (clasesCadena,
                    strlen(clasesCadena) + (strlen(bufer) + 1)
                                             * sizeof(char));
         strcat(clasesCadena, bufer);
}
```

```
            for (j = 0; j < Ndescriptors; j++)
                clasesROs[vacio][j] = 0;

            classLogFile = fopen ("d:\\cadenaClases.csv", "w");
            fputs (clasesCadena, classLogFile);
            fclose(classLogFile);
            storeKnoledgeFile(pathFile,clasesROs,vacio,Ndescriptors);
            for (i = 0; i < vacio + 1; i++)
                free(clasesROs[i]);
            free (clasesROs);
            for (i = 0; i < vacio; i++)
                free(MADs[i]);
            free (MADs);
            free (GADs);
            free (clasesCadena);
            break;
        case 0:
            MessagePopup ("Error generico", errorMap[5]);
            break;
    }
    return 4;
}
//==========================================================
/// HIFN  LAMDA_  On Line Learning(aprendizaje en tiempo real).
/// HIPAR normData/Vector de datos de entrada normalizados.
/// HIPAR claseReconocida/Regresa por referencia la clase
//        reconocida por el algoritmo.
/// HIPAR normDataVectorSize/El numero de elementos que contiene
//        el vector de entrada de datos normalizados.
/// HIRET Function returns error code to mapping in errorMap[][].
int LAMDA_OLL (double *normData, int *claseReconocida, size_t
                                                normDataVectorSize)
{
    int maxI = 0, minI = 0, j = 0, k = 0;
    double max = 0, min = 0, L = 0.6;
    FILE *clasesFile;

    if (normDataVectorSize != Ndescriptors)
        return 5;
    for (j = 1; k < Nclases + 1; j++)
    {
        if (clasesROs[j][0] == 0)
        {
```

```
        vacio = j;
        break;
    }
}
for (j = 0; j < vacio; j++)
    for (k = 0; k < Ndescriptors - 1; k++)
        MADs[j][k] = pow(clasesROs[j][k + 1], normData[k]) *
             pow((1 - clasesROs[j][k + 1]), 1 - normData[k]);
GetCtrlVal (panelHandle, PANEL_LAMBDA, &L);
for (j = 0; j < vacio; j++)
{
    MaxMin1D (MADs[j], Ndescriptors - 1, &max, &maxI, &min,
                                                    &minI);
    GADs[j] = L * min + (1 - L) * max;
}
MaxMin1D (GADs, vacio, &max, &maxI, &min, &minI);
if (maxI == nic)
{
    clasesROs[vacio][0] = 2;
    for (j = 1; j < Ndescriptors; j++)
        clasesROs[vacio][j] =0.5+((normData[j - 1] - 0.5)/ 2);
    clasesROs = (double **) realloc (clasesROs, (vacio + 2)
                                        * sizeof(double *));
    clasesROs[vacio+1]= 0;
    clasesROs[vacio+1]=(double*)realloc(clasesROs[vacio + 1],
                                Ndescriptors * sizeof(double));
    for (j = 0; j < Ndescriptors; j++)
        clasesROs[vacio + 1][j] = 0;
    MADs = (double **) realloc (MADs, (vacio + 1)
                                        * sizeof(double *));
    MADs[vacio] = 0;
    MADs[vacio] = (double *) realloc (MADs[vacio],
                            (Ndescriptors - 1) * sizeof(double));
    for (j = 0; j < Ndescriptors - 1; j++)
    MADs[vacio][j] = 0;
    GADs = (double *)realloc(GADs,(vacio+1)* sizeof(double));
    GADs[vacio] = 0;
    vacio++;
}
else
   {
     clasesROs[maxI][0]++;
     k = clasesROs[maxI][0];
```

```
        for (j = 1; j < Ndescriptors; j++)
            clasesROs[maxI][j] = clasesROs[maxI][j]
                  + ((normData[j - 1] - clasesROs[maxI][j])/ k);

      }
   *claseReconocida = maxI;
   for (j = 0; j < Ndescriptors; j++)
       clasesROs[vacio][j] = 0;
   return 4;
}
//=============================================================
/// HIFN  LAMDA_ Recognition On Line (reconocimiento de patrones
//        On line).
/// HIPAR normData/ Vector de datos de entrada normalizados.
/// HIPAR claseReconocida/ Regresa por referencia la clase
//        reconocida por el algoritmo.
/// HIPAR normDataVectorSize/El numero de elementos que contiene
//        el vector de entrada de datos normalizados.
/// HIRET Function returns error code to mapping in errorMap[][].
int LAMDA_ROL (double *normData, int *claseReconocida,
                                        size_t normDataVectorSize)
{
    int maxI = 0, minI = 0, j = 0, k = 0;
    double max = 0, min = 0, L = 0.6;
    FILE *clasesFile;

    if (normDataVectorSize != Ndescriptors)
        return 5;

    for (j = 1; k < Nclases + 1; j++)
    {
        if (clasesROs[j][0] == 0)
        {
            vacio = j;
            break;
        }
    }
    for (j = 0; j < vacio; j++)
        for (k = 0; k < Ndescriptors - 1; k++)
            MADs[j][k] = pow(clasesROs[j][k + 1], normData[k])
                  * pow((1 - clasesROs[j][k + 1]), 1 - normData[k]);
     GetCtrlVal (panelHandle, PANEL_LAMBDA, &L);
     for (j = 0; j < vacio; j++)
```

```
        {
            MaxMin1D (MADs[j], Ndescriptors - 1, &max, &maxI, &min
                                                              , &minI);
            GADs[j] = L * min + (1 - L) * max;
        }

        MaxMin1D (GADs, vacio, &max, &maxI, &min, &minI);
        *claseReconocida = maxI;
        return 4;

}
```

Bibliografía

[1] INEGI,"*Información por Entidad Federativa*", Distrito Federal, Censo de población y vivienda 2010, `http://cuentame.inegi.org.mx/monografias/informacion/df/poblacion/default.aspx?tema=me&e=091993`

[2] Metro de la ciudad de México, *Comparación de Afluencia Total 2009 - 2011*, Datos de operación `http://www.metro.df.gob.mx/operacion/compaflu.html`

[3] Metro de la ciudad de México, *Parque Vehicular*, Datos de operación, Marzo de 2008, `http://www.metro.df.gob.mx/operacion/index.html`

[4] Roberto Alamillo Granados, Ma. del Rocío Díaz Vargas, "*Manual Banco Simulador*", Sistema de Transporte Colectivo Metro, Pruebas y Detección de Fallas del equipo Chopper, Julio 2 de 2011.

[5] Evalua DF, "*Pobreza y Educación en el DF 1995 - 2010*", Consejo de Evaluación del Desarrollo Social del Distrito Federal.

[6] SIEMENS Transportation Systems, ''*SACEM, Driver Operated Trains: Mexico City - Line A, Line 8, Line B*'', A High performance ATC Brouchure, 50 Rue Barbès - BP 531, 92532 Montrouge Cedex, France, `http://www.siemens.fr/transportation`.

[7] Sergio R. Guevara Hernández, Anabel Gaytán M, ''*Desarrollo del Software SPY Para el Mantenimiento del Sistema de Pilotaje SACEM de las Líneas 8 y A*'', Jornadas de Innovación y Desarrollo Tecnológico 2009 del STC, No. de Registro 03-2010-0301 10054200-01.

[8] Joseph Aguilar-Martín, López Mántaras, "*The Process of Classification and Learning The Meaning of Linguistic Descriptors of Concepts*", In: M.M. Gupta et E. Sanchez (eds.) Approximate Reasoning in Decision Analysis, North Holland, pp. 165-175, 1982.

[9] J. Carlos Aguado Chao, Joseph Aguilar-Martín "*A Mixed Qualitative-Quantitative Classification Method Applied to Diagnosis*" QR99, 13th International Workshop on Qualitative Reasoning, Chris Price, Loch Awe, Scotland, pp 124-128, 1999.

[10] Joseph Aguilar-Martín, "*Knowledge-Based Supervision and Diagnosis of Complex Process* ", Intelligent Control/Intelligent Systems and Semiotics, 1999. Proceedings of the 1999 IEEE International Symposium on Cambridge, MA, ISSN 2158-9860, 15 Sep 1999.

[11] Tatiana Kempowsky, Joseph Aguilar-Matín, Audine Subias, Marie-Veronique Le Lann, "*Classification Tool Based on Interactivity Between Expertise and Self-Learning Techniques*" IFAC-Safeprocess 2003, Washington D. C. (USA), June 9-11, 2003.

[12] Martha Galindo de Ariza, Joseph Aguilar-Martín, ''*Clasificación de la Personalidad y sus Trastornos con la Herramienta LAMDA de Inteligencia Artificial en una Muestra de Personas de Origen Hispano que Viven en Toulouse-Francia*'', Revista de estudios sociales, Universidad de Los Andes, ISSN 0123-885X, Colombia, 2004.

[13] A. Orantes M. Tatiana Kempowsky, et. al, ''*Selection of Sensors by a New Methodology Coupling a Classification Technique and Entropy Criteria*'', Journal of The European Institution of Chemical Engineers, Volume 85, Issue 6, pp 825-838, 2007.

[14] Germán A. Morales, Juán J. Mora F. et. al, ''*Evaluación Comparativa de Tres Métodos de Clasificación Aplicados a la Localización de Fallas en una Red Eléctrica*'', Universidad Tecnológica de Pereira, ISSN:0122-1701, Vol. XIII número 035, Colombia, pp 19-24, Agosto 2007.

[15] Marie Chan, Joseph Aguilar-Martín, Pierre Celsis, et. al, "*Classification Techniques for Feature Extraction in Low Resolution Tomographic Evolutive Images: Application to Cereblal Blood Flow Estimation*" 12ºColloque Sur le Traitement du Signal et des Images, pp 969-972, DOI:969-972/11392, 1989.

[16] Mónica Sánchez, Francesc Prats, Núria Agell, Joseph Aguilar-Martín, "*A Characterization of Linearly Compensated Hybrid Connectives Used in Fuzzy Classifications*" ed., 'ECAI' , IOS Press, ISBN 1-58603-452-9, pp 1081-1082, 2004.

[17] Sergio Guadarrama Cotado, "*Representación del conocimiento impreciso: Revisión Parcial de las Teorías de Conjuntos Borrosos*" Universidad Politécnica de Madrid, Trabajo de Fin de Carrera, Facultad de Informática, Septiembre de 2000.

[18] Agustín Flores, Eduardo Quiles, et. al., ''*Avances en el Diagnóstico de Fallas en Sistemas Eléctricos de Transporte Mediante Redes Neuronales: Un Enfoque Modular*'', RIELAC, vol. XXXII Abril-Julio 2011.

[19] Juan Carlos Serna A., Victor M. Gómez R., Johan S. Bedoya, ''*Metodología de Implementación en PLC de un Sistema de Diagnóstico Automático para Bandas Transportadoras Basado en el Algoritmo LAMDA*'', Undécima Conferencia Iberoamericana en Sistemas Cibernética e Informática CISCI, Orlando Florida, U. S. A. 17-20 de Julio, 2012.

[20] Alethya D. Salas Armendariz, ''*Caracterización de Fallas Multiplicativas en Generadores de Residuos Basado en Observadores*'', Universidad Autónoma de Nuevo León (UANL),

Facultad de Ingeniería Mecánica y Eléctrica, División de Estudios de Post-Grado, Tesis Para Obtener el Grago de Maestro en Ciencias de la Ingeniería Eléctrica con Especialidad en Control, Ciudad Universitaria, Diciembre de 2002.

[21] Allan S. Willsky, ''*A Survey of Design Methods for Failure Detection on Dynamic Systems*'', Automatica, Vol 12, pp 601-611, Pergamon Press, Great Britain, 1976.

[22] Marcos Guillen, José Paredes y Oscar Camacho, ''*Un Enfoque Para la Detección y Diagnóstico de Fallas en la Instrumentación de un Proceso Usando Reconocimiento de Patrones en el Dominio Wavelet*'', Universidad de los Andes Venezuela, 8ºCongreso Iberoamericano de Ingeniería Mecánica, Cusco Perú, 23-25 Octubre, 2007.

[23] S. de Lira, B. Puig, J. Quevedo, ''*PEM Fuel Cell System Robust LPV Model-Based Fault Diagnosis*'', 7th IFAC Symposium on Fault Detection Supervision and Safety of Technical Processes, DOI:10.3182/20090630-4-ES-2003.00088, ISSN: 1474-6670, pp 528-533, 30-Jun al 03-Jul, 2009.

[24] Jesus E. Paez Castillo, ''*Recobro de Sencibilidad a Ciertas Fallas Multiplicativas de Los Generadores de Residuos Basados en Observadores*'', Universidad Autónoma de Nuevo León (UANL), Facultad de Ingeniería Mecánica y Eléctrica, División de Estudios de Post-Grado, Tesis Para Obtener el Grago de Maestro en Ciencias de la Ingeniería Eléctrica con Especialidad en Control, San Nicolás de los Garza NL, Enero de 2003.

[25] Ungar L. H. and B. A. Powell, ''*Fault Diagnosis Using Non-linear Adaptive Networks*'', University of Pennsylvania, AIChE Annual National Meeting, November 1988.

[26] Ungar L. H., B. A. Powell and S. N. Kamens, ''*Adaptive Networks for Fault Diagnosis and Process Control*'', University of Pennsylvania, Computers Chem. Engng. Vol. 14 No. 45 pp 561-572, January 1990.

[27] L. Felipe Blázquez, Luis J. de Miguel, ''*Diagnóstico Automático de Fallos para Sistemas Dinámicos no Lineales*'', Reporte de Investigación, Departamento de Ingeniería de Sistemas y Automática, Universidad de Valladolid, España, 1999.

[28] C. Angeli, ''*On-line Fault Detection for Nuclear Power Plant Using Recurrent Neural Networks*'', International Journal of Computer Science and Applications Vol. 1, pp 12-30, 2004.

[29] K. S. Swarup, H. S. Chandrasekharaiah, ''*Fault Detection and Diagnosis of Power Systems Using Artificial Neural Networks*'', Department of high voltage engineering, Indian institute of Science, India 1991.

[30] S. R. Naidu, E. Zafiriou and T. J. McAvoy, ''*The Use of Neural Networks for Sensor Failure Detection During the Operation of a Control System*'', Syst. Res. Center, Maryland Univ., College Park, MD, USA IEEE Control Systems Magazine, DOI:10.1109/37.55124, 05/1990.

[31] A. S. Aneetha, S. Bose, ''*The Combined Approach for Anomaly Detection Using Neural Networks and Clustering Techniques*'', Department of Computer Science and Engineering, Anna University, Computer Science & engineering: An International Journal (CSEIJ), Vol. 2 No. 4, India, August 2012.

[32] Manoranjan Pradhan, Sateesh Kumar P., *et al.*, ''*Anomaly Detection Using Artificial Neural Networks*'', International Journal of Engineering Sciences and Engineering Technologies, Volume II, 2012.

[33] S. C. Yao and E. Zafiriou, ''*Control System Sensor Failure Detection Via Networks of Localized Receptive Fields*'', System research center, Harvard University, IEEE American Control Conference, pp 2472-2477, 23-25 May 1990.

[34] Xiao Xu, Wesley Hines, *et al.*, ''*Sensor Validation and Fault Detection Using Neural Networks*'',Maintenance and Reliability Center, The University of Tennessee, Proceedings of the mantenance and Reliability Conference (MARCON), Gatlinburg TN, 1999.

[35] A. J. Piñón Pasos, ''*Aplicación de RNAs a la Detección de Fallos en Sensores*'', Universidade Da Coruña, XXV Jornadas de Automática, Ciudad Real, 8-10 de Sep 2004.

[36] J. M. F. Calado, M. Kowal, M. J. G. Mendes, *et al.*, ''*Neuro-fuzzy Models for Fault Detection Approach Using a Profibus Network*'', Proceedings of the 10th Mediterranean Conference on Control and Automation, Lisbon, Portugal, July 9-12, 2002.

[37] Addison Ríos B., F. Hidrobo, P. Guillén, ''*Diagnóstico de Fallas en Procesos Dinámicos: Un Enfoque Basado en SVM*'', Quinto Congreso Iberoamericano de Estudiantes de Ingeniería Eléctrica (V CIBELEC), Mérida Venezuela, sala 2, del 7-11 Mayo 2012.

[38] D. Julio César Ramírez Valenzuela, ''*Diagnóstico de Fallos en Sistemas Industriales Basado en Razonamiento Borroso Probabilístico*'', Universidad Politécnica de Valencia, Departamento de Ingeniería de Sistemas y Automática, Tesis Doctoral, Valencia España, 20 Abril 2007.

[39] Veelenturf Leo P. J., ''*Analysis and Applications of Artificial Neural Networks*'', Prentice Hall International (UK), ISBN: 0-13-489832-X, 1995.

[40] Warren S. McCulloch, Walter H. Pitts, ''*A Logical Calculus of The Ideas Immanent in Nervous Activity*'', Bulletin of Mathematical Biophysics, Vol. 5, Number 4, DOI: 10.1007/BF02478259, pp 115-133, 1943.

[41] Antonio J. Serrano, Emilio Soria, José D. Martín, ''*Redes Neuronales Artificiales*'', Universidad de Valencia, Escuela Técnica Superior de Ingeniería, Depto. de Ingeniería electrónica, Open Course Ware, curso 2009-2010.

[42] Donald O. Hebb, ''*The Organization of Behaviour, A Neuropsychological Theory*'', McGill University, Jhon Willey & Sons Inc, U. S. A., 1949 (Current Edition ISBN:0805843000).

[43] Eduardo Gasca A., ''*Artificial Neural Networks*'', Instituto Tecnológico de Toluca, México, Apuntes.

[44] Xavier Basogain Olabe, ''*Redes Neuronales Artificiales y sus Aplicaciones*'', Dpto. de Ingeniería de Sistemas y Automática, Escuela Superior de Ingeniería de Bilbao, Retrieved November 14, 2012, from eduCommons Web site: http://ocw.ehu.es/ensenanzas-tecnicas/redes-neuronales-artificiales-y-sus-aplicaciones/contenidos/images/xabier-basogain-olabe, (October 01 2008)

[45] Frank Rosenblatt, ''*The Perceptron a Probabilistic Model For Information Storage And Organization in The Brain*'', Cornell Aeronautical Laboratory, Psychological Reviews 65(6):386-408, November 1958.

[46] Simon Haykin, ''*Neural Networks: A Comprehensive Foundation*'', Second Edition, McMaster University, Prentice Hall, 1999.

[47] Lorenzo Sanzol Iribarren, ''*Implantación de Plan de Mantenimiento TPM en Planta de Congeneración*'', Universidad Pública de Navarra, Escuela Técnica Superior de Ingenieros Industriales y de Telecomunicación, Tesis para obtener el título de ingeniero técnico industrial mecánico, Pamplona España, Septiembre 15 de 2010.

[48] D. Michie, D.J. Spiegelhalter C.C. Taylor, ''*Machine Learning, Neural and Statistical Calssification: Introduction*'', University of Strathclyde, MRC Biostatistics Unit Cambridge and University of Leeds, February 1995.

[49] Ethem Alpaydin, ''*Introduction to Machine Learning*'', Second Edition, The Massachusetts Institute of Technology, ISBN 978-0-262-01243-0, 2010.

[50] Tom M. Mitchell, ''*Machine Learning*'', McGraw-Hill, ISBN 0070428077, March 1 1997.

[51] Sergios Theodoridis and Konstantinos Koutroumbas, ''*Pattern Recognition*'', Elsevier Inc., ISBN-13: 978-1597492720, November 3, 2008.

[52] Raúl Santiago Montero, ''*Clasificador Híbrido de Patrones Basado en la Learnmatrix de Steinbuch y el Linear Associator de Anderson-Kohonen*'', IPN, Tesis para obtener el grado de maestro en computación, Junio 2003.

[53] Seema Asht and Rageshwar Dass, ''*Pattern Recognition: A Review*'', International Journal of Computer Science and Telecommunications[Vol. 3, issue 8, pp 25-29], August 2012.

[54] Xiao-Hu Yu, ''*Can backpropagation error surface not have local minima*'', Neural Networks, IEEE Transactions on Neural Networks (Volume:3, Issue: 6, pp 1019 - 1021, ISSN :1045-9227), Dept. of Radio Eng., Southeast Univ., Nanjing, China, Nov 1992.

[55] Masters Timothy, ''*Practial Neural Networks Recipes in C++*'', Academic Press Inc., ISBN: 0-12-479040-2, 1993.

[56] Iebeling Kaastra and Milton Boyd, ''*Designing a Neural Network for Forecasting Financial and Economic Time Series*'', ELSEVIER Science, Neurocomputing 10, pp 215-236, SSDI: 0925-2312(95)00039-9, 1996.

[57] Mohamed A. Shahin, Mark B. Jaksa, Holger R. Maier, ''*State of The Art of Artificial Nueral Networks in Geotechnical Engineering*'', Electronic Journal in Geotechnical Engineering, 2008.

[58] Steve Lawrence, C. Lee Giles, Ah Chung Tsoi, ''*What Size Neural Network Gives Optimal Generalization? Convergence Properties of Backpropagation*'', Technical Report, University of Maryland, Computer Science Department (UMIACS-TR-96-22), October 1998.

[59] Scott E. Fahlman and Lebiere Christian, ''*The Cascade-Correlation Learning Architecture*'', Carnegie Mellon University, Computer Science Department (CMU-CS-90-100), Research Showcase, Paper 1938, February 1990.

[60] Mark Azzam, Jan-Christoph Haag, Peter Jeschke, ''*Application Concept of Artificial Neural Networks for Turbomachinery Design*'', Computer Assisted Mechanics and Engineering Sciences 16, pp: 143 to 160, Institute of Fundamental Technological Research, Polish Academy of Sciences, 2009.

[61] Gábor HORVÁTH, ''*Neural Networks in System Identification*'', Proceedings of the NATO Advanced Study Institute on Neural Networks for Instrumentation, Measurement and Related Industrial Aplications, Crema Italy, 9-20 October 2001.

[62] Martin T. Hagan, Howard B. Dermuth, ''*Neural Network Design*'', PWS Publishing Company, ISBN: 7-111-10841-8, 2002.

[63] Daniel L. Chester, ''*Why Two Hidden Layers Are Better Than One*'', IJCNN-90-WASH-DC, Lawrence Erlbaum, volume 1, pp 265-268, 1990.

[64] Guoqiang Zhang, B. Eddy Patuwo, Michael Y. Hu, ''*Forecasting with Neural Networks: The State of The Art*'', ELSEVIER Science, International Journal Of Forecasting 14, pp 35-62, 1998.

[65] Mark R. Baker, Ragendra B. Patil, ''*Universal Approximation Theorem for Interval Neural Networks*'', Spring, Reliable Computing Journal Vol. 4, issue 3, pp 235-239, 1998.

[66] Kur´ Hornik, Maxwell Stinchcombe, Halber White, ''*Multilayer Feedforward Networks Are Universal Approximators*'', ELSEVIER Science, Journal in Neural Networks, Vol. 2 issue 5, pp 359-366, March 1989.

[67] Lior Rokach, ''*A Survey of Clustering Algorithms*'', Springer, Data Mining And Discovery Handbook (2nd Edition), ISBN 978-0-387-09822-7, pp 269-298, 2010.

[68] Marija J. Norušis, ''*IBM SPSS Statistics 19 Statistical Procedures Companion*'', Pearson PLC, ISBN-10: 0-321-74842-5, pp 361-391, febrero 2011.

[69] Guojun Gan, Chaoqun Ma, Jianhong Wu, ''*Data Clustering: Theory, Algorithms, and Applications*'', ASA-SIAM Series on Statistics and Applied Probability, SIAM, Philadelphia, ASA, Alexandria, VA, ISBN: 978-0-898716-23-8, 2007.

[70] Richard O. Duda, Peter E. Hart, David G. Stork, ''*Pattern Classification*'', 2nd edition, Wiley Editorial, ISBN: 978-0-471-05669-0, November 2000.

[71] Jayaram, Balasubramaniam and Klawonn, Frank, ''*Can Fuzzy Clustering Avoid Local Minima and Undesired Partitions?*'', Springer Berlin Heidelberg, Computational Intelligence in Intelligent Data Analysis, pp 31-44, vol. 445, ISBN: 978-3-642-32377-5, 2013.

[72] Lotfi A. Zadeh, ''*Fuzzy Sets*'', Elsevier, Information and control, Vol. 8 Issue 3, pp 338-353, 1965.

[73] Joseph Aguilar-Martín, ''*Inteligencia Artificial Para la Supervisión de Procesos Industriales*'', Universidad de Los Andes, Consejo de Publicaciones, ISBN: 9789801110309, 2007.

[74] Timothy J. Ross, ''*Fuzzy Logic With Engineering Applications*'', Wiley, 3rd Edition, ISBN: 978-0-470-74376-8, 2010.

[75] M. Carbonell, M. Mas, G. Mayor, J. Suñer, J. Torrens, ''*Sobre algunas propiedades en ternas de De Morgan*'', VI Congreso español sobre Tecnologá y Lógica Fuzzy (ESTYLF 96), pp. 71-75, Oviedo 1996.

[76] Ronald E. Walpole, Arnold H. Myers, Sharon L. Myers, ''*Probabilidad y Estadística Para Ingenieros*'', Prentice Hall, 6a. Ed., ISBN: 970-17-0264-6, México 1999.

[77] I. M. Mujtaba, M. A. Hussain, ''*Application of Neural Networks and Other Learning Technologies in Process Engineering*'', Imperial College Press, ISBN: 1-86094-263-6, Singapore 2001.

[78] Juan J. Montaño-Moreno, ''*Redes Neuronales Aplicadas al Análisis de Datos*'', UNIVERSITAT DE LES ILLES BALEARS, Tesis doctorado en Psicología, Palma de Mayorca 2002.

[79] James A. Freeman, James M. Skapura, ''*Neural Networks Algorithms, Applications and Programming Techniques*'', Adison-Wesley, ISBN: 0-201-51376-5, USA 1991.

[80] Lakhmi Jain, Anna Ma. Fanelli, ''*Recent Advances in Artificial Neural Networks, Design and Applications*'', The CRC Press International Series on Computational intelligence, ISBN: 0-8493-2268-5, USA 2000.

[81] M. B. I. Raez, M. S. Hussain and F. Mohd-Yasin, ''*Techniques of EMG Signal Analysis: Detection, Processing, Classification and Applications*'', Biol Proced Online; vol. 8, pp 11-35, March 2006, http://www.ncbi.nlm.nih.gov/pmc/articles/PMC1455479/#!po=1.78571

[82] M.I.A. Lourakis, ''*A Brief Description of The Levenberg-Marquardt Algorithm Implemented by Levmar*'', Technical Report, Institute of Computer Science, Foundation for Research and Technology, Hellas 2005.

[83] Tatiana Kempowsky, Joseph Aguilar-Martín, ''*S. A. L. S. A. User's Manual*'', SALSA Application User's Manual, ver. 1.1 of SALSA, DISCO Group from LAAS-CNRS, All rights reserved, January 2004.

[84] Rosa Guerequeta, Antonio Vallecillo ''*Técnicas de Diseño de Algoritmos*'', Servicio de publicaciones de la Universidad de Málaga, ISBN: 84-7496-666-3, Segunda Edición: Mayo 2000.

[85] S. A. Slah, M. A. Duarte-Mermoud, et. al., ''*Selección de Características Usando Algoritmos Genéticos para Clasificación de Vinos Chilenos*'', Proyecto FONDEF D01-1016, CONICYT-Chile, Universidad de Chile, 2003.

[86] Tor A. Ramstad, ''*Still Image Compression*'', Norwegian University of Science and Technology (NTNU), Chapter 52 of Digital Signal Processing Handbook, CRC Press LLC, IEEE Press, ISBN-13: 9780849385728, 1999.