UNIVERSIDAD DE BUENOS AIRES
FACULTAD DE CIENCIAS EXACTAS Y NATURALES
DEPARTAMENTO DE COMPUTACION

TESIS DE LICENCIATURA

TRATAMIENTO DE UN PROBLEMA DE ANALISIS DE RIESGO CREDITICIO USANDO TECNICAS DE REDES NEURONALES

Gabriela AQUERMAN **LU 259/87**
Verónica SANCHEZ **LU 232/87**

DIRECTOR: **Dr. Enrique Carlos SEGURA**

Diciembre de 1999

RESUMEN

En este trabajo se estudia el problema de evaluación de riesgo crediticio tomando en cuenta antecedentes históricos de comportamiento y se analiza la posibilidad de su tratamiento mediante redes neuronales artificiales. Tres técnicas de esta naturaleza se proponen e implementan: una basada en aprendizaje supervisado (perceptrón multicapa con backpropagation), otra que utiliza aprendizaje no supervisado (modelo de Kohonen), y una tercera técnica híbrida que combina un aprendizaje hebbiano no supervisado con un perceptrón multicapa y backpropagation. Los resultados obtenidos son evaluados y comparados.

ABSTRACT

This work studies the problem of credit scoring taking into account historical behavior and analyzes the possibility of its treatment by means of artificial neural networks. Three different techniques are proposed and implemented: one is based on supervised learning (multilayer perceptron with backpropagation), another one uses unsupervised learning (Kohonen feature mapping), and the third is a hybrid technique which combines both types of learning (unsupervised hebbian learning with a multilayer perceptron and backpropagation). The results of these applications are evaluated and compared.

AGRADECIMIENTOS:

A Enrique, que además de guiarnos en la investigación, nos alentó en todo momento.

A nuestras familias que colaboraron y nos acompañaron incondicionalmente desde el principio de este trabajo.

Gracias
Gabriela y Verónica

INDICE

APENDICES

Capítulo 1

1. INTRODUCCION

En este capítulo se presenta el tema a tratar. Como antecedentes se citan brevemente distintos métodos utilizados a lo largo del tiempo para resolver el problema planteado y una rápida descripción de los trabajos realizados recientemente en el área de las redes neuronales y de análisis de riesgo crediticio. Por último se describe como está organizado el presente trabajo.

La cantidad de información que se acumula en bases de datos de diversa índole ha ido aumentando indiscriminadamente con el correr del tiempo. Todo este volumen de información no es aprovechable si no se explota de manera de encontrar relaciones en los datos que justifiquen su almacenamiento.

Los *sistemas inteligentes* son una categoría de programas computacionales que permiten encontrar patrones y descubrir relaciones en grandes cantidades de datos. El descubrimiento de conocimiento se conoce también como "data mining", que puede definirse como: la extracción no trivial de información implícita en los datos, previamente desconocida y de potencial utilidad. La categoría de sistemas inteligentes abarca diversas técnicas, entre ellas, redes neuronales, algoritmos genéticos, lógica difusa, aprendizaje automático (machine learning) y sistemas expertos.

En la actualidad se están implementando estos sistemas para automatizar la toma de decisiones gerenciales que, hasta el momento, estaban a cargo de personas con mucha experiencia. El uso de estos sistemas inteligentes tiene dos ventajas muy importantes. Una de ellas es el incremento de la calidad del servicio, ya que las decisiones son más precisas, objetivas y consistentes que las de los operadores humanos y los sistemas son más rápidos y no se cansan. La otra ventaja radica en la reducción de costos porque luego del desarrollo inicial e instalación, el costo de uso es muy bajo y es posible tomar decisiones que ahorran dinero.

En particular, en el área de análisis de riesgo crediticio es importante poder predecir en forma automática y objetiva, a partir del comportamiento de un individuo en el pasado, si el mismo será confiable o no en el futuro. Para

realizar esta predicción se pueden analizar los datos personales y el comportamiento financiero de una amplia cantidad de personas y encontrar ciertos patrones o relaciones en los datos que identifiquen a los individuos confiables y a los no confiables.

1.1 Breve descripción del problema

Con una extensa base de datos de antecedentes comerciales y datos personales de diferentes personas, el problema que se plantea radica en poder predecir si una persona resulta confiable o no, en un futuro inmediato, para operar financieramente o pagar deudas que contraiga.

La forma de clasificar a los individuos en confiables y no confiables consiste en compararlos con otros casos de similares características de los cuales ya se conoce su comportamiento en un período pasado cercano que se toma como resultado.

La primera tarea a realizar es clasificar con algún criterio cada uno de los casos de la base de datos. Luego, para un caso no clasificado hay que obtener una respuesta de acuerdo a su similitud con otros casos almacenados.

Las técnicas que se estudian para resolver el problema planteado son diversos enfoques de redes neuronales artificiales (RNA). Para cada técnica se analiza cómo incorporar los datos de entrada para que resulten significativos, cuáles son los mejores parámetros para el algoritmo y cuál es la mejor configuración de la red para lograr una buena predicción. Finalmente, se comparan los resultados obtenidos con los distintos tipos de redes neuronales artificiales.

1.2 Antecedentes

El problema a estudiar se puede incluir en el área conocida como *credit scoring* (puntaje crediticio) que consiste en la asignación de un puntaje a una solicitud de crédito basándose en ciertas características del solicitante y, de acuerdo a este puntaje, decidir si se le otorga el crédito o no.

Históricamente, la asignación de puntaje se realiza por medio de un método simple llamado '*score-cards*' (tarjetas de puntaje), para el cual se determinan las variables a calificar y para cada individuo se suman los puntos obtenidos en

cada uno de los ítems. Como forma de clasificación este método es muy limitado, ya que no tiene en cuenta relaciones entre las variables (por ejemplo: otorgar crédito a postulantes jóvenes, otorgar crédito a postulantes con muy altos ingresos, pero no otorgarlos a los que cumplan ambas condiciones ya que pertenecen a una categoría de riesgo). Igualmente, este método sigue siendo más confiable que la opinión subjetiva de un gerente o de un analista de créditos.

Además de estas técnicas intuitivas se han desarrollado diversos enfoques bottom-up del problema (que construyen modelos matemáticos basados en ejemplos históricos) y otras soluciones top-down en los que un humano con experiencia en el tema debe explicitar las reglas.

Los enfoques top-down abarcan sistemas expertos, algoritmos genéticos y lógica difusa. Su principal ventaja es la claridad en la explicación de sus respuestas y su desventaja es que se deben explicitar reglas iniciales.

Los algoritmos genéticos [Davis, 1991; Goldberg, 1989] se inspiran en la biología evolutiva e imitan el Principio de Darwin que formula la supervivencia de los más aptos. La idea principal es generar un conjunto inicial de soluciones candidatas (un conjunto de reglas) y luego ir produciendo nuevas generaciones de soluciones mejores que la anterior (combinando o cambiando antecedentes). Las ventajas de los algoritmos genéticos son: eficiencia con conjuntos grandes de datos, posibilidad de entrenamiento en paralelo, utilización de datos incompletos, adaptabilidad a cambios en el entorno y explicación de los resultados en forma humanamente comprensible. Las limitaciones son que el tiempo de entrenamiento puede ser muy largo y que la representación del problema afecta mucho la performance.

Los sistemas de lógica difusa [Cox 1994; Kosko, 1992] son conjuntos de reglas que manejan conceptos de lenguaje impreciso (por ejemplo: pequeño, grande, joven, viejo) que se definen como funciones de grado de pertenencia a conjuntos. El resultado final se deriva de la activación de todas las reglas cuyas condiciones coincidan en cualquier grado con los datos de entrada. Esta característica hace a este método muy flexible frente a inconsistencias o incompletitud de los datos. Al igual que los otros mecanismos basados en reglas producen resultados fáciles de comprender y explicar. Las desventajas

son que un humano debe explicitar las reglas y las funciones de pertenencia y que, ante cualquier cambio en el contexto, se debe redefinir estas reglas.

Los enfoques bottom-up provienen de tres áreas de estudio diferentes: estadística, aprendizaje automático o machine learning y redes neuronales [Graf & Nakhaeisadeh, 1993].

Las técnicas estadísticas pueden dividirse en dos subclases: lineales y no lineales. Los métodos lineales incluyen diversas variantes de regresión lineal cuya dificultad radica en que debe conocerse de antemano la función de densidad de las variables. Cuando esta función no es conocida se pueden aplicar modelos no lineales como k-nearest-neighbour, probit o logit. El problema de estos métodos es que requieren mucho conocimiento previo para seleccionar los datos, transformar las variables y estructurar el problema. Además, presuponen una relación fija entre los datos y tienen serios problemas en el manejo de datos incompletos.

Los enfoques de aprendizaje automático se basan en el principio de inducción automática de reglas a partir de ejemplos. Se hicieron muy populares debido al cuello de botella causado por la adquisición de conocimiento en el área de los sistemas expertos. Existen dos formas de aplicarlos: de lo específico a lo general, que comienza generando un modelo consistente de las hipótesis más específicas que explican un ejemplo positivo; y de lo general a lo específico, que inicializa el modelo para contener las hipótesis más generales que rechazan un ejemplo negativo. Luego, ambos modifican sus hipótesis para aceptar ejemplos positivos y rechazar negativos a medida que se presentan [Carbonell & Langley, 1990]. La técnica más conocida de aprendizaje automático es el algoritmo de árboles de decisión [Statlog, 1993; Quinlan, 1986].

Los métodos de aprendizaje automático son intuitivamente más fáciles de entender que los estadísticos y no requieren mucha experiencia para aplicarlos correctamente. Los problemas que presentan son la explosión combinatoria que genera árboles demasiado complejos y la tendencia a un sobre-entrenamiento (dado que tratan de explicar cada uno de los ejemplos) que no permite una buena generalización.

Los modelos de redes neuronales artificiales han sido estudiados durante muchos años con la esperanza de imitar el comportamiento humano en una gran

variedad de campos de aplicación [Hetch-Nielsen, 1990; Wasserman, 1989; Rumelhart, 1986]. Las RNA están compuestas por varios elementos de procesamiento computacionales que funcionan en paralelo y están organizados de manera similar a las neuronas en el cerebro humano. En vez de ejecutar instrucciones en forma secuencial como en un programa de Von Neumann, los modelos de redes neuronales exploran varias hipótesis en forma simultánea. A diferencia de otras técnicas de inteligencia artificial que dependen de algoritmos basados en reglas, las RNA están gobernadas por sistemas de ecuaciones diferenciales. Estas ecuaciones son modificadas por el aprendizaje, variando así la respuesta del sistema. De esta manera las RNA logran aprender las relaciones entre los datos de entrada sin usar técnicas tradicionales de programación.

Las RNA crecieron en popularidad en los últimos años debido a que producen modelos de *credit scoring* más predictivos que los sistemas estadísticos. Además requieren menos conocimiento para ser utilizadas efectivamente.

La dificultad más notoria que presentan las RNA, y que se analiza en este trabajo, es la poca capacidad para explicar sus decisiones.

Conceptualmente, los modelos estadísticos, de machine learning y las redes neuronales no son tan diferentes. Todos utilizan la base de estimación de parámetros de la estadística. La diferencia de estos nuevos sistemas inteligentes con los métodos estadísticos es que tratan de estimar los parámetros a partir de los datos y sin tener que hacer suposiciones sobre la distribución de los mismos ni la relación entre las variables. Estos algoritmos no sólo estiman los parámetros de un modelo de datos en particular, sino que frecuentemente cambian la estructura del modelo en sí mismo.

Otra ventaja de los nuevos métodos sobre los estadísticos tradicionales es que tienen mayor capacidad de manejar temas como complejidad, tamaño y capacidad de los modelos que producen. También resuelven el problema de los "outliers" (valores atípicos en la base de datos que, a veces provienen de un error de carga y que son muy distintos a otros con similar comportamiento a predecir), frente al cual las técnicas estadísticas pueden bajar mucho su nivel de predicción para acomodar el modelo a un solo dato atípico. En cambio, los sistemas inteligentes pueden determinar qué datos de entrada son importantes y cuáles no.

Las técnicas estadísticas más populares se pueden formular como redes neuronales simples. La regresión lineal se puede especificar como un perceptrón con una sola unidad de salida y una función de activación lineal. La regresión logística (LOGIT) se puede lograr con un perceptrón simple y una función de activación sigmoide. Por lo tanto, una red neuronal con más de una unidad en el nivel oculto ofrece mayor posibilidad de generalización que las regresiones. Además, las **RNA** no dependen de suposiciones iniciales sobre independencia o colinealidad de las variables de entrada como los métodos de regresión convencionales.

Los métodos de inducción de árboles de decisión (CHAID, CART, etc) dentro del área de machine learning, a diferencia de las redes neuronales, tienen la desventaja de trabajar con valores de variables de entrada discretos y se puede perder información al transformar los valores continuos en categorías.

Ningún método, incluyendo las redes neuronales, es perfecto para garantizar una solución. Las redes neuronales no deben ser tomadas como un sustituto de las técnicas estadísticas tradicionales, sino como un complemento. Los mejores resultados surgen de la combinación de métodos, supliendo las desventajas de uno por las capacidades de otro.

Por ejemplo, se puede aplicar análisis de componentes principales para reducir la dimensionalidad de los datos que ingresan a una **RNA**. Otra combinación posible es aplicar métodos de árboles de decisión para preprocesar variables categóricas antes de ingresarlas a una **RNA**.

Por otro lado, las **RNA** pueden aumentar el poder de las técnicas estadísticas convencionales, usándolas para eliminar la no linealidad de los datos de un problema, para luego analizar los errores residuales con regresión; de este modo también se potencian las **RNA** al lograr explicar los resultados.

Otra posibilidad de complementar a las redes neuronales para que no sean una "caja negra" es aplicar una técnica de machine learning como CART para realizar una ingeniería reversa de las salidas de las redes supervisadas o los clusters de las no supervisadas. De esta forma, los resultados de la RNA se transforman en un árbol de decisión visibles gráficamente, con reglas escritas

en lenguaje natural y simples de entender, que enuncian las características del segmento de la población que se está analizando.

También se puede utilizar redes neuronales para explicitar las reglas de un sistema de lógica difusa, y de esta manera se elimina la desventaja de alimentación manual de los sistemas de lógica difusa y, por otro lado, se obtiene una clara explicación de las respuestas.

Existen diversas combinaciones posibles de técnicas como utilizar algoritmos genéticos para encontrar los pesos de una red neuronal o para definir las funciones de pertenencia en la lógica difusa. Otra variante es utilizar reglas de un sistema experto como reglas iniciales de un algoritmo genético.

El siguiente cuadro presenta un resumen de las ventajas y desventajas de los distintos métodos con que se puede analizar el problema:

Método	Ventajas	Desventajas
Scorecards	Explicación de respuestas.	No detecta relaciones entre variables.
Estadística	Bajo costo computacional.	Supone un modelo de datos fijo. Poca flexibilidad a cambios. No maneja datos incompletos. Sensible a outliers. Conjuntos de tamaño limitado.
Redes neuronales	Aprendizaje automático. Flexible a cambios. Permite datos incompletos.	Poca explicación de respuestas.
Machine learning	Aprendizaje automático. Flexible a cambios. Permite datos incompletos.	Aprendizaje costoso. Modelos de salida complicados. Entradas discretas.
Sistemas expertos	Respuestas en lenguaje muy comprensible.	Costosa explicitación de reglas en forma manual. Poco flexible a cambios.
Algoritmos genéticos	Aprendizaje automático. Flexible a cambios. Permite datos incompletos.	Afectado por la forma de representación.
Lógica difusa	Aprendizaje automático. Permite datos incompletos. Respuestas comprensibles.	Costosa explicitación de reglas en forma manual. Poco flexible a cambios.

1.3 Actual estado del arte

Es difícil hacer una lista de aplicaciones de redes neuronales en el área de *credit scoring*. Muchas entidades financieras investigaron el tema y desarrollaron prototipos pero es complicado establecer cuántas aplicaciones están realmente en operación, ya que las compañías en general mantienen como

información confidencial el uso de tecnología novedosa. Algunas aplicaciones no pasaron la etapa de prototipo debido a cuestiones políticas de la empresa.

Existen muchas corporaciones en todo el mundo cuya finalidad es específicamente recolectar toda la información necesaria para evaluar riesgo crediticio de personas y empresas, y brindar informes al que lo solicite. En los últimos años, estas empresas han estado investigando y desarrollando productos basados en diversas técnicas para lograr una predicción de riesgo además de la simple información. A continuación se nombran algunas de las principales empresas especializadas en el tema.

En Estados Unidos y Canadá una de las formas tradicionales de evaluar riesgo crediticio es por medio del sistema de "Scorecards" conocido como FICO, desarrollado por Fair, Isaac & Company de San Francisco. Este sistema se basa en la asignación de puntajes a cada variable de análisis y el resultado es la suma de estos puntos.

En Canadá, gran parte del mercado financiero utiliza el sistema Falcon, que también se expandió a otros países de América. La empresa dispone de un sistema on-line denominado "Falcon Credit Alert" para brindar a los usuarios acceso inmediato a la información crediticia.

En España la empresa AIS (Aplicaciones de Inteligencia Artificial) capta la mayor parte del mercado financiero y desarrolló un sistema denominado "Riesgo Global" que, entre otras funciones, permite determinar la probabilidad de morosidad para cada cliente en función de su comportamiento financiero, según el riesgo que el cliente desee asumir. Según su publicidad, los métodos que utiliza para hacerlo son: estadística multivariante, herramientas de clasificación automática y funciones de regresión logística.

Equifax es una multinacional, originada en Estados Unidos, con mucha experiencia en el tema de recolección de datos y credit scoring. Basado en su amplia base de datos, uno de los sistemas de predicción de riesgo que posee se llama "Equifax Risk Score '98", el cual define un modelo analítico que predice la probabilidad de delitos económicos o quiebras dentro de un período de dos años. Este sistema elabora puntajes de riesgo y además brinda explicaciones narrativas de su puntuación. Otro sistema que ofrecen, orientado exclusivamente a la predicción de quiebras, se denomina "Equifax Bankruptcy

Navigator Index '99" que combina la gran cantidad de información de la que disponen con la utilización de diversas herramientas de análisis con: redes neuronales, técnicas de econometría, scorecards y técnicas de segmentación. Este sistema logra mucho mejores predicciones que el sistema de scorecards simple.

En Argentina, la empresa Veraz tiene amplia experiencia en el tema de brindar información de riesgo crediticio y, desde hace poco tiempo, forma parte de Equifax. Posee un área especializada en Sistemas de decisión que implementa soluciones integrales para procesar e interpretar información, facilitando la toma de decisiones en cuanto a otorgamiento de créditos, entre otros fines. Sus productos: Veraz Experto, Decision Power, Exchange 2000 y Bankruptcy Navigator se desarrollaron aplicando las más modernas técnicas de Scoring, Data Mining y Datawarehousing. Sus modelos, al estar representados por reglas, pueden ser modificados sencillamente por el cliente y brindan claras explicaciones a las decisiones que se tomen por su intermedio.

La Corporación Oracle, recientemente, adquirió la empresa Thinking Machines especializada en Data mining que ha estado desarrollando desde mediados de 1994 un sistema denominado Darwin, para cubrir la necesidad de herramientas de este tipo entre sus productos. Se trata de un típico sistema inteligente, tal como los definimos en la introducción, que elabora modelos basados en diversas técnicas para encontrar relaciones y patrones en gran cantidad de información. Oracle Darwin soporta cinco tipos primarios de data mining: redes neuronales, inducción de árboles de decisión, memory-based reasoning o *k*-nearest neighbor, bayesian learning y clustering.

En 1997 se realizó un estudio en la Universidad de Harvard [Galindo, 1998] que compara métodos de redes neuronales, machine learning y estadística para resolver un problema de credit scoring. Las pruebas se realizaron utilizando el sistema Darwin, se entrenó con 2000 ejemplos y se testeó sobre otros 2000. De la comparación de los errores de predicción obtenidos se concluyó que el error más bajo se logró con un método de árboles de decisión (CART) y fue de 8,31%; las redes neuronales resultaron en segundo lugar con un 11%, pero los autores aclaran que se podría haber obtenido mejores resultados si se hubiese entrenado durante un tiempo más largo; el algoritmo de k-nearest-neighbor obtuvo un error promedio de 14,95% y por último, con Probit se obtuvo el error más alto de 15,13%. Los autores también resaltan que los errores podrían

disminuir si se contara con mayor cantidad de datos para entrenamiento y testeo (alrededor de 20000).

El Security Pacific Bank publicitó el uso de redes neuronales para credit scoring en 1991 [Leigh, 1995], los resultados de las RNA fueron comparados con los de un sistema FICO de scorecard tradicional y se comprobó que la capacidad de predicción de las RNA era superior.

El Citibank desarrolló un prototipo utilizando redes neuronales para reemplazar un sistema de score-cards [Leigh, 1995].

Con una gran cantidad de transacciones de datos – 16 millones de transacciones promedio por día – VISA incorporó el uso de redes neuronales para combatir el fraude con tarjetas de crédito [Goonatilake, 1995]. Esta tecnología resultó muy exitosa en cuanto a tiempo (cuanto más rápido se detecte un fraude, más dinero se ahorra) y en cuanto a la tarea de reconocimiento de patrones de fraude, que fue mucho más eficiente que el proceso manual para tal fin.

Dutta & Shekhar (1988) describen el uso de una red neuronal para predecir la clasificación de bonos corporativos con 10 variables de entrada que describen varios aspectos del estado financiero de la compañía. La red neuronal alcanzó un nivel de exactitud en la predicción del 82%.

A diferencia de otras tarjetas de crédito American Express no tiene un tope de gasto para el cliente. El proceso de autorización de crédito es una tarea crítica. Si el crédito se extiende libremente, los clientes pueden no pagar sus cuentas. Si se niega muy frecuentemente, la compañía podría perder negocios y dañar las relaciones con el cliente potencial. Por ello, American Express cuenta con tres proyectos de redes neuronales en desarrollo [Didner, 1995].

En el ámbito académico [Schumann, 1992] se cita un trabajo que compara el comportamiento de varios tipos de redes neuronales con soluciones tradicionales de credit scoring y se obtienen buenos resultados con algunas de las redes. Las variables de los ejemplos que alimentan la red son: posee cuenta bancaria, posee depósitos en bancos o fondos, comportamiento en el pago de créditos anteriores, antigüedad del empleo actual, antigüedad de la residencia

actual, posee propiedades, relación entre la cuota y los ingresos mensuales, importe del crédito, cantidad de cuotas del crédito en meses.

1.4 Organización del trabajo

En el capítulo 2 se describe el funcionamiento de las redes neuronales artificiales, los distintos algoritmos de aprendizaje, los campos de aplicación y las ventajas y desventajas de su utilización.

En el capítulo 3 se explica cómo se procesaron los datos reales para obtener las variables más significativas y poder tomarlos como entrada de una red neuronal.

En el capítulo 4 se expone el perceptrón multicapa con aprendizaje supervisado, básicamente el algoritmo de Backpropagation. Luego se resumen los resultados de aplicar este algoritmo al problema planteado y se comparan diversas variantes en cuanto a parámetros del algoritmo y arquitectura de la red.

En el capítulo 5 se estudia el aprendizaje no supervisado y dentro de éste, el algoritmo de Kohonen. Se comentan los resultados obtenidos de aplicar esta técnica.

En el capítulo 6 se plantea una solución híbrida que combina aprendizaje no supervisado y supervisado. En primer lugar, se utiliza aprendizaje no supervisado hebbiano para reducir la dimensionalidad de los datos de entrada, y luego se aplica backpropagation para obtener una clasificación de los datos.

En el capítulo 7, se comparan los resultados de los tres tipos de experiencias y se extraen conclusiones generales.

En el capítulo 8, se enuncian futuras líneas de investigación posibles como extensión a lo realizado en este trabajo.

Capítulo 2

2. EL ENFOQUE CONEXIONISTA

A continuación se desarrolla el enfoque conexionista, su similitud con el conocimiento biológico que se tiene de las neuronas y se describe a las redes neuronales y su clasificación.

2.1 Introducción a las redes neuronales

El enfoque conexionista o paradigma neuronal propone una estructura que evolucione, dada una entrada (o múltiples entradas) y una salida.

Las redes neuronales artificiales están inspiradas en el comportamiento de las células nerviosas en el cerebro humano (redes neuronales naturales).

La siguiente figura muestra una neurona biológica y sus componentes:

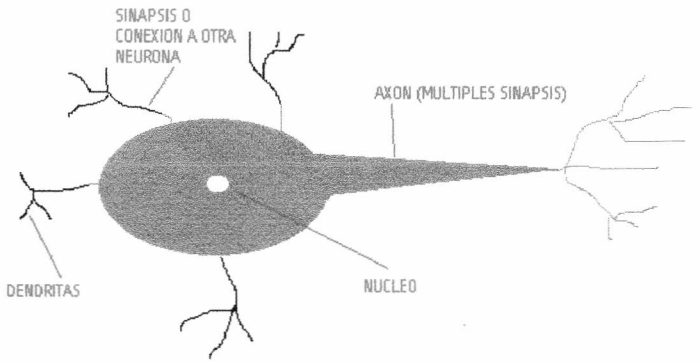


Figura 2.1: Neurona biológica.

El cerebro está compuesto por 10^{11} neuronas de diferentes tipos. Las *dendritas* son fibras nerviosas en forma de árbol que salen del cuerpo de una neurona. Además cada célula posee una fibra nerviosa más larga que se denomina *axón* y tiene varias terminaciones arboladas que actúan como transmisores hacia otras neuronas. Estas transmisiones se llaman *sinapsis* y los terminales receptores se encuentran en las dendritas o en el cuerpo de las células.

La transmisión de una señal de una célula a otra durante una sinapsis es un complejo proceso químico donde ciertas sustancias específicas son liberadas por el transmisor. Esto produce una diferencia en el potencial eléctrico dentro del cuerpo de la neurona receptora y, si el potencial alcanza un determinado valor, el cuerpo de la neurona libera un pulso de duración fija. Este impulso se distribuye a todas las células conectadas al axón.

Las redes neuronales artificiales se diseñaron construyendo un paralelo entre la neurobiología y un modelo matemático.

Una red neuronal artificial se compone de varias unidades paralelas e interconectadas (denominadas nodos o elementos de procesamiento). Cada nodo ejecuta algunas operaciones simples y luego comunica su resultado a las unidades vecinas. Si se cuenta con la tecnología necesaria, estos elementos computacionales pueden trabajar en paralelo.

Las redes neuronales cuyas unidades están organizadas en niveles donde cada unidad pasa resultados sólo a las unidades del nivel inmediato superior (conexiones feed-forward) se denominan perceptrones.

En la *Figura 2.2* se puede apreciar un perceptrón de dos niveles con sus nodos y conexiones:

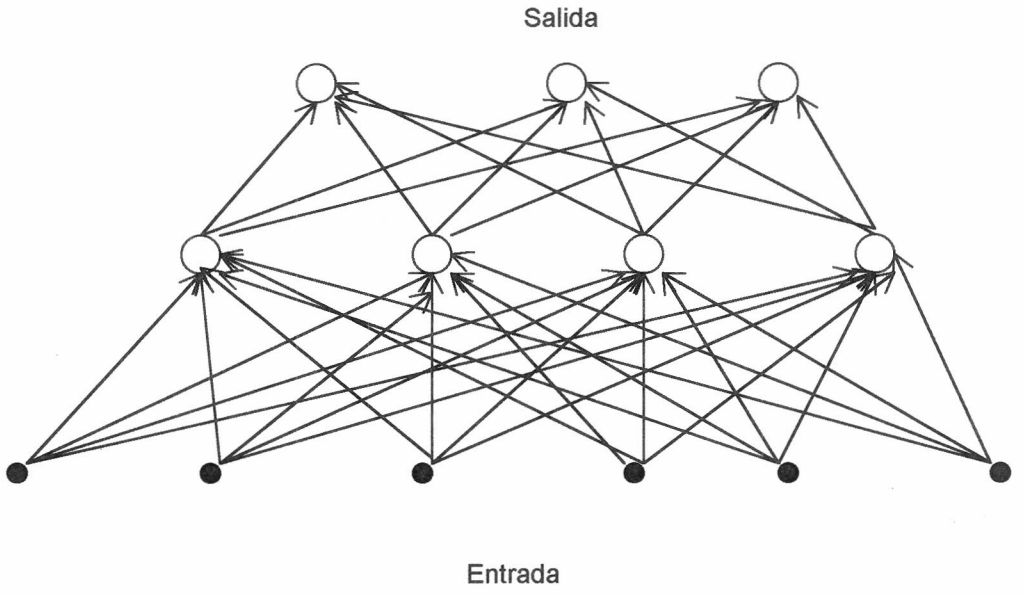


Figura 2.2: Perceptrón de dos niveles.

Mientras los programas de computación convencionales con arquitectura de Von Neumman deben contener una serie de instrucciones paso a paso para realizar una tarea en particular, las redes neuronales, similarmente al cerebro humano, pueden aprender a realizar esa misma tarea por medio de exposiciones sucesivas de diferentes ejemplos.

De esta última observación surgen las principales ventajas de las redes neuronales: pueden reconocer patrones a partir de datos incompletos o con ruido y ejecutar tareas para las que no hay expertos disponibles o no existen reglas claras ya formuladas. Las redes neuronales se están aplicando actualmente a problemas que antes se resolvían usando métodos estadísticos como la regresión lineal o a tareas que antes eran ejecutadas por expertos humanos con muchos años de experiencia.

En general, los nodos de una red neuronal se organizan en niveles (ver Figura 2.2) donde cada unidad de un nivel tiene conexiones a cada unidad del nivel siguiente y a cada conexión se le asocia un peso. Durante el reconocimiento de los patrones de entrenamiento cada nodo suma todas las entradas ponderadas

por los pesos correspondientes (multiplica el peso de la conexión por el estado del nodo del nivel anterior) y luego aplica una función de activación, generalmente no lineal, de la que obtiene su salida. El entrenamiento de una red consiste en encontrar el valor óptimo de los pesos para un determinado problema. Para encontrar estos pesos se utiliza un algoritmo de aprendizaje.

2.2 Clasificación de las técnicas de redes neuronales

Las redes neuronales se clasifican de acuerdo a la forma en que aprenden durante el entrenamiento. Existen dos tipos de aprendizaje:

- Supervisado: el aprendizaje se realiza a través de la comparación directa de las salidas de la red con las respuestas correctas.

- No supervisado: la red crea categorías agrupando los datos de entrada y produce salidas que indican a qué categoría corresponde la entrada presentada. Este tipo de aprendizaje se denomina no supervisado ya que no hace falta proveer las respuestas correctas para cada patrón de entrada.

Cualquiera sea la técnica de aprendizaje hay que ser cuidadoso en no aprender ejemplos que sean demasiado específicos. Las relaciones aprendidas deben representar realmente al problema en general y no sólo reflejar propiedades del conjunto de entrenamiento. Si una red es sobre-entrenada sólo va a poder reconocer los ejemplos en el conjunto de entrenamiento y no va a tener la flexibilidad o capacidad de generalización que demanda la actividad estudiada. Para evitar este problema toda red debe ser testeada con datos fuera del conjunto de entrenamiento. Además es conveniente basarse en los resultados sobre el conjunto de testeo para monitorear el aprendizaje de la red durante el entrenamiento y no solo en el error sobre el conjunto de entrenamiento (esta característica se denomina validación cruzada).

2.3 Aplicaciones de las redes neuronales y sus ventajas

Los ámbitos en que se están aplicando actualmente las redes neuronales son, entre otros:

- Clasificación de señales sonoras y de radar.
- Conversión de voz en texto.

- Aplicaciones médicas.
- Otorgamiento de créditos, robo de tarjetas de crédito, cheques falsos.
- Inversiones y operaciones bursátiles.

En el ámbito de los negocios y las finanzas, las redes neuronales presentan las siguientes ventajas frente a otros sistemas computacionales:

- Aprendizaje: aprenden decisiones o tareas directamente de los datos, es decir, pueden derivar un modelo económico sólo contando con muchos ejemplos de transacciones pasadas. Al prescindir de un experto humano que explicite reglas las redes neuronales tienen ventajas respecto de la consistencia, son muy objetivas en sus decisiones y tienen muy bajo costo.

- Adaptabilidad: las reglas y condiciones en los negocios cambian constantemente, las redes neuronales pueden ser fácilmente reentrenadas cada vez que se presentan cambios en la situación.

- Flexibilidad: los seres humanos pueden tomar decisiones con información incompleta, las redes neuronales también.

- Descubrimiento: las redes neuronales pueden no sólo automatizar tareas que están siendo ejecutadas manualmente, sino también descubrir nuevas relaciones o procesos.

Capítulo 3

3. TRATAMIENTO PRELIMINAR DE LA INFORMACION

El objetivo de este capítulo consiste en conocer la información a manejar en el presente trabajo. Esto es, con qué datos se cuenta, cuál es su codificación, qué describe cada uno de ellos, cómo están organizados, si es necesario algún procesamiento previo para poder utilizarlos, etc.

3.1 Obtención de los datos del problema

La información a analizar proviene de la base de datos de una empresa real donde se registran antecedentes financieros y comerciales de la población, como juicios, cierres de cuenta, inhibiciones, quiebras, concursos, ejecuciones hipotecarias, etc., a los que se denominan *problemas*. Cuando cualquier cliente de esta empresa solicita un informe sobre alguna persona, se registra la fecha y el tipo de la consulta. El volumen de consultas se puede tomar como un indicador de la actividad comercial de la persona.

Los datos por persona son:

1. Sexo
2. Edad
3. Historia de consultas por fecha
4. Historia de problemas por fecha y tipo

Se estableció una fecha tope que se considera el momento actual, los hechos anteriores a esa fecha son tomados como antecedentes y los posteriores como respuesta. De esta manera se puede clasificar a los individuos como confiables o no confiables según si tuvieron problemas o no, luego de esa fecha tope.

Se obtuvo información de 15883 personas. Tomando el último año observado como período de respuesta, resultaron 786 casos no confiables y 15097 casos confiables. Estas cifras muestran que los casos no confiables representan sólo un 5% del total. Esta relación es un factor a tener en cuenta en la selección del conjunto de entrenamiento para las redes neuronales y en el análisis de los resultados.

3.2 Pre-procesamiento de los datos

Para convertir los datos en una tabla plana que pueda ser la entrada a una red neuronal, se pre-procesó la información y se seleccionaron las siguientes variables para cada registro:

- Sexo: 0 indica masculino, 1 indica femenino.
- Edad: expresada en años a la fecha tope.
- Volumen de consultas: cantidad de consultas registradas a la fecha tope.
- Ultima consulta: antigüedad de la última consulta en meses a la fecha tope.

Para representar los problemas se consideraron dos opciones de agrupamiento: por antigüedad o por tipo.

1. Por antigüedad: los problemas se almacenaron en distintas variables de acuerdo a su antigüedad de ocurrencia. En cada variable se guardó la cantidad de problemas encontrados en los siguientes períodos con respecto a la fecha tope:

- Problemas Tiempo 1: Menos de 1 año
- Problemas Tiempo 2: Entre 1 y 3 años
- Problemas Tiempo 3: Entre 3 y 6 años
- Problemas Tiempo 4: Entre 6 y 10 años
- Problemas Tiempo 5: Más de 10 años

2. Por tipo: los problemas se almacenaron en distintas variables de acuerdo a su tipo. En cada variable se guardó la cantidad de problemas encontrados para cada tipo:

- Problemas Tipo 1: Cierres de cuenta
- Problemas Tipo 2: Juicios
- Problemas Tipo 3: Quiebras
- Problemas Tipo 4: Deudas en Tarjetas de crédito
- Problemas Tipo 5: Ejecuciones hipotecarias

Existe un tipo de problema que indica si alguna persona relacionada con la persona analizada tuvo algún problema. Estos antecedentes se indicaron en una variable adicional denominada:

- Relacionados: 0 indica que no existieron relacionados con problemas, 1 indica que sí.

- Variable Respuesta: es binaria e indica la clasificación otorgada a la persona según haya presentado problemas o no luego de la fecha tope; 0 indica confiable, 1 no confiable.

Dado que los valores de las variables difieren en escala, se llevaron todas las variables al rango [0,1]. Para realizar esto, se establecieron valores máximos y mínimos posibles de cada variable y se distribuyeron los valores de la misma entre 0 y 1 usando la siguiente fórmula:

Valor = (Valor – Min) / (Max - Min)

Donde: Min = Mínimo valor que puede tomar la variable
Max = Máximo valor que puede tomar la variable

Al comenzar las pruebas se presentó el inconveniente de que existían muchos casos de variables con valor 0 (en su mayoría las variables que cuentan problemas) y, para solucionarlo, se llevaron los valores 0 a 0.01 para que las entradas tengan mayor influencia sobre todos los pesos de la red.

Se analizaron ambas formas de representación de los datos y se decidió tomar el esquema de variables agrupadas por antigüedad. De esta forma los datos quedaron distribuidos más uniformemente. En cambio, al agruparlos por tipo y por tratarse en su mayoría de juicios y cierres de cuenta, el resto de las variables no aportaba información relevante.

Por lo tanto, las variables usadas finalmente corresponden al esquema "Por antigüedad", siendo las mismas: Sexo, Edad, Volumen de consultas, Ultima consulta, Problemas Tiempo 1, Problemas Tiempo 2, Problemas Tiempo 3, Problemas Tiempo 4, Problemas Tiempo 5, Relacionados y Respuesta.

Capítulo 4

4. APRENDIZAJE SUPERVISADO

En este capítulo se incluye el estudio del aprendizaje supervisado: cómo funciona, cómo deben ser las entradas a la red y qué retorna como salida. También se describe un algoritmo de este tipo como lo es el Algoritmo de Backpropagation (o algoritmo de propagación hacia atrás). Se analizan sus variantes y su aplicación al problema. Por último, se muestran cuáles fueron los resultados arrojados y las conclusiones pertinentes.

El aprendizaje supervisado se basa en comparar la salida calculada por la red neuronal con la respuesta correcta. Es decir que se le proveen entradas de prueba a la red enseñándole cuáles son las respuestas correctas.

4.1 Descripción del algoritmo básico de backpropagation

El algoritmo de Backpropagation (propagación hacia atrás) fue descripto varias veces, por [Bryson y Ho, 1969], [Werbos, 1974], [Parker, 1985], [Rumelhart, Hinton y Williams, 1986] y [Le Cun, 1985].

Es un algoritmo de aprendizaje supervisado, que ajusta los pesos a partir de patrones de entrada que incluyen la respuesta correspondiente. Se presenta una entrada al primer nivel o nivel de entrada (input layer) y se propaga por todos los elementos de procesamiento de la red para producir salidas en el nivel de salida (output layer). Esta salida es comparada con la respuesta deseada para ese patrón y el error obtenido se propaga hacia atrás a través de la red para ajustar los pesos de las conexiones. El proceso se repite para cada ejemplo del conjunto de entrenamiento y se vuelve a pasar por todos los patrones de entrada hasta obtener un error lo suficientemente pequeño. Cada recorrida completa por el conjunto total de patrones se denomina *época*.

A continuación se enuncia una versión incremental del algoritmo básico de Backpropagation [Hertz, 1990]:

Considerando una red con M niveles m=1,2,….,M se denomina V_i^m a la salida de la unidad i del nivel m. V_i^o es un sinónimo de ξ_i (la entrada i). w_{ij}^m

simboliza la conexión de V_j^{m-1} a V_i^m. k es la cantidad de variables de patterns μ de entrada.

La *Figura 4.1* esquematiza la notación anterior:

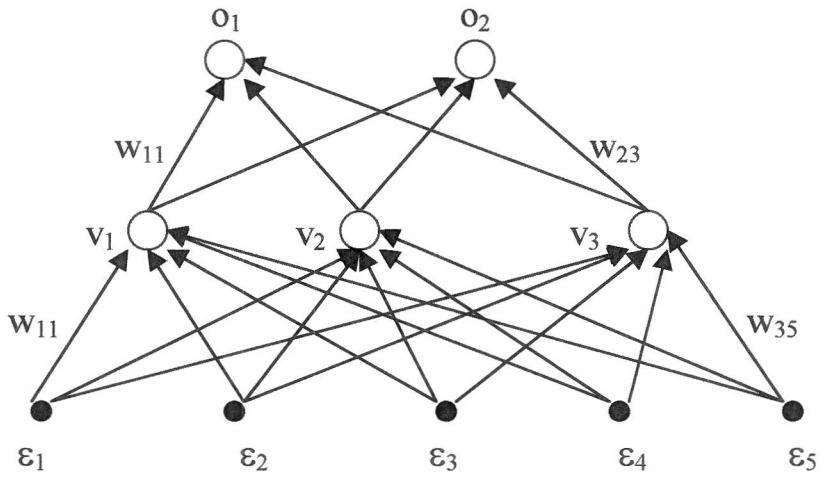


Figura 4.1: Perceptrón con 5 unidades de entrada, 3 unidades ocultas y 2 unidades de salida.

Los pasos a seguir en el algoritmo de Backpropagation son los siguientes:

1. Inicializar los pesos en forma aleatoria con valores pequeños.

2. Seleccionar un patrón ξ_κ^μ y presentarlo al nivel de entrada (m=0) tal que:

$$V_\kappa^0 = \xi_\kappa^\mu \text{ para todo } \kappa$$

3. Propagar la señal a través de la red:

$$V_i^m = g(h_i^m) = g\left(\sum_j w_{ij}^m V_j^{m-1}\right)$$

para cada i y m hasta que se hayan calculado las salidas finales V_i^M.

4. Calcular los deltas para el nivel de salida.

$$\delta_i^M = g'(h_i^M)\ [\xi_i^\mu - V_i^M\ ]$$

comparando las salidas actuales V_i^M con las salidas deseadas ε_i^μ para el pattern μ que se está considerando.

5. Calcular los deltas para los niveles anteriores, propagando los errores hacia atrás:

$$\delta_i^{m-1} = g'(h_i^{m-1}) \sum_j w_{ji}^m \delta_j^m$$

para m = M, M-1, …., 2 hasta que se hayan calculado los deltas para todas las unidades.

6. Usar:

$\Delta w_{ij}^m = -\eta \delta_i^m V_j^{m-1}$ (η es el parámetro de la velocidad del aprendizaje)

para actualizar todas las conexiones de acuerdo a $w_{ij}^{new} = w_{ij}^{old} + \Delta w_{ij}$

7. Volver al paso 2 y repetir para el próximo pattern.

8. Se define la medida de error promedio o función de costo como:

$$E[w] = 1/2 * \sum_{\mu\, i} [\xi_i^\mu - o_i^\mu]^2$$

Si el valor de E [w] es mayor al deseado repetir para todos los patrones de entrada a partir del paso 2.

La función de activación ***g(h)*** es normalmente una sigmoide que debe ser derivable y, generalmente es deseable que se sature en ambos extremos para que las salidas sean valores con un rango fijo. Además es conveniente que la derivada de la función se pueda expresar en términos de la función en sí misma.

Para una salida en el rango [0,1] se puede usar:

$$g(h) = 1 / (1 + \exp(-2\beta h))$$

$$g'(h) = 2\beta g (1 - g)$$

Para una salida en el rango [-1,1] se puede usar:

$$g(h) = \tanh(\beta h)$$

$$g'(h) = \beta (1 - g^2)$$

4.2 Descripción de variantes al algoritmo básico de Backpropagation

El algoritmo básico de Backpropagation es demasiado lento para converger. Por ello, se propusieron muchas variantes para hacerlo más rápido.

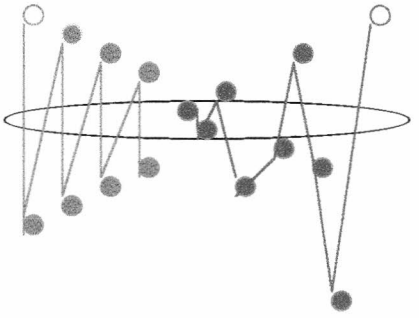
Con momentos:

Esta técnica agrega al algoritmo básico de Backpropagation un *término de momentos.*
La idea es dar a cada conexión w_{pq} una *inercia o momento* para que tienda a cambiar en la dirección del promedio de la pendiente, en lugar de oscilar ampliamente con cada pequeño salto. Así el aprendizaje puede alcanzarse sin que ocurran oscilaciones divergentes. Esto se implementa adicionando una contribución del paso anterior del algoritmo para cambiar los pesos actuales.

$$\Delta w_{pq}(t+1) = - \eta \frac{\partial E}{\partial w_{pq}} + \alpha \Delta w_{pq}(t)$$

El parámetro η es llamado *parámetro de velocidad de aprendizaje.*
El parámetro α, denominado *parámetro del término de momentos*, debe estar en el rango [0,1]. [Hertz, 1990]

La *Figura 4.2* que a continuación se muestra esquematiza el descenso de gradientes en la superficie cuadrática:



A la izquierda se observa la evolución sin la técnica de momentos (α= 0), mientras que a la derecha α = 0.5.

Figura 4.2: Descenso de gradientes en la superficie cuadrática.

Parámetros Adaptivos:

No resulta una tarea fácil elegir cuáles son los valores apropiados para los parámetros η y α para un problema particular.
Además, los mejores valores al comienzo del entrenamiento pueden no ser tan buenos a medida que progresa el mismo. Por esta razón, varios autores han sugerido el ajuste automático de parámetros a medida que avanza el entrenamiento [Cater, 1987], [Franzini, 1987], [Vogl, 1988], [Jacobs, 1988].
El enfoque consiste en chequear si la actualización de un peso en particular produjo que la función de costo decreciera. Si no ocurrió esto, el proceso se excedió y η debería ser menor.
Por otro lado si varios pasos hicieron decrecer la función de costo quizás resulte demasiado conservador y se podría tratar de incrementar η.
Lo mejor sería incrementar η con una constante y disminuirlo geométricamente para permitir un rápido descenso cuando sea necesario.
El esquema es el siguiente:

$$\Delta\eta = \begin{cases} +a & \text{si } \Delta E < 0 \text{ consistentemente.} \\ -b\eta & \text{si } \Delta E > 0. \\ 0 & \text{en otro caso.} \end{cases}$$

Con a,b constantes apropiadas y ΔE igual al cambio de la función de costo. El término "consistentemente" implica, por ejemplo, tener en cuenta los últimos k pasos. Cuando un mal paso decrementa η, es posible deshacer el paso y poner $\alpha=0$ hasta llegar a un buen paso.
[Cater, 1987] propuso tener η^u parámetros, uno para cada pattern u.
[Jacobs, 1988] sugirió un η_{pq} para cada conexión pq.

***Mínimos locales*:**

El descenso de gradientes y demás técnicas de optimización pueden estacionarse en algún mínimo local de la función de costo.
El tamaño de los pesos iniciales aleatorios resulta muy importante.
Si son valores muy grandes la sigmoide se saturará desde el principio y el sistema se estacionará en un mínimo local cercano al comienzo.
Una estrategia sería elegir pesos random tal que la magnitud de una entrada h_i a la unidad i sea menor al valor 1. Esto se logra tomando los pesos w_{ij} del orden de $1/\sqrt{k_i}$ donde k_i es el número de conexiones que entran a la unidad i.
Un tipo común de mínimo local es aquel que no es muy profundo y basta una pequeña fluctuación random o *ruido* para salirse de este mínimo.
Una manera de introducir ruido es elegir los patrones del conjunto de entrenamiento en orden aleatorio, ya que si se eligieran cíclicamente en el mismo orden se llegaría a un estancamiento, dado que se produce una serie de cambios en los pesos que se van cancelando. También es posible introducir ruido explícitamente cambiando aleatoriamente los pesos con pequeños valores o incorporando el ruido directamente en las entradas independientemente en cada presentación, aunque esto último puede reducir considerablemente el proceso de aprendizaje.

4.3 Experiencias y resultados

Este estudio se enfoca desde dos ángulos. Por un lado, se estudia el comportamiento del algoritmo en la etapa de entrenamiento comparando distintas variantes y el impacto de las mismas en los resultados. Las pruebas consistieron en la variación de los diferentes parámetros del algoritmo básico y la aplicación de las variantes explicadas anteriormente.

Por otro lado, se debe analizar la estructura o arquitectura de la red. Esto consiste en estudiar la cantidad de niveles ocultos, cantidad de unidades por nivel, qué capacidad de aprendizaje tiene cada estructura y qué capacidad de generalización posee.

Se definen los dos conceptos de comparación a analizar como:

- **CAPACIDAD DE APRENDIZAJE**: es la capacidad que tiene una red neuronal para aprender los ejemplos que se le presentan. En esta implementación esta característica se mide con el **Error de Entrenamiento.** Este error se calcula de la siguiente manera:

$$\textbf{Error de Entrenamiento} = \frac{\textbf{Cantidad de patterns mal} * 100}{\textbf{Total de patterns del conjunto de entrenamiento}}$$

Donde *Cantidad de patterns mal* son los patterns que, en la etapa de entrenamiento, resultaron calificados como confiables cuando en realidad eran no confiables o viceversa.

Este error se puede calcular al final de cada época del algoritmo. A los efectos de comparar dos entrenamientos de una red, se obtiene el mínimo valor del error a lo largo de un número determinado de épocas.

- **CAPACIDAD DE GENERALIZACIÓN**: es la capacidad que tiene una red neuronal para generalizar lo aprendido sobre el total de ejemplos que no estuvieron presentes en el entrenamiento. Esta capacidad es medida con el **Error de Testeo** que se calcula como sigue:

$$\textbf{Error de Testeo} = \frac{\textbf{Cantidad de patterns mal en el testeo} * 100}{\textbf{Total de patterns del conjunto de testeo}}$$

donde *Cantidad de patterns mal* son los patterns que resultaron calificados en el testeo general como confiables cuando en realidad eran no confiables o viceversa.

Al igual que el error de entrenamiento, este error se puede calcular al final de cada época del algoritmo, utilizando los pesos resultantes del entrenamiento. A los efectos de comparar la capacidad de generalización de dos corridas

distintas, se calcula el mínimo valor del error a lo largo de un número determinado de épocas.

Es muy importante que la red óptima sea determinada en base a la performance de la validación del conjunto de testeo (capacidad de generalización) y no en cuanto a la performance de los datos del entrenamiento (capacidad de aprendizaje), por lo tanto, una vez seleccionados los mejores valores de los parámetros en el entrenamiento es importante obtener la mejor arquitectura en relación a la capacidad de generalización de la red.

Después de un gran número de épocas en el algoritmo de Backpropagation, en muchos casos se produce un sobre-entrenamiento en la red. Esto significa que el error de testeo, aunque oscilante, crece a medida que avanza el número de épocas, mientras el error de entrenamiento sigue disminuyendo o se mantiene constante. El siguiente gráfico muestra este fenómeno para un entrenamiento en particular:

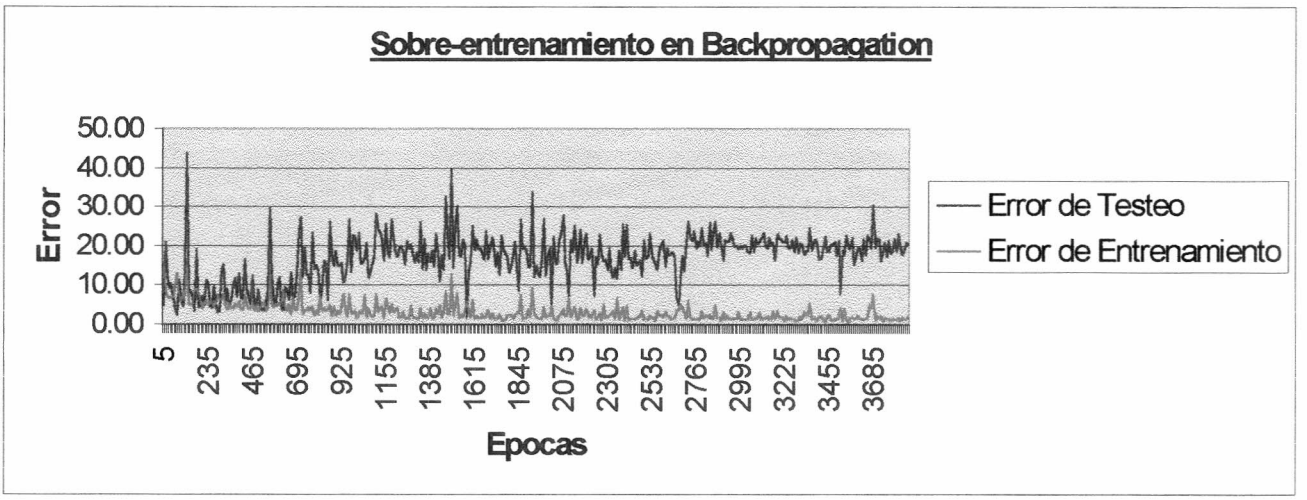


Figura 4.3: Evolución de errores al sobre-entrenar con Backpropagation.

El sobre-entrenamiento se produce porque la red aprende demasiado específicamente el conjunto de entrenamiento y pierde capacidad de generalizar los resultados al resto de los datos. Si se presenta este caso, debería estudiarse el entrenamiento antes de llegar al sobre-entrenamiento de la red.

En la implementación se utiliza la tangente hiperbólica en la función de activación, por lo tanto, los valores devueltos por la red se encuentran en el

rango [-1,1]. Por esta razón, la variable respuesta también se adaptó a estos valores: -1 significa individuo confiable y 1 indica individuo no confiable.

Para determinar la respuesta de la red en cada ejemplo se estableció como punto de corte el valor 0. Un ejemplo es considerado un caso no confiable cuando el valor de salida excede el valor 0 y confiable cuando la salida está por debajo de ese valor. Para lograr resultados precisos los casos que se encuentren en el límite pueden ser tratados particularmente por el experto humano.

4.3.1 Resultados del estudio del algoritmo

Las experiencias realizadas con el algoritmo básico de Backpropagation arrojaron los siguientes resultados para un conjunto de entrenamiento de 500 patterns (260 ejemplos de confiables y 240 de no confiables) y una arquitectura de red fija:

Variación del parámetro β:

El parámetro η se mantiene fijo en 0,2 y se compara el mínimo % de Error de Entrenamiento (% de ejemplos no aprendidos) a lo largo de 3500 épocas.

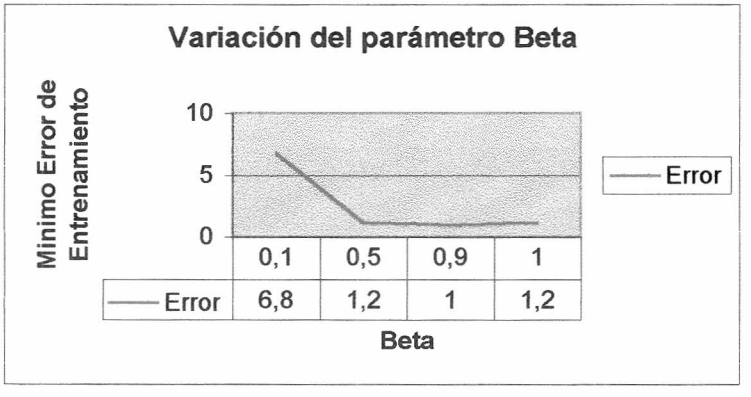


Figura 4.4: Variación del parámetro Beta según las experiencias.

Se desprende de estas experiencias que con el algoritmo básico de Backpropagation seteado con valores de β cercanos pero menores a 1, el error obtenido es el menor.

Estudiando el comportamiento de los distintos entrenamientos se puede notar que con valores de β muy pequeños, el aprendizaje se vuelve muy lento con un error siempre decreciente y por el contrario, con valores grandes de β, el aprendizaje es más rápido y el error oscilante.

El parámetro β regula la pendiente de la función de activación. Cuando el valor de β es pequeño la derivada de la sigmoide en valores cercanos a 0 toma también valores pequeños y, por lo tanto el aprendizaje se torna muy lento. Si se aumenta el valor de β la evolución es más rápida.

Variación del parámetro η (velocidad de aprendizaje):

El valor de β se mantiene fijo en 0,9 y se compara el mínimo % de Error de Entrenamiento (% de ejemplos no aprendidos) a lo largo de 3500 épocas.

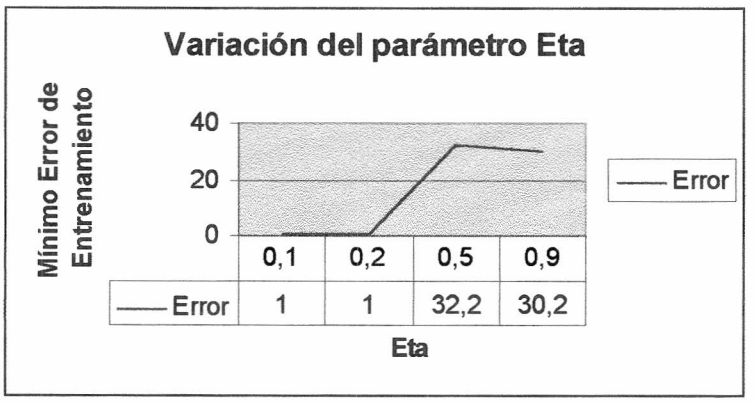


Figura 4.5: Variación del parámetro Eta según las experiencias.

Aquí se observa que valores pequeños del parámetro η contribuyen a obtener un error aceptable. Se aprecia también que dicho error aumenta considerablemente con η muy alto (se recuerda que la bibliografía recomienda el rango [0.05,0.25] para η) [Freeman-Skapura, 1992]. Este parámetro regula directamente la magnitud de los cambios en los pesos. Cuando este valor es muy alto, la variación de los pesos es muy brusca y no es posible alcanzar el mínimo buscado.

Esto ocurre porque η regula toda la velocidad del proceso de aprendizaje, a diferencia de β que lo hace en las primeras etapas y luego el efecto se va moderando.

Calculando el gradiente se está buscando un mínimo local de la función de costo. Si el η es chico, se baja progresivamente en una misma curva, si es demasiado grande se corre el riesgo de saltar a otro tramo y no encontrar el mínimo local como lo muestra la *Figura 4.6*:

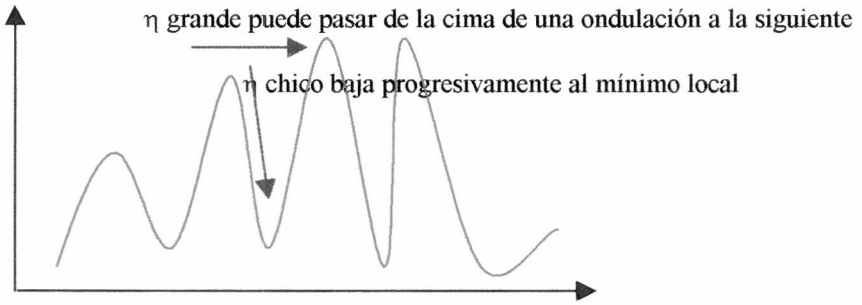


Figura 4.6: Efectos no deseados en la selección del valor para η.

La función de costo tiene una amplitud promedio y se puede estudiar cuál es el η óptimo según esa amplitud. Esto se ve en la *Figura 4.7*:

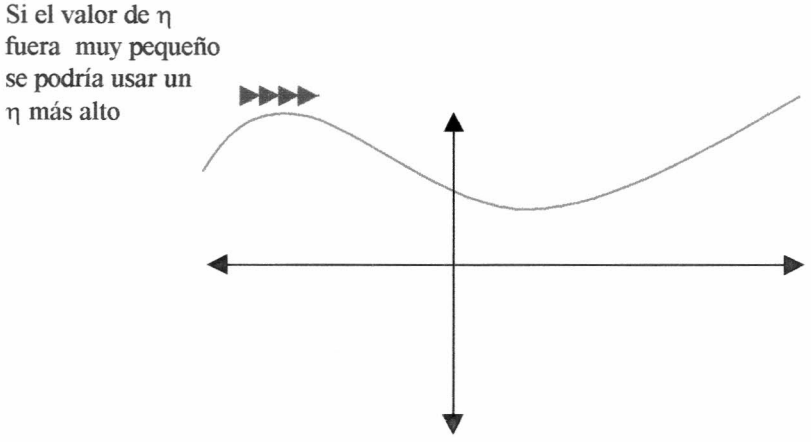


Figura 4.7: Selección óptima para el valor de η.

Parámetros adaptivos:

Se incluye en el algoritmo esta variante implementada como se detalla:

$$\eta = \begin{cases} \eta + a & \text{si en las ultimas } k_1 \text{ épocas, el error fue decreciente.} \\ \eta - (b * \eta) & \text{si en las ultimas } k_2 \text{ épocas, el error fue creciente.} \end{cases}$$

Con $k_1, k_2 > 0$.

Los valores apropiados para las constantes a y b fueron obtenidos luego de observar distintos comportamientos en la variación de η a lo largo de varias épocas.

Con respecto a los valores de k_1 y k_2, estos fueron seteados con valores no muy altos puesto que el error oscilaba muy frecuentemente.

Así se llegó a la configuración apropiada de valores: $a = 0.01$, $b = 0.03$, $k_1 = 4$ y $k_2 = 3$, para un valor inicial de $\eta = 0.2$.

Agregado de la técnica de momentos y variación del parámetro α:

En todos los casos se mantuvo fijo el parámetro β con valor 0.4 y el parámetro η fue adaptado según la variante de parámetros adaptivos.

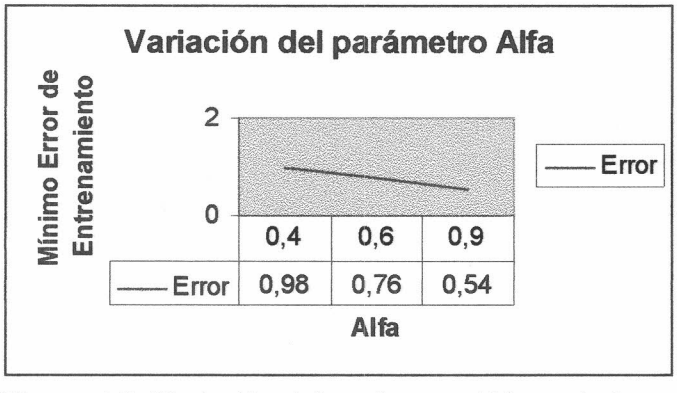


Figura 4.8: Variación del parámetro Alfa según las experiencias.

Se observa que a valores más grandes de α el porcentaje de error de entrenamiento que se obtiene es menor. Esto fundamenta el uso de la variante de momentos del algoritmo de Backpropagation.

Mínimos locales:

Para evitar los mínimos locales, en esta implementación se inicializan los pesos con valores del orden de $1/\sqrt{k_i}$ donde k_i es el número de conexiones que entran a la unidad i. Además, se presentan los ejemplos a la red en un orden aleatorio que varía en cada época.

Analizando los resultados de las distintas variantes al algoritmo básico de Backpropagation se observa que el mínimo error de entrenamiento fue 0.54 obtenido en la variante de Momentos y Parámetros Adaptivos conjuntamente.

4.3.2 Análisis de la arquitectura de la red

En este punto se estudia qué importancia tiene el número de conexiones en el entrenamiento de una red. Suponiendo que la variante del algoritmo de Backpropagation elegida es la mejor, cabe preguntarse:

- ¿Resulta mejor poner uno o dos niveles ocultos?
- ¿Qué es más conveniente?, ¿tener un nivel oculto con muchas unidades o más niveles ocultos con pocas unidades?

4.3.2.1 Comparación de la capacidad de aprendizaje

Se comparó el error de entrenamiento de diferentes experiencias con arquitecturas de un nivel oculto y dos niveles ocultos. La cantidad de épocas se mantuvo constante.

Las experiencias fueron seleccionadas por haber optimizado el comportamiento del algoritmo según los criterios tomados en el estudio anterior. Fueron realizadas sobre un conjunto de entrenamiento de 922 patrones (443 ejemplos de confiables y 479 de no confiables).

En las tablas que a continuación se verán, se informa la cantidad total de conexiones de cada red. Este número se calcula, para una red de n niveles, como sigue:

$$\text{Total de conexiones} = \sum_{i=2}^{n} (c_{i-1} * c_i)$$

donde

c_i: cantidad de unidades del nivel i.
n: cantidad de niveles contando el de entrada y el de salida.
Nivel 1 es el nivel de entrada y $Nivel_n$ es el nivel de salida.

- Arquitectura de un sólo nivel oculto:

La *Tabla 4.1* muestra las experiencias realizadas con una arquitectura de un nivel oculto, ordenadas por cantidad de conexiones:

Unidades Nivel oculto	Conexiones	Mínimo % de Error de Entrenamiento
1	12	2,49
2	24	1,95
4	48	1,84
6	72	0,65
12	144	0,54
15	180	0,33

Tabla 4.1: Resultados obtenidos con una arquitectura de un nivel oculto.

En esta tabla se puede observar que el mínimo porcentaje de error de entrenamiento se obtiene al entrenar con mayor número de conexiones. Es decir, que en una arquitectura de un nivel oculto, la red aprende mejor cuanto más unidades posee.

- Arquitectura de dos niveles ocultos:

La *Tabla 4.2* muestra las experiencias que fueron realizadas con una arquitectura de dos niveles ocultos.

Si se ordenan las experiencias seleccionadas según la cantidad total de conexiones, se obtiene la siguiente tabla (donde Unidades indica las cantidades de unidades en los dos niveles ocultos):

Unidades	Conexiones	Mínimo % de Error de Entrenamiento
1 y 1	13	3,60
2 y 2	28	1,60
4 y 5	69	1,00
10 y 7	187	0,40

Tabla 4.2: Resultados obtenidos con una arquitectura de dos niveles ocultos.

En esta tabla se observa que el mínimo porcentaje de error de entrenamiento se obtiene al entrenar con mayor número de conexiones. Es decir, que en una arquitectura de dos niveles ocultos, la red aprende mejor cuanto más unidades posee, al igual que lo afirmado para arquitecturas de un nivel oculto.

Por lo tanto, para el problema de predecir si una persona resulta confiable o no en términos comerciales en base a su historia y de acuerdo a lo experimentado, en una arquitectura de un sólo nivel oculto, 15 unidades son suficientes para lograr un equilibrio en términos de aprendizaje y costo del entrenamiento. Esto mismo se afirma para una arquitectura de dos niveles ocultos, con la cantidad de 10 y 7 unidades por cada nivel oculto respectivamente.

4.3.2.2 Comparación de la capacidad de generalización

Para testear el comportamiento de la red se analiza el resultado de aplicar los pesos obtenidos en el entrenamiento al total del conjunto de testeo (14961 patrones).

- Arquitectura de un sólo nivel oculto:

Nuevamente se presentan las experiencias realizadas con una arquitectura de un nivel oculto para comparar ahora el error de testeo.

Unidades	Conexiones	Mínimo % de Error de Testeo
1	12	1,13
2	24	1,07
4	48	1,05
6	72	1,04
12	144	1,12
15	180	1,11

Tabla 4.3: Resultados obtenidos con una arquitectura de un nivel oculto.

Se observa que el error de testeo es mayor en los extremos. El mínimo error de testeo se obtuvo con una cantidad intermedia de conexiones, en este caso con 72 conexiones.

Puede notarse que la mejor experiencia respecto del error de entrenamiento (180 conexiones), no es la misma que teniendo en cuenta el error de testeo (72 conexiones). Esto ocurre porque en las arquitecturas con muchas conexiones la red aprende específicamente el conjunto de entrenamiento y no logra una correcta generalización (overfitting).

- Arquitectura de dos niveles ocultos:

Si se analizan nuevamente las experiencias con dos niveles ocultos pero ahora teniendo en cuenta el mínimo porcentaje de error de testeo obtenido, los mejores resultados se muestran en la *Tabla 4.4*:

Unidades	Conexiones	Mínimo % de Error de Testeo
1 y 1	13	2,61
2 y 2	28	1,07
4 y 5	69	1,87
10 y 7	187	1,85

Tabla 4.4: Resultados obtenidos con una arquitectura de dos niveles ocultos.

Aquí se ve que el error de testeo es mayor en los extremos, es decir cuando la red cuenta con muchas o con muy pocas conexiones, encontrándose el mínimo con 28 conexiones.

4.3.2.3 Arquitectura de un nivel oculto versus arquitectura de dos niveles ocultos

Se comparan aquí los resultados de las experiencias hechas sobre una arquitectura de una capa oculta y las realizadas sobre una arquitectura con dos niveles ocultos.

Si se agrupan por cantidad de capas ocultas y se calculan los promedios y mínimos, se obtienen los siguientes resultados:

Niveles Ocultos	Promedio Error Entr.	Promedio Error Testeo	Mínimo Error Entr.	Mínimo Error Testeo
1	1,30	1,08	0,33	1,04
2	1,70	1,85	0,40	1,07

Tabla 4.5: Resumen de las experiencias realizadas con distintas arquitecturas.

De esta tabla se desprende que los mínimos porcentajes de ambos errores y los mejores promedios corresponden a la arquitectura de un nivel oculto. Además, se compararon las experiencias de acuerdo a un número similar de conexiones pero distribuidas o no en dos capas ocultas:

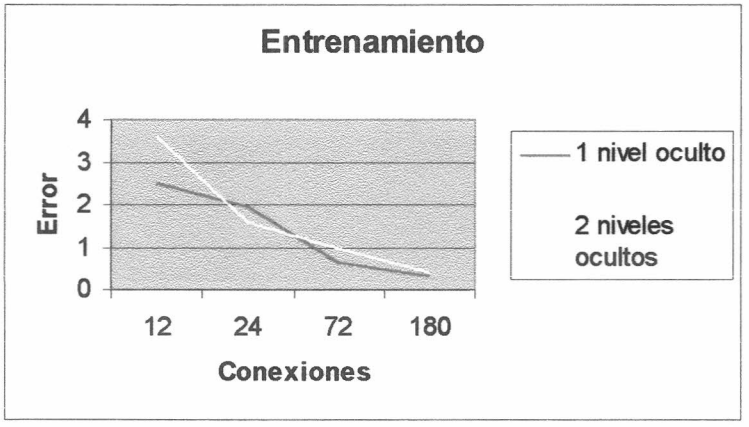


Figura 4.9: Cuadro comparativo del Error de Entrenamiento para arquitecturas de uno y dos niveles ocultos.

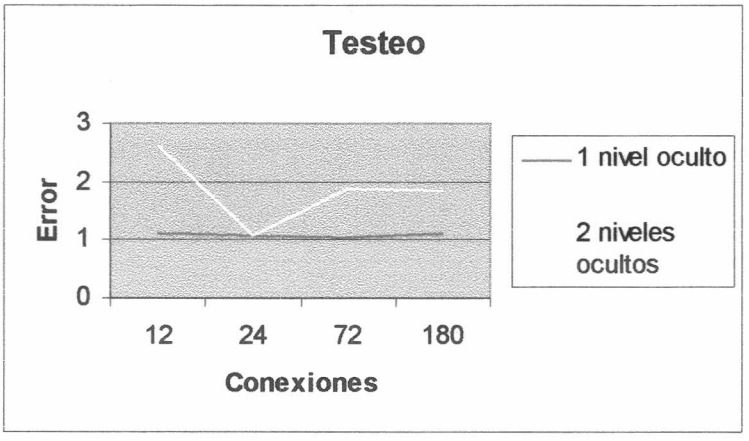


Figura 4.10: Cuadro comparativo del Error de Testeo para arquitecturas de uno y dos niveles ocultos.

En estos gráficos se observa que en el entrenamiento *(Figura 4.9)* el comportamiento de arquitecturas con uno o dos niveles ocultos es muy similar. En el testeo sobre el total de los ejemplos disponibles *(Figura 4.10)*, los mínimos errores en ambos casos se dan entre las 24 y las 72 conexiones. El error de las arquitecturas de 1 nivel oculto se mantiene casi constante con las diferentes cantidades de unidades, mientras que el error en el caso de redes de 2 niveles ocultos es marcadamente menor hacia el centro en el eje de conexiones, encontrando un punto de inflexión cercano a las 24 conexiones

Se concluye en base a los resultados logrados que los mínimos errores (de entrenamiento y testeo) se presentan en arquitecturas de un sólo nivel oculto.

4.3.3 Análisis del tamaño del conjunto de entrenamiento

El tamaño del conjunto de entrenamiento es un criterio importante para tener en cuenta al entrenar una red. También lo es la selección de las entradas que conforman el conjunto de entrenamiento. Esto significa, no incluir demasiados ejemplos muy específicos, elegir la mejor proporción de casos representativos de cada clase, etc. En el problema de esta tesis y de acuerdo a lo sugerido en la bibliografía, se ha elegido una proporción equilibrada en cuanto a los ejemplos confiables y no confiables en el conjunto de entrenamiento a pesar que en la realidad la cantidad de casos confiables supera ampliamente la de no confiables. Se recuerda que el porcentaje de no confiables es sólo el 5% del total de ejemplos.

Se compararon experiencias realizadas sobre idénticas arquitecturas pero con dos conjuntos de entrenamiento de distinto tamaño. Se extrajeron dos conjuntos de datos:

C1: 500 patrones (260 ejemplos de confiables y 240 de no confiables)
C2: 1000 patrones (550 ejemplos de confiables y 450 de no confiables).

Con estos dos conjuntos se entrenó una red de idénticas características y se obtuvieron los siguientes resultados:

Patrones	Mínimo Error Entr.	Mínimo Error Testeo
500	0,4	1,85
1000	0,4	1,09

Tabla 4.6: Resultados obtenidos variando el tamaño del conjunto de entrenamiento.

Así se observa que en la etapa de entrenamiento, en ambos casos, se lograron similares resultados mientras que en la etapa de testeo el mínimo porcentaje de error corresponde al conjunto de entrenamiento de mayor tamaño.

Es decir que al incrementar el tamaño del conjunto de entrenamiento se obtuvieron mejores resultados de generalización.

4.3.4 Selección de la mejor configuración de red

En las experiencias realizadas se tomaron como parámetros de comparación los errores de entrenamiento y testeo, que se definieron como la cantidad de casos no acertados sobre el total. Esta no es la única posibilidad de selección del mejor "momento" de la red.

Para guardar los pesos de la configuración con mejores resultados, surge el dilema de decidir qué resultado se considera superior a otro.

Si sólo se considera que en el testeo se acierte la mayor cantidad de casos basta con guardar la configuración que minimice la cantidad de casos equivocados. Pero como en la situación particular que se estudia, la cantidad de casos confiables es el 96% del total, esta opción beneficia la precisión de respuesta en los casos confiables y no tiene en cuenta el porcentaje de error sobre los no confiables. Si, en cambio, se minimiza la cantidad de no confiables erróneos, sube demasiado la cantidad de confiables no acertados.

La "mejor" configuración depende de la pérdida que ocasione considerar casos confiables como no confiables y viceversa. Por ejemplo, se puede analizar dicha pérdida en el contexto particular de una entidad bancaria que utiliza este sistema para determinar si le otorga un crédito hipotecario a una persona o no. Al considerar a un confiable como no confiable el banco está perdiendo la ganancia completa que implicaba otorgar el crédito. En cambio al considerar un no confiable como confiable la persona va a dejar de pagar en algún momento y el banco sólo perderá los intereses desde ese momento en adelante, ya que el capital lo recupera al rematar el inmueble hipotecado. Suponiendo que la pérdida es menor en el caso de equivocarse en un confiable, se debería seleccionar la configuración con menor cantidad de confiables equivocados.

Para tener la posibilidad de guardar distintas configuraciones se define una función de selección de la siguiente manera:

$$Fs = (p * mc) + ((1 - p) * mn)$$

donde p es el peso que se otorga a los casos confiables, mc es la cantidad de casos confiables no acertados, mn es la cantidad de casos no confiables no acertados.

Como prueba se seleccionaron 6 valores de p y se guardaron las 6 configuraciones cuyo testeo minimizaba el valor de Fs. La corrida se realizó sobre una red de 12 nodos distribuidos en un sólo nivel oculto, con la técnica de momentos. El conjunto de entrenamiento contenía 922 casos (443 ejemplos de confiables y 479 de no confiables) y el de testeo 8780 casos. Los resultados fueron los siguientes:

	P	% No Conf. Mal	% Conf. Mal	% Total Mal	Epoca
1	0,96	25,96	0,13	1,05	90
2	0,75	25,96	0,13	1,05	90
3	0,50	25,96	0,13	1,05	90
4	0,25	13,68	1,20	1,65	300
5	0,04	4,21	4,14	4,14	1940
6	0,01	1,40	14,62	14,14	750

Tabla 4.7: Resultados obtenidos para distintos valores de p.

De los datos de la *Tabla 4.7* se deduce que en las configuraciones 1, 2, y 3 los resultados fueron idénticos y se privilegia el acierto de los confiables. En las número 4 y 5 se logran distintos puntos de equilibrio entre las dos clases, y en la 6 se privilegia el acierto de los no confiables, causando un aumento del porcentaje de confiables no acertados.

También se puede observar que a la configuración con mayor acierto en los confiables se llega muy rápidamente (época 90), luego la red comienza a tender a configuraciones con mayor precisión en los no confiables, y por último a las más equilibradas.

En la *Figura 4.11* se muestra la comparación de las distintas configuraciones:

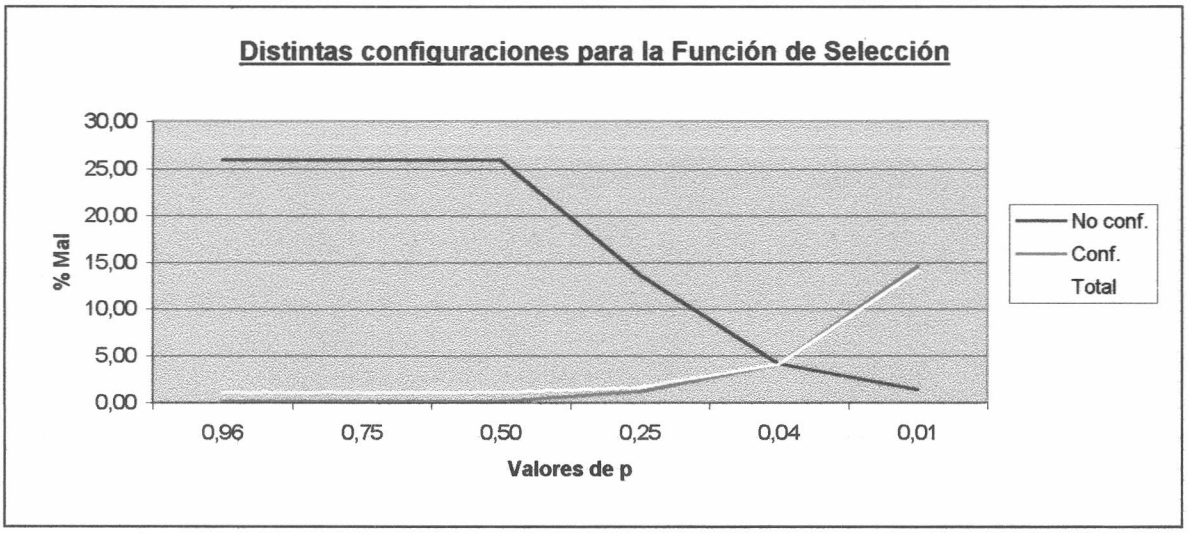


Figura 4.11: Comparación de configuraciones para la Función de Selección.

Se puede observar que para p=0,04 el % de ejemplos sin acierto es similar para confiables, no confiables y total, mientras que en los extremos aprende mejor una de las 2 clases de patrones.

4.4 Explicación de respuestas

Una limitación que presentan las RNA radica en su dificultad para poder otorgar una explicación de la salida calculada por la red, es decir, explicar las respuestas.
Si se analiza este aspecto sobre un Perceptrón Multicapa entrenado con el algoritmo de Backpropagation, se observa que es factible explicar la salida de la red analizando dos ítems:

i) La evolución de los pesos que conectan las entradas con la capa oculta.
ii) La varianza (dispersión) de esos pesos en el conjunto de entrenamiento.

De esta manera se puede estimar la incidencia de una cierta variable en la respuesta, algo que resulta no trivial.

4.5 Conclusiones

En base a las pruebas realizadas para determinar si un potencial deudor resulta confiable o no confiable de acuerdo a su comportamiento comercial en el pasado, la mejor experiencia lograda con el algoritmo de aprendizaje

supervisado de Backpropagation (con momentos y parámetros adaptivos) fue realizada utilizando un conjunto de entrenamiento de 922 ejemplos con un error del 1.04% al generalizar los resultados a un conjunto de 15000 ejemplos. Es decir que la eficiencia en la predicción fue del 98.96%.

De acuerdo con los resultados obtenidos y como conclusión en cuanto al estudio del algoritmo, se afirma que para el problema planteado la mejor variante resultó ser Backpropagation con Momentos y Parámetros Adaptivos. También, en este algoritmo, se incluyen técnicas para evitar mínimos locales.

En cuanto a la capacidad de aprendizaje, se puede decir que la red aprende mejor cuanto más unidades posee, tanto en una arquitectura de un sólo nivel oculto como en una arquitectura de dos niveles ocultos. La desventaja de una arquitectura con demasiadas unidades radica en que aunque mejora el aprendizaje, el tiempo de entrenamiento crece considerablemente. Agregar más unidades de las utilizadas equivale a una ínfima mejora en el proceso de aprendizaje con el riesgo de incrementar el tiempo en la etapa de entrenamiento.

Con respecto a la capacidad de generalización se prueba que la red generaliza mejor cuando posee un número moderado de conexiones. Esto se debe a que en el proceso de agregar unidades, a partir de cierto momento, el incremento obtenido en la capacidad de representación o aproximación no logra contrarrestar el aumento de la complejidad del espacio de configuraciones que crece en una dimensión por cada unidad que se agrega. Esto es observable tanto en una arquitectura de un sólo nivel oculto como en una arquitectura de dos niveles ocultos.

Si lo que se quiere evaluar es la arquitectura a elegir se puede afirmar, en base al mínimo error obtenido, que para el problema de este trabajo es conveniente utilizar una arquitectura de un sólo nivel de unidades ocultas.

Como conclusión en base al tamaño del conjunto de entrenamiento se puede observar que las capacidades de la red fueron perfeccionadas cuando se incrementó al doble la cantidad de patrones de entrada en el conjunto de entrenamiento. Lógicamente, este último conjunto de entrenamiento es más completo que el primero y por lo tanto se logró una mejor generalización.

La Función de Selección también es una medida para comparar resultados. Por ello, la mejor configuración para esta función depende de las necesidades del usuario de la red y se podrá elegir la configuración que más convenga de las presentadas, o buscar otras variando el valor de p.

Por último, con un aprendizaje supervisado en Backpropagation, es posible aunque costoso, encontrar una respuesta a la salida de la red analizando los pesos.

Capítulo 5

5. APRENDIZAJE NO SUPERVISADO

Aquí se realiza una introducción al tipo de aprendizaje no supervisado y sus diferencias con el aprendizaje supervisado. Luego se explica el aprendizaje competitivo y un algoritmo de este tipo: el Algoritmo de Kohonen y se incluyen los resultados obtenidos en cuanto a este tipo de aprendizaje y las conclusiones elaboradas.

A diferencia del entrenamiento con aprendizaje supervisado, donde se indica a la red cuál es la salida correcta para cada entrada, en aprendizaje no supervisado no hay respuestas del entorno para decidir si las salidas de la red son acertadas o no. La red debe descubrir patrones, características, regularidades, correlaciones o categorías en los datos de entrada y asignarles una determinada codificación que los clasifique en la salida. Para realizar este aprendizaje las unidades y conexiones de la red deben tener capacidad de auto-organización.

Sólo se puede aplicar aprendizaje no supervisado cuando hay redundancia en los datos de entrada, ya que es necesario tener muchos ejemplos de cada clase para poder identificarla.

El tipo de patrón que una red no supervisada detecta en los datos de entrada depende de la arquitectura de la misma. Existen distintas posibilidades en cuanto al formato de salida de la red:

1. Familiaridad: una salida con valor continuo puede indicar cuán familiar o típico es un patrón con respecto a los observados anteriormente. La red va aprendiendo gradualmente qué es típico.
2. Análisis de componentes principales: si se extiende el caso previo a un conjunto de unidades se construye un conjunto de ejes donde se mide la similitud de un patrón con los anteriores. En estadística esto se realiza utilizando las direcciones de los autovectores de la matriz de correlación de los ejemplos de entrada.
3. Aprendizaje competitivo: la salida es un conjunto de valores binarios que se prenden de a uno por vez y cada uno indica la pertenencia a una determinada clase.
4. Prototipos: la red devuelve un ejemplo que caracterice a cada clase en vez de decir a cuál pertenece.

5. Codificación: la salida es una versión codificada de la entrada con menos bits y mantiene la mayor cantidad de información posible. Esto puede usarse para compresión de datos.
6. Mapeo de características (Feature Mapping): si las unidades de salida tienen un ordenamiento geométrico (por ejemplo una matriz) y se prenden una a la vez, se puede realizar una representación topográfica del conjunto de entrada, donde ejemplos similares hacen encender unidades cercanas.

Cuando el objetivo del aprendizaje es que se encienda una y sólo una unidad de salida, luego del proceso de un patrón de entrada, el aprendizaje se llama competitivo.

Las unidades de salida compiten para ser seleccionadas y, por ello se las denomina "winner-take-all" (la ganadora se lleva todo).

Estas redes sirven para categorizar los datos de entrada, es decir, conforman una aplicación de clustering. Patrones similares se clasificarán en la misma categoría y, por lo tanto, deberán activar la misma unidad de salida. Como en todo aprendizaje no supervisado, las clases se forman de acuerdo a las correlaciones de los datos de entrada, ya que las entradas no tienen su correspondiente respuesta o etiqueta.

Las desventajas de este tipo de red son:

- Necesitan una unidad de salida para cada clase.
- No son robustas a degradación o fallas. Si una unidad de salida falla se pierde la clase.
- No pueden representar conocimiento jerárquico. Dos patrones de entrada caen en la misma clase o no pero no hay forma de representar una categoría dentro de otra.
- Como en este tipo de aprendizaje, el problema de decisión se reduce a clustering, existe un costo adicional que se manifiesta en la rotulación manual de los datos (conocimiento externo que debe agregarse a los datos para evaluar el entrenamiento y testeo de la red). Esto se contrapone con el perceptrón multicapa con Backpropagation que al ser de tipo supervisado no exige conocimiento externo.
- La naturaleza no-determinística de la red neuronal se hace notable en este tipo de aprendizaje.

5.1 Descripción del algoritmo de aprendizaje competitivo

Existe un nivel de unidades de salida $\mathbf{O_i}$, cada una totalmente conectada a un conjunto de entradas ε_i con conexiones $\mathbf{w_{ij}} >= 0$. Se considera que las unidades son binarias. Sólo una de las unidades de salida puede activarse por vez. Generalmente la unidad ganadora es la que tiene la entrada más grande.

$\mathbf{h_i} = \sum_j \mathbf{w_{ij}}\, \varepsilon_j = \mathbf{w_i} \cdot \varepsilon$ para el vector de entrada ε

Entonces: $\mathbf{w_{i^*}} \cdot \varepsilon \geq \mathbf{w_i} \cdot \varepsilon \;\; \forall \mathbf{i}$ define a la unidad ganadora **i* con** $\mathbf{O_{i^*} = 1}$.

Si los pesos de cada unidad están normalizados tal que $|\mathbf{w_i}| = 1 \;\forall \mathbf{i}$ entonces la expresión anterior es equivalente a $|\mathbf{w_{i^*}} - \varepsilon| \leq |\mathbf{w_i} - \varepsilon| \;\; \forall \mathbf{i}$.

Esto quiere decir que la unidad ganadora es la que posee el vector de pesos normalizado más cercano al vector de entrada ε.

Existen diversas formas de lograr la característica de que gane sólo una unidad de salida. En una simulación computacional se puede buscar simplemente el máximo $\mathbf{h_i}$. En una red neuronal real se puede implementar una inhibición lateral entre las unidades de salida (cada unidad inhibe a las otras). Distintos esquemas para redes "winner-take-all" fueron propuestos por: [Grossberg, 1976a, 1980], [Feldman & Ballard, 1982], [Lippman, 1987], [Winters & Rose, 1989].

Una red "winner-take-all" implementa un clasificador de patrones usando el criterio definido anteriormente. El problema consiste en cómo encontrar clusters en los datos de entrada y cómo elegir el valor de los pesos para ello.

Se comienza asignando a los pesos valores pequeños y aleatorios para romper cualquier simetría. Luego se aplica un conjunto de patrones de entrada ε^μ a la red en orden aleatorio. Para cada entrada se encuentra la unidad ganadora **i*** y se actualizan los pesos $\mathbf{w_{i^*j}}$ sólo para la unidad ganadora para que $\mathbf{w_{i^*}}$ sea el vector más cercano a ε^μ. Esto hace que la unidad **i*** tenga la probabilidad más alta de ganar para la misma entrada en el futuro. La forma más obvia de hacer esto es:

$\Delta\, \mathbf{w_{i^*j}} = \eta\; \varepsilon^\mu_j$ pero no sirve porque hace crecer los pesos indefinidamente y la misma unidad ganaría en todos los casos.

Otra solución, denominada "Regla de aprendizaje competitivo standard" es: $\Delta w_{i^*j} = \eta (\varepsilon^{\mu}_j - w_{i^*j})$ que mueve w_{i^*} directamente hacia ε^{μ}.

Esta regla se aplica sólo para la unidad ganadora. Suponiendo que $O_{i^*} = 1$ y $O_i = 0$ para $i \neq i^*$ se puede escribir como $\Delta w_{ij} = \eta O_i (\varepsilon^{\mu}_j - w_{ij})$

Aplicando este tipo de aprendizaje se presenta un problema que consiste en que las unidades con pesos lejos de cualquier patrón de entrada nunca ganan y, por lo tanto, nunca aprenden. A estas unidades se las llama "unidades muertas". Esto se puede prevenir de alguna de las siguientes maneras:

- Inicializar los pesos con valores idénticos a ejemplos de entrada para asegurar que estén en el dominio correcto.
- Actualizar los pesos de todos los perdedores pero con un η mucho más chico [Rumelhart & Zipser, 1985], [Grossberg 1987b]. De esta forma una unidad que estuvo siempre perdiendo se moverá gradualmente a la dirección promedio de la entrada hasta que eventualmente gane la competencia. Esto se denomina *leaky learning* (aprendizaje permeable).
- Si las unidades están dispuestas de manera geométrica, por ejemplo en una matriz, se pueden actualizar también los pesos de las unidades vecinas a la ganadora. Esta es la esencia del Algoritmo de Kohonen, también llamado de Mapeo de características.

5.2 Descripción del Algoritmo de Kohonen

Las unidades de salida O_i se disponen en un arreglo (en general de una o dos dimensiones) y cada una está conectada totalmente por medio de los pesos w_{ij} a las entradas.
La diferencia con el aprendizaje competitivo común está en la regla de aprendizaje donde: $\Delta w_{ij} = \eta \Lambda(i,i^*) (\varepsilon_j - w_{ij}) \quad \forall$ **i,j.** [Kohonen,1982, 1989]

La función de vecindad $\Lambda(i,i^*)$ vale **1** para $i=i^*$ y disminuye a medida que crece la distancia $|r_i - r_{i^*}|$ entre unidades **i** e **i*** en el arreglo de salida (r_i es la ubicación de la salida de la unidad **i** en el arreglo de salida). Entonces las unidades cercanas a la ganadora y la ganadora incrementan sus pesos de manera apreciable mientras las más lejanas sienten menos el efecto. El siguiente gráfico explicita la idea de vecindad:

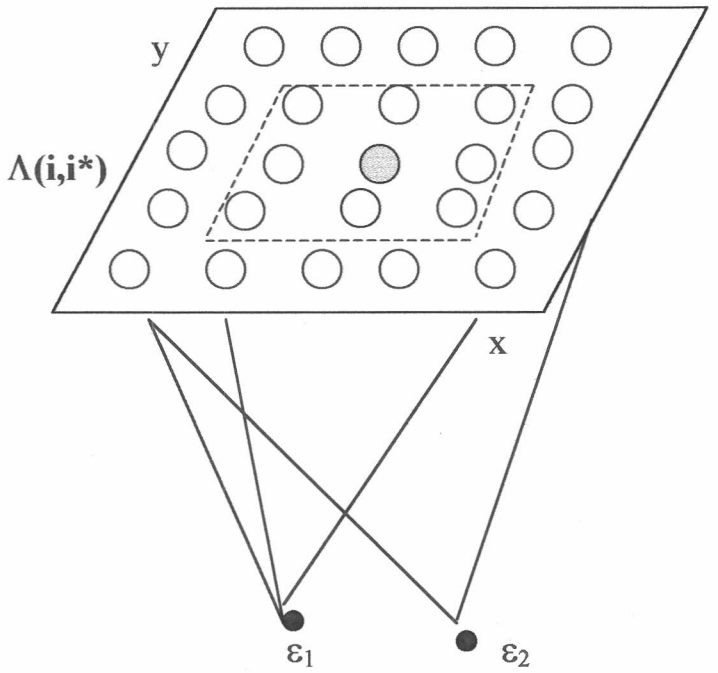


Figura 5.1: Unidades de salida ubicadas en un mapa topográfico

En la *figura 5.1* se ve resaltada la unidad ganadora en la matriz de salida y su correspondiente vecindad denotada **Λ(i,i*)**. Se muestran sólo algunas conexiones entre el nivel de entrada y el de salida aunque ambos niveles están completamente conectados. Los pesos que se actualizan en mayor magnitud son aquellos que corresponden a las conexiones entre las unidades de entrada ε_1 ,ε_2 y las unidades incluidas en la vecindad. De esta manera se brinda información topológica a la red: las unidades cercanas reciben actualizaciones similares y por ende responden a patrones de entrada similares.

Por lo tanto se puede pensar en una especie de red elástica en el espacio de entrada que tiende a parecerse lo más posible a las entradas. La red tiene la topología del arreglo de salida (una línea o un plano).

Para construir un algoritmo en la práctica hay que especificar **Λ(i,i*)** y η. Resulta útil e inclusive es indispensable para la convergencia cambiar estos valores dinámicamente durante el aprendizaje. Se comienza con un rango amplio para **Λ(i,i*)** y un valor de η grande y luego se los reduce gradualmente.

Esto permite que la red se organice rápidamente y luego se vaya adaptando más lentamente a los patrones de entrada.

Una expresión típica para $\Lambda(i,i^*)$ es:

$$\Lambda(i,i^*) = \exp\left(-|r_i - r_{i^*}|^2 / 2\sigma^2 \right)$$

Los parámetros $\eta(t)$ y $\sigma(t)$ decrecen gradualmente con el tiempo y pueden tomar los siguientes valores:

- **1/t**
- **a – bt** (con $b < a/t$ para que la ganancia en la velocidad de aprendizaje no sea negativa y asegurar convergencia en el cálculo del ancho de la vecindad)
- $\alpha \cdot t^{-\alpha}$ $(0 < \alpha \leq 1)$

5.3 Experiencias y resultados

Se implementó el algoritmo de Kohonen con un arreglo de salida de 2 dimensiones (una matriz). Los patrones de entrada contienen los mismos 10 atributos utilizados en el aprendizaje supervisado pero normalizados a una escala [0,1]. La expresión utilizada para la función de vecindad es la descripta anteriormente con $\sigma(t) = 1/t$.

Para analizar la capacidad de aprendizaje de la red y la capacidad de generalización es necesario decidir, una vez finalizado el entrenamiento, qué significa que un ejemplo se encuentre en una determinada celda de la matriz de salida (a la que se denomina cluster). Es decir, se necesita asignar el rótulo "confiable" o "no confiable" a cada par de coordenadas de la matriz.

Como el entrenamiento es no supervisado, es conveniente aclarar que el atributo que indica si el caso analizado resultó confiable o no confiable no fue entrado a la red. En cambio, este atributo fue tomado en cuenta posteriormente a la salida del algoritmo como método para clasificar a los clusters obtenidos en las dos clases que interesan. Para ello, se calculó la proporción de ejemplos de cada clase que caían en un cluster y se asignó al mismo la clase con mayor porcentaje.

De acuerdo con las definiciones dadas en capítulos anteriores de Error de Entrenamiento y Testeo, se redefinen las expresiones teniendo en cuenta los mismos conceptos pero adaptados a este tipo de aprendizaje:

$$\text{Error de Entrenamiento} = \frac{\text{Cantidad de patterns que se ubican en clusters cuya mayoría es de otra clase} * 100}{\text{Total de patterns del conjunto de entrenamiento}}$$

Para calcular la capacidad de generalización se ubica el total de ejemplos disponibles en los clusters de la matriz, utilizando los pesos obtenidos en la etapa anterior, y se compara para cada patrón la clase asignada al cluster correspondiente en la etapa de entrenamiento con la clase correcta.
De esta manera:

$$\text{Error de Testeo} = \frac{\text{Cantidad de patterns que se ubican en clusters con clase inversa} * 100}{\text{Total de patterns del conjunto de testeo}}$$

5.3.1 Análisis de la influencia de la composición del conjunto de entrenamiento

Se estudió la influencia de la proporción de ejemplos confiables y no confiables del conjunto de entrenamiento sobre la capacidad de aprendizaje y la capacidad de generalización para el total de ejemplos y para cada clase por separado. Estos resultados fueron obtenidos al entrenar sobre un conjunto de 600 ejemplos en una matriz de 10x10.

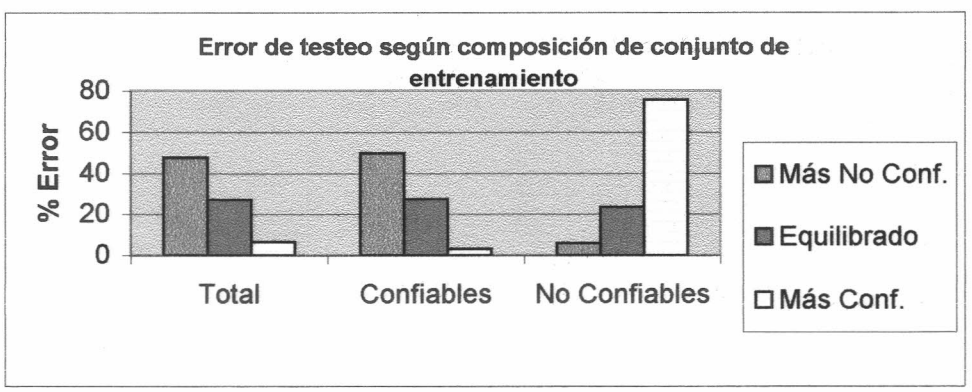


Figura 5.2: Comparación de errores de testeo según composición del conjunto de entrenamiento.

Prop. Confs.	Prop. No Confs.	Error Entren. Total	Error Testeo Total	Error Entren. Conf.	Error Testeo Conf.	Error Entren. No Conf.	Error Testeo No Conf.
I) 20	80	12.83	47.69	45.83	49.82	4.58	5.74
II) 50	50	21.50	27.19	26,33	27.39	16.67	23.37
III) 80	20	16.67	6.61	4.17	3.11	66.67	75.59

Tabla 5.1: Resultados obtenidos en Kohonen con conjuntos de entrenamiento con distintas proporciones de ejemplos de cada clase.

Observando la *Tabla 5.1* se puede notar que en la proporción I) donde existen 20% de confiables y 80% de no confiables, el error de testeo es el mayor obtenido. Esto ocurre porque esta proporción es inversamente proporcional a la proporción de confiables y no confiables que hay en el total de los datos usados para el testeo. Así, en el entrenamiento la red aprende muy poco de los confiables mientras que en el testeo, al existir tantos confiables y saber muy poco de ellos, se equivoca en un alto porcentaje. Lo contrario ocurre con la proporción III) donde la proporción es más similar a la proporción del total de los datos. La proporción II) logra un error intermedio.

Con respecto al error de testeo de los confiables y el error de testeo de los no confiables se ve que:

- La proporción I) se equivoca mucho más en los confiables.
- En la proporción II) ambos errores son similares.
- La proporción III) se equivoca mucho más en los no confiables.

En los tres casos la explicación radica justamente en la proporción de confiables y no confiables con que se entrenó la red.

También se puede observar que cuanto más lejos se está de la proporción que se presenta en el total de los datos (donde el porcentaje de no confiables no alcanza el 5%), mayor es la diferencia entre los errores totales de entrenamiento y testeo. Inclusive el error de testeo es superior al de entrenamiento en I) y II) donde la proporción de confiables y no confiables no se asemeja a la proporción total de los datos.

Por otro lado, cabe preguntarse cuál es la proporción adecuada en este problema, para entrenar una red con el algoritmo de Kohonen. Para ello hay que analizar el costo del error y lo que representa dicho error para el usuario de esta red. Como ejemplo, se puede estudiar cuál es la política que utiliza un Banco para aceptar o rechazar un crédito a un cliente.

Existen dos puntos de vista diferentes:

- una política clientelista: donde el objetivo es la captación y conservación de clientes. De este modo, al banco no le conviene rechazar un crédito a un 'supuesto no confiable por la red' ya que si resultase solvente significaría la pérdida de un cliente. En este caso, el costo del error del algoritmo al considerar a un real confiable como no confiable, sería equivalente a perder clientes o potenciales clientes.

- una política conservadora: apunta a que el banco esté absolutamente seguro que la persona podrá responder a la obligación crediticia. En este caso el error de considerar a un real confiable como no confiable no sería tan costoso ya que el banco se asegura no otorgar ese crédito en coherencia con su política conservadora.

En ambas políticas, el error de considerar a un real no confiable como confiable representa para el banco un gran riesgo de no recuperar el capital otorgado. Este error es inaceptable en una política conservadora ya que bajo este esquema se intenta asegurar al máximo el cobro del crédito.

Por lo tanto, seleccionar la proporción adecuada significa analizar los errores emergentes y relacionarlos con la política que defiende, en este caso, un banco.

El error de testeo en los confiables (observando la *Tabla 5.1*) equivale a considerarlos no confiables cuando realmente fueron confiables. Por lo tanto este error debe ser mínimo bajo una política clientelista.

El error de testeo en los no confiables representa el error de considerar a un cliente confiable cuando resulta ser no confiable. Este error debe ser mínimo en una política conservadora para no correr el riesgo de la incobrabilidad de un crédito.

Por lo anterior, la proporción III) se seleccionaría para una política clientelista donde el error en los confiables es el menor. Y la proporción I) para una política conservadora ya que el error en los no confiables es mínimo.

Como conclusión, la proporción de patrones de cada clase influye directamente en la habilidad de predicción, por lo cual es muy importante tener en claro cuál es la política elegida por el usuario de la red neuronal para seleccionar la proporción más conveniente de confiables y no confiables para entrenar la red.

5.3.2 Análisis de la influencia del tamaño del conjunto de entrenamiento

Se comparan las experiencias realizadas sobre idénticas arquitecturas (una matriz de salida de 10x10) pero con dos conjuntos de entrenamiento de distinto tamaño. La primer experiencia fue realizada sobre un conjunto de entrenamiento de 500 patrones (250 ejemplos de confiables y 250 de no confiables) mientras que la segunda experiencia fue realizada sobre un conjunto de entrenamiento de 974 patrones (494 ejemplos de confiables y 480 de no confiables).

La proporción de casos confiables y no confiables en ambos conjuntos es similar.

Cantidad Patrones	Error Entren. Total	Error Testeo Total	Error Entren. Conf.	Error Testeo Conf.	Error Entren. No Conf.	Error Testeo No Conf.
500	18.80	20.72	15.60	20.33	22.00	28.46
974	11.91	8.27	7.69	7.87	16.25	16.32

Tabla 5.2: Resultados obtenidos en Kohonen con conjuntos de entrenamiento de distinto tamaño.

En la *Tabla 5.2* se observa que todos los errores se redujeron notablemente al incrementar casi al doble la cantidad de patrones del conjunto de entrenamiento y que, sobre todo, la capacidad de generalización es muy superior al entrenar con el conjunto de mayor tamaño.

5.3.3 Análisis de la estructura de la red

A continuación se estudia el impacto de la arquitectura de red en cuanto a capacidad de aprendizaje y capacidad de generalización. Esto es, de qué manera influye la dimensión de la matriz de salida en ambos tipos de errores.

Las pruebas fueron realizadas siempre sobre el mismo conjunto de entrenamiento de 974 patrones de entrada con una proporción equilibrada entre clases (494 ejemplos de confiables y 480 de no confiables). El testeo se realizó sobre un conjunto de 14909 ejemplos.

El siguiente cuadro muestra los resultados de las experiencias realizadas:

DIMENSION MATRIZ	% ERROR ENTREN.	% ERROR TESTEO
5x5	31.11	28.20
7x7	24.54	14.76
8x8	16.63	16.00
9x9	15.09	12.95
10x10	11.91	8.27
11x11	13.35	5.92
12x12	14.17	8.42
13x13	12.42	12.83
15x15	10.16	10.00
16x16	9.55	10.97
18x18	9.75	9.63
20x20	9.45	8.46

Tabla 5.3: Resultados obtenidos en Kohonen con matrices de distinto tamaño.

La figura siguiente muestra estos resultados gráficamente:

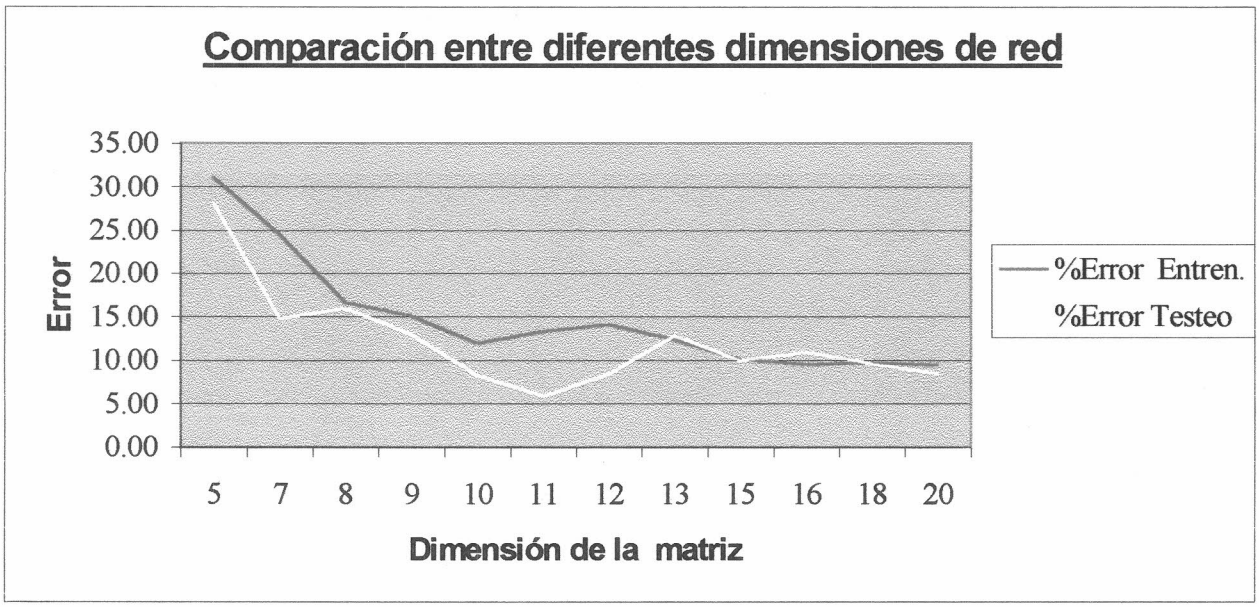


Figura 5.3: Gráfico de resultados de Kohonen con matrices de distinto tamaño.

Como se puede observar en la *Figura 5.3*, el error de entrenamiento baja a medida que aumenta la dimensión de la matriz. En cambio, el error de testeo alcanza su valor más bajo en la matriz de 11x11 y luego se mantiene más o menos constante.

Para entender los resultados siguientes se define como *cluster* para cada unidad de salida al conjunto de todos los patrones de entrada para los cuales esa unidad es ganadora y se activa. Se analiza en la *Tabla 5.4* la distribución y contenido de los clusters en cada entrenamiento y se obtiene:

DIMENSION	CLUSTERS USADOS	CLUSTERS CONF.	CLUSTERS NO CONF.	CLUSTERS CON CLASE UNICA	CLUSTERS CON EMPATE	CLUSTERS CON POCA DIFERENCIA DE CLASES	CLUSTERS DUDOSOS
5x5	24 (96%)	54.17 %	41.67 %	25.00 %	4.17 %	20.83%	25.00%
7x7	41 (84%)	39.02 %	56.10 %	34.15 %	4.88 %	12.19%	17.07%
8x8	61 (96%)	37.70 %	60.66 %	52.46 %	1.64 %	4.91%	6.55%
9x9	70 (87%)	41.43 %	58.57 %	57.14 %	0.00 %	4.28%	4.28%
10x10	89 (89%)	32.58 %	67.42 %	64.04 %	2.25 %	1.12%	3.37%
11x11	99 (82%)	34.12 %	60.14 %	66.32 %	2.32 %	0.93%	3.25%
12x12	122 (85%)	36.07 %	61.48 %	67.21 %	2.46 %	2.45%	4.91%
13x13	132 (79%)	34.09 %	65.15 %	70.45 %	0.76 %	4.55%	5.31%
15x15	173 (77%)	39.88 %	57.23 %	74.57 %	2.89 %	2.89%	5.78%
16x16	200 (79%)	37.50 %	60.00 %	79.00 %	2.50 %	1.50%	4.00%
18x18	228 (71%)	36.84 %	59.21 %	79.39 %	3.95 %	1.75%	5.70%
20x20	279 (70%)	34.77 %	61.65 %	83.87 %	3.58 %	1.79%	5.37%

Tabla 5.4: Comparación de resultados de Kohonen.

Clusters usados = clusters donde se clasificó algún patrón en el entrenamiento

Clusters Confiables = Cantidad de clusters con mayor proporción de confiables * 100 / Total de clusters usados

Clusters No Confiables = Cantidad de clusters con mayor proporción de no confiables * 100 / Total de clusters usados

Clusters con clase única = Cantidad de clusters con clase única * 100 / Total de clusters usados

Clusters con empate = Cantidad de clusters con igual proporción de clases * 100 / Total de clusters usados

Clusters con poca diferencia de clases = Cantidad de clusters cuya diferencia entre proporción de clases no supera el 20% * 100 / Total de clusters usados

Clusters dudosos = Clusters con empate + Clusters con poca diferencia de clases

Si se analiza el porcentaje total de clusters usados, es notable que a mayor dimensión de la matriz menor es dicho porcentaje. Es decir, en matrices grandes hay mayor porcentaje de clusters vacíos (clusters donde no se clasificó ningún patrón de entrenamiento). Este hecho puede deberse a que las redes de mayor dimensión necesitan pasar mayor cantidad de veces por los patrones de entrenamiento para llegar a cubrir todos los clusters y en las pruebas la cantidad de épocas se mantuvo constante para todas las dimensiones de red.

Para analizar si tiene sentido la clasificación manual de los clusters como "confiables" o "no confiables" de acuerdo con qué proporción de casos de cada clase se ubicó en ese cluster, se cuentan por un lado los clusters con clase única (albergan sólo confiables ó sólo no confiables), y por otro lado, los que tienen igual o casi igual proporción de casos de ambas clases (no aportarían información para la rotulación manual de los clusters, ya que no es posible determinar su clasificación).

Se nota que cuanto mayor es la dimensión de la matriz hay más cantidad de clusters con clase única (las clases quedan mejor separadas). Esto explica la mejora en el error de entrenamiento a mayores dimensiones de matriz.

Si se observa la cantidad de clusters con empate de clases, ésta es siempre inferior al 5%. Los clusters con diferencia entre ambas clases inferior al 20% son significativos en matrices pequeñas (de dimensión 5 y 7) y en el resto de las pruebas son menos del 5%. Sumando ambas situaciones los clusters dudosos representan menos del 6% en matrices con dimensiones mayores a 7x7 y, por lo tanto, la mayor parte de los clusters clasifican ampliamente una clase o la otra. De esto se deduce que el algoritmo separa correctamente los casos de entrada en las 2 clases de interés.

Otra característica notable es que la proporción de clusters rotulados como no confiables es mayor a la de confiables. Se considera que esto ocurre porque los ejemplos confiables son en su mayoría más parecidos entre sí (tiene en general todas las variables de problemas con valores bajos) y, por lo tanto se aglutinan en menos clusters.

Un hecho que llama la atención y que se mantiene constante en muchas de las diferentes dimensiones de matriz es que el error de testeo es menor al de entrenamiento. Analizando la composición de los clusters en el entrenamiento se puede descubrir que en los clusters que fueron clasificados como no confiables casi el 100% de los patrones es realmente no confiable, en cambio una parte de los clusters clasificados como confiables contienen alguna proporción de no confiables lo cual contribuye a subir el error de entrenamiento (la clasificación de esos patrones no confiables como confiables resulta incorrecta). En el testeo, dado que sólo el 5% de los patrones es de clase no confiable, en esos mismos clusters donde la clasificación de confiable era

dudosa en el entrenamiento caen muy pocos patrones de clase no confiable y, por lo tanto no es muy significativa su influencia en el error total.

En la comparación de los errores de testeo se observa que éste disminuye a partir de matrices con dimensión 10, y luego se mantiene más o menos constante. El mejor resultado se obtiene con una matriz de 11x11 donde el error es sólo del 5%. En este caso se puede notar que la proporción de clusters dudosos también alcanza su menor valor.

¿A qué se atribuye el hecho de que el error de testeo no baja considerablemente al aumentar el tamaño de la matriz, como lo hace el error de entrenamiento? Al igual que en el aprendizaje supervisado, este fenómeno se debe a la sobre-generalización. Cuando los clusters se hacen demasiado específicos la clasificación de nuevos patrones corre mayor riesgo de error, ya que se comienzan a separar patrones por características poco relevantes.

De los resultados se deduce que con una matriz de dimensión mayor o igual a 10x10 el error de testeo es aceptable. Para obtener un equilibrio entre calidad de los resultados y tiempo de entrenamiento se debería utilizar matrices de dimensión no muy superior a 10x10 (con matrices más grandes se incrementa notablemente el tiempo de entrenamiento).

Se pueden observar algunas de las particularidades analizadas anteriormente en los respectivos gráficos de la distribución por clase de las matrices resultantes, que se presentan en el Apéndice D.

5.4 Explicación de respuestas

Si se analizan los conjuntos de ejemplos que se asignaron a cada casilla de la matriz de salida, es posible descubrir ciertas características que los identifican. Como el algoritmo distribuye los clusters de una manera topográfica, los patrones en casillas vecinas también tienen similitudes. Analizando los valores promedio de los atributos de los patrones en cada cluster se puede ver, para una corrida ejemplo, que los patrones ubicados cerca del extremo inferior izquierdo de la matriz tienen volumen de consultas alto pero hace mucho tiempo, los ubicados en el inferior derecho tienen muchos antecedentes de problemas hace poco tiempo y muchas consultas, los del superior derecho tienen relacionados con problemas, los ubicados en el borde derecho y en el centro tienen también muchas consultas pero no tantos antecedentes de problemas, etc. Estas características comunes no son útiles para determinar si un nuevo ejemplo que pertenezca al cluster es confiable o no, pero sí pueden servir para encontrar justificaciones a las respuestas de la red. Por ejemplo: si los clusters del extremo inferior derecho resultan no confiables se puede decir que se debe a que tuvieron antecedentes de problemas financieros hace muy poco tiempo.

5.5 Conclusiones

La mejor experiencia lograda con el algoritmo de aprendizaje no supervisado de Kohonen fue realizada sobre un conjunto de entrenamiento de 974 ejemplos con un error de predicción del 5.92% sobre un conjunto de 15000 ejemplos. Es decir que la eficiencia en la predicción fue del 94.08%.

En el caso estudiado se puede apreciar cómo el algoritmo separa topológicamente los patrones de entrada en clusters representativos de las variables más importantes y sus integrantes tienen similitudes fáciles de detectar si se necesita dar una explicación. Esta capacidad para explicar las respuestas es un aspecto en el cual esta técnica aventaja considerablemente a otras.

Además en las experiencias se observa que los clusters resultantes del entrenamiento están en su mayoría compuestos por patrones de la misma clase, por lo tanto, la red clasifica los datos de manera apropiada para el problema en cuestión.

Otra conclusión que se puede extraer de las pruebas es que la proporción mayor de ejemplos de una clase en el conjunto de entrenamiento determina una mejor predicción para los casos de esa clase, por lo tanto, es conveniente entrenar con más patrones de la clase que más interese acertar.

Las mejores cifras de acierto en la predicción se obtuvieron con matrices de dimensión 11 x 11 y, el error no mejoró demasiado en matrices de mayor tamaño, por lo cual se deduce que, para el problema estudiado, la mejor dimensión de matriz es la de las cercanas a 100 celdas.

Capítulo 6

6. TECNICAS HIBRIDAS

En este capítulo se explica la forma de combinar un algoritmo de aprendizaje no supervisado, que reduce la dimensión de los datos de entrada, con un algoritmo supervisado que clasifica los patrones obtenidos en el proceso no supervisado. Se describe el aprendizaje no supervisado hebbiano con sus variantes en cuanto a cantidad de unidades de salida y se presentan las reglas de aprendizaje de Oja y Sanger. Luego se incluyen los resultados obtenidos al aplicar este tipo de aprendizaje junto con un algoritmo supervisado de Backpropagation.

Utilizando un método de aprendizaje no supervisado hebbiano es posible reducir la dimensionalidad de los datos de entrada. En el problema planteado los patrones de entrada están constituidos por 10 atributos. Si se entrena una red no supervisada hebbiana se puede transformar estos patrones en vectores de menor cantidad de componentes. En el caso implementado, se realizaron diversas pruebas sobre una red con 4 unidades de salida, por lo cual, los patrones de entrada quedaron reducidos a 4 componentes, tratando de mantener la mayor cantidad de la información implícita en los datos.

Luego de reducir la dimensión de los datos, se ingresaron los resultados de esa red no supervisada a un perceptrón multicapa con aprendizaje supervisado Backpropagation para lograr una clasificación de los patrones en las dos clases de interés (confiables y no confiables).

Las ventajas que se esperan obtener con esta técnica híbrida son:

- Las unidades de entrada a la red son menos de la mitad, por lo cual baja la complejidad del aprendizaje y se ahorra tiempo de entrenamiento.

- Los patrones de entrada tienen sólo información esencial, ya que se obtuvieron sus componentes principales, y, por lo tanto, la red debería aprender a clasificar sin inconvenientes.

6.1 Descripción del algoritmo de aprendizaje hebbiano

El aprendizaje hebbiano es una técnica de aprendizaje no supervisado que se basa en conexiones para aprender usando la "Regla de Hebb modificada". Las unidades de salida no compiten entre sí (no hay una ganadora).

El objetivo de este aprendizaje es obtener medidas de familiaridad entre los patrones de entrada. Si la red tiene más de 1 unidad de salida el entrenamiento logra reducir la dimensionalidad de los datos de entrada, proyectándolos en sus "componentes principales".

Este método de aprendizaje no tiene como finalidad producir un clustering o clasificación de los datos sino establecer la familiaridad de un nuevo caso con una muestra analizada.

6.1.1 Aprendizaje hebbiano con una unidad lineal

Para una red con una sola unidad de salida, la respuesta, luego de aplicar un patrón de entrada, es un valor continuo.

Se supone que se tienen vectores de entrada ε_i para i=1,2,...,n con una distribución P(ε) donde los componentes de los patrones pueden tener valores binarios o continuos. En cada paso se aplica a la red una instancia ε. Luego de recibir suficientes ejemplos la red puede indicar si un determinado patrón de entrada (que no participó en el entrenamiento) se adapta o no a la distribución, es decir cuán parecido es el patrón al conjunto de entrenamiento.

Si se expresa como fórmula:

$$V = \sum_j w_j \varepsilon_j$$

donde V es la unidad de salida y w_j es el peso que conecta la entrada j a la salida.

V debe convertirse en una medida escalar de familiaridad, es decir, cuanto más probable sea una entrada en particular ε, más grande será |V|.

Gráficamente:

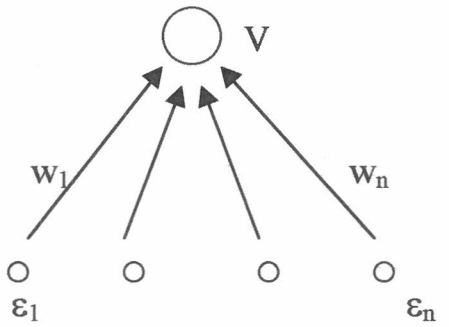


Figura 6.1: Aprendizaje hebbiano

Para lograr que la red aprenda, lo más natural es probar con aprendizaje hebbiano simple, en el cual los pesos se modifican a cada paso de la siguiente manera:

$$\Delta w_i = \eta V \varepsilon_i$$

donde η es el parámetro de velocidad de aprendizaje.

Los patrones más frecuentes tienen mayor influencia en esta fórmula y producirán una salida más grande.

El problema de esta regla es que los pesos crecen indefinidamente y el aprendizaje no termina nunca.

Para prevenir la divergencia del aprendizaje es necesario restringir el crecimiento del vector de pesos w. Hay varias formas de lograr esto, por ejemplo, realizar una simple renormalización $w_j' = \alpha \cdot w_j$ para todos los pesos luego de cada actualización, eligiendo α tal que $|w'| = 1$.

Una solución más sencilla es modificar la Regla de Hebb para que el vector de pesos mantenga un valor constante ($|w'| = I$) sin hacer normalizaciones a mano [Oja, 1982]. A esta modificación se la denomina Regla de Oja y consiste en hacer decaer el valor del vector de pesos en forma proporcional a V^2.

$$\Delta w_i = \eta V (\varepsilon_i - V w_i) \qquad \text{(Regla de Oja)}$$

Esta fórmula es similar al aprendizaje con regla delta reversa (Δw depende de la diferencia entre la entrada actual y la salida propagada hacia atrás).

6.1.2 Aprendizaje hebbiano con varias unidades lineales

Un método estadístico muy común para analizar datos es el análisis de componentes principales. Se trata de encontrar M vectores ortogonales en el espacio de los datos que representen lo mejor posible la varianza de los datos. Al proyectar los datos de su espacio original de N dimensiones en otro de M dimensiones donde M<N se obtiene una reducción de dimensionalidad que generalmente retiene la mayor parte de la información intrínseca de los datos.

La primera componente principal está siempre en la dirección con máxima varianza.

La regla de Oja encuentra un vector de pesos que es la primer componente principal de los datos. Sería deseable tener una red con M salidas que encuentre las M primeras componentes principales.

Las redes son lineales y cada salida V_i está dada por:

$$V_i = \sum_j w_{ij} \varepsilon_j$$

La siguiente figura muestra una red para i = 3:

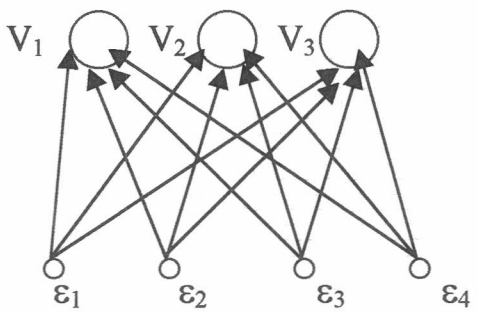


Figura 6.2: Aprendizaje hebbiano con varias unidades lineales

Para un espacio de dimensión N las reglas de aprendizaje pueden ser:

Regla de Sanger [Sanger, 1989a]:

$$\Delta w_{ij} = \eta V_i (\varepsilon_j - \sum_{k=1}^{i} V_k w_{kj})$$

Regla de Oja [Oja, 1989]:

$$\Delta w_{ij} = \eta V_i (\varepsilon_j - \sum_{k=1}^{N} V_k w_{kj})$$

Como se puede observar, la única diferencia entre las fórmulas es el límite superior de la sumatoria. Para M=1 las reglas son idénticas a la regla de Oja para una unidad lineal.

En la Regla de Sanger los vectores de pesos resultan exactamente y en orden las direcciones de las M componentes principales, mientras que, en la de Oja los M vectores forman un subespacio idéntico al de las M componentes pero no lo son.

6.2 Experiencias y resultados

6.2.1 Análisis de resultados del algoritmo no supervisado

Dado que el entrenamiento hebbiano no supervisado encuentra una función de distribución de los datos (una medida de similitud), se entrenó la red sólo con ejemplos de una clase (sólo confiables o sólo no confiables).

El conjunto de entrenamiento utilizado para Backpropagation se dividió en dos subconjuntos según su clase (confiables y no confiables). Cada uno de los conjuntos tiene alrededor de 500 ejemplos.

Se realizaron entrenamientos de la red con ambos conjuntos de datos, utilizando alternadamente las reglas de Sanger y Oja.

Las pruebas realizadas y sus correspondientes identificadores son los siguientes:

Prueba 1: Se utilizó el conjunto confiable y la regla de Sanger (1-C-S).
Prueba 2: Se utilizó el conjunto confiable y la regla de Oja (1-C-O).
Prueba 3: Se utilizó el conjunto no confiable y la regla de Sanger (1-N-S).
Prueba 4: Se utilizó el conjunto no confiable y la regla de Oja (1-N-O).

El resultado del aprendizaje es un conjunto de 4 valores continuos para cada patrón de entrada, que resume la información de sus 10 atributos.

A continuación se recuerda el contenido de los patrones de entrada y se indica entre paréntesis cómo se los va a identificar:

1. Sexo (Sexo).
2. Edad (Edad).
3. Volumen de consultas (Vol.).
4. Antigüedad de la última consulta (Ant.).
5. Problemas Tiempo 1: Menos de 1 año (P1).
6. Problemas Tiempo 2: Entre 1 y 3 años (P2).
7. Problemas Tiempo 3: Entre 3 y 6 años (P3).
8. Problemas Tiempo 4: Entre 6 y 10 años (P4).
9. Problemas Tiempo 5: Más de 10 años (P5).
10. Relacionados con problemas (Rel.).

En todas las pruebas, analizando los resultados a simple vista se puede ver que cada uno de los cuatro valores resultantes representa un atributo o alguna combinación de atributos de entrada, ya que el valor de cada uno de los nuevos atributos aumenta o disminuye en forma directa o inversamente proporcional a algún atributo o conjunto de atributos de entrada.

Si se grafica, aproximadamente, los resultados de las pruebas realizadas por número de prueba y variable se obtiene la siguiente tabla:

	Variables									
Prueba	Sexo	Edad	Vol.	Ant.	P1	P2	P3	P4	P5	Rel.
1-C-S		1°		4°	3°				2°	
2-C-O	4°	1°		3°				2°		
3-N-S		1°	1°	4°	2°,3°	3°			2°	
4-N-O		1°			1°,2°		3°		2°	

Tabla 6.1: Intervención de las variables de entrada en las componentes principales.

Los números en la *Tabla 6.1* indican qué componente de salida representa a esa variable. Las casillas que tienen dos números indican que esa variable influencia a 2 atributos de salida.

Se puede deducir de estos resultados que al entrenar con un conjunto de ejemplos de clase no confiable, la red descubre la importancia de las variables de problemas, y en menor grado, de la edad y la antigüedad y volumen de las consultas. Cuando se entrena con ejemplos de clase confiable las variables de problemas no tienen mucha influencia (porque en la mayoría de los casos tienen valores cercanos a cero) y cobran importancia la edad, el sexo y la antigüedad de las consultas.

6.2.2 Análisis de resultados del algoritmo supervisado

Para aprovechar esta reducción de dimensionalidad lograda con el algoritmo no supervisado se entrenó con el resultado del entrenamiento anterior una red con aprendizaje supervisado backpropagation. Los patrones de entrada se conforman por los valores de las cuatro componentes principales encontradas y se incorpora la variable respuesta para que sea utilizada como respuesta esperada en el aprendizaje supervisado.

Se realizaron pruebas con los conjuntos obtenidos de los cuatro entrenamientos no supervisados. Para el entrenamiento supervisado se agregó a cada conjunto los ejemplos correspondientes a la clase contraria, calculando los cuatro nuevos atributos con los vectores de pesos obtenidos por la red no supervisada correspondiente. De esta forma todos los conjuntos tienen alrededor de 1000 ejemplos (con igual proporción de cada clase).

Para cada conjunto se entrenó una red backpropagation con un nivel oculto de 11 unidades, $\beta = 0,9$, $\eta = 0,2$ inicialmente con parámetros adaptivos y momentos con $\alpha = 0,9$. El tiempo de entrenamiento fue obviamente mucho menor que el necesario para entrenar con entradas de 10 variables.

El conjunto de entrenamiento tiene 974 ejemplos y el de testeo 14909.

La *Figura 6.3* muestra el comportamiento de los 4 conjuntos en el entrenamiento (qué porcentaje de los datos de entrada no logran aprenderse).

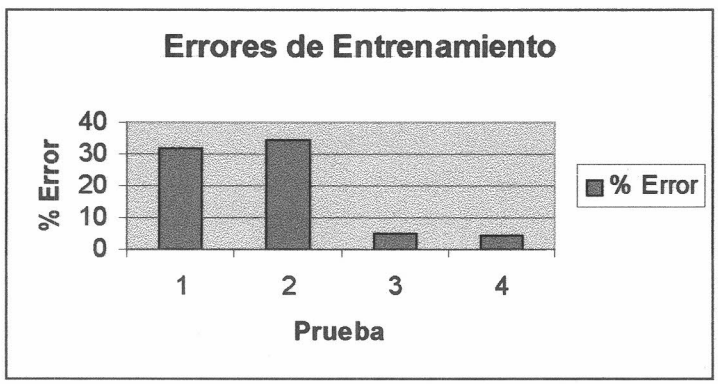


Figura 6.3: Errores de entrenamiento de una red supervisada sobre conjuntos de datos resultantes de entrenamientos no supervisados.

Observando este gráfico resulta evidente que en el entrenamiento se comportan mucho mejor las pruebas realizadas con los datos de entrada resultantes del aprendizaje hebbiano de los conjuntos de no confiables (Pruebas 3 y 4). Como se había notado en el análisis a simple vista, en los aprendizajes con no confiables las componentes principales estaban más influenciadas por las variables de problemas. Esto ocurre porque los atributos de problemas tienen una varianza más alta en los patrones no confiables y en los confiables tienen en su mayor parte valor 0. Los casos de clase no confiable tienen características más distintivas que los casos de clase confiable y, el aprendizaje no supervisado pudo resumir mejor los 10 atributos de entrada en sólo 4 variables y luego, asignarles valores de salida diferentes a los ejemplos confiables.

Luego de entrenar las 4 redes se realizó el testeo de las mismas con los patrones de testeo. Como ya se analizó en el capítulo de aprendizaje

supervisado, el mejor conjunto de pesos para testear depende de la precisión con que se quiera acertar en cada una de las clases o en el total. Para diferenciar las diversas situaciones se calculó el menor número de predicciones inválidas en el total, en confiables y en no confiables asignando distintos pesos de importancia a las dos clases.

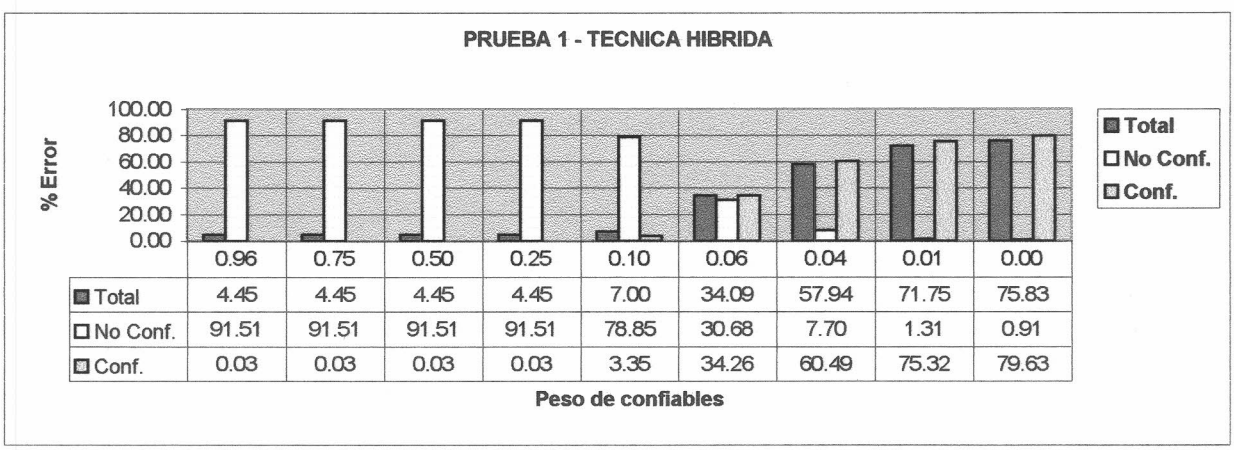
Para tener la posibilidad de guardar distintas configuraciones durante el entrenamiento se definió una función de selección de la siguiente manera:

$$Fs = (p * mc) + ((1 - p) * mn)$$

donde p es el peso que se le otorga a los casos confiables, mc es la cantidad de casos confiables no acertados, mn es la cantidad de casos no confiables no acertados.

Se seleccionaron 7 valores posibles de p.

Para la prueba 1 (Confiables con regla de Sanger) los resultados se observan en el siguiente gráfico:



	0.96	0.75	0.50	0.25	0.10	0.06	0.04	0.01	0.00
Total	4.45	4.45	4.45	4.45	7.00	34.09	57.94	71.75	75.83
No Conf.	91.51	91.51	91.51	91.51	78.85	30.68	7.70	1.31	0.91
Conf.	0.03	0.03	0.03	0.03	3.35	34.26	60.49	75.32	79.63

Figura 6.4: Error de generalización en prueba 1.

Se puede observar que los errores son constantes para p >= 0,25. Esto ocurre porque la proporción de casos no confiables es de 5% del total de patrones, por lo tanto, hay que darle mucho peso a los no confiables para lograr que sean significativos. Por la misma razón de composición del conjunto total, el error total es bajo cuando es bajo el error de los confiables.

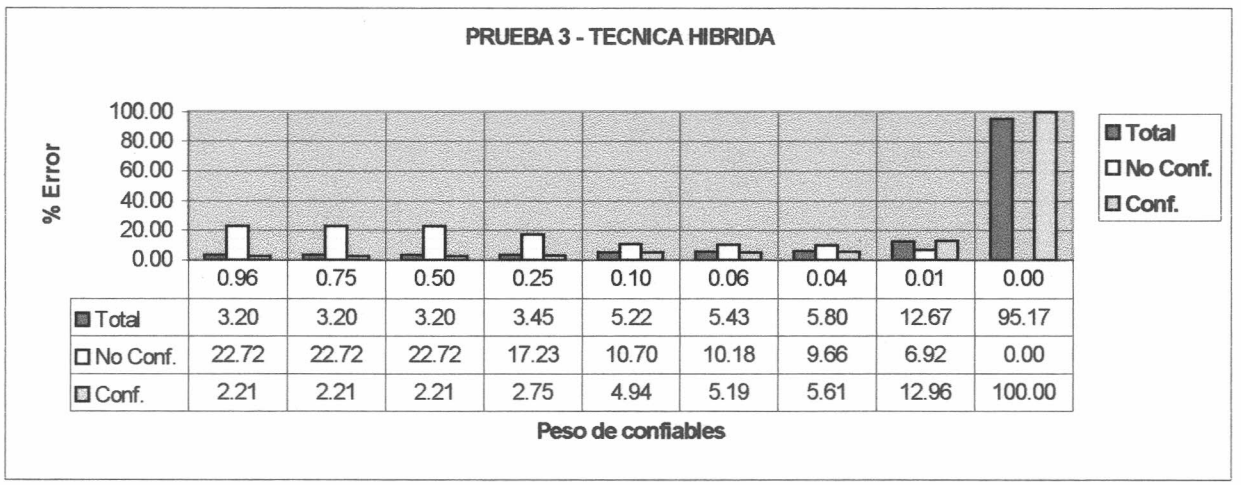
El error en la predicción de confiables es casi 0 en los p entre 0,96 y 0,25. Es decir la red responde correctamente en todos los casos de confiables. Para estas mismas configuraciones el error en los no confiables es muy alto (91,51 %), por lo cual no sería razonable seleccionar esta configuración de pesos.

Para p=0,06 el error es de aproximadamente 30% en ambas clases, resultando la opción más equilibrada de esta experiencia.

Para p = 0, el error en los no confiables es menor a 1, pero es demasiado alto en los confiables (75,83%). En este caso la configuración es apropiada sólo si se quiere ser muy preciso con los no confiables y no interesa equivocarse en el 75% de los confiables.

La prueba 2 (Confiables con Regla de Oja) dio resultados mucho más pobres que la prueba 1, y, por lo tanto no será analizada.

La prueba 3 (No confiables con Regla de Sanger) arrojó los siguientes resultados:



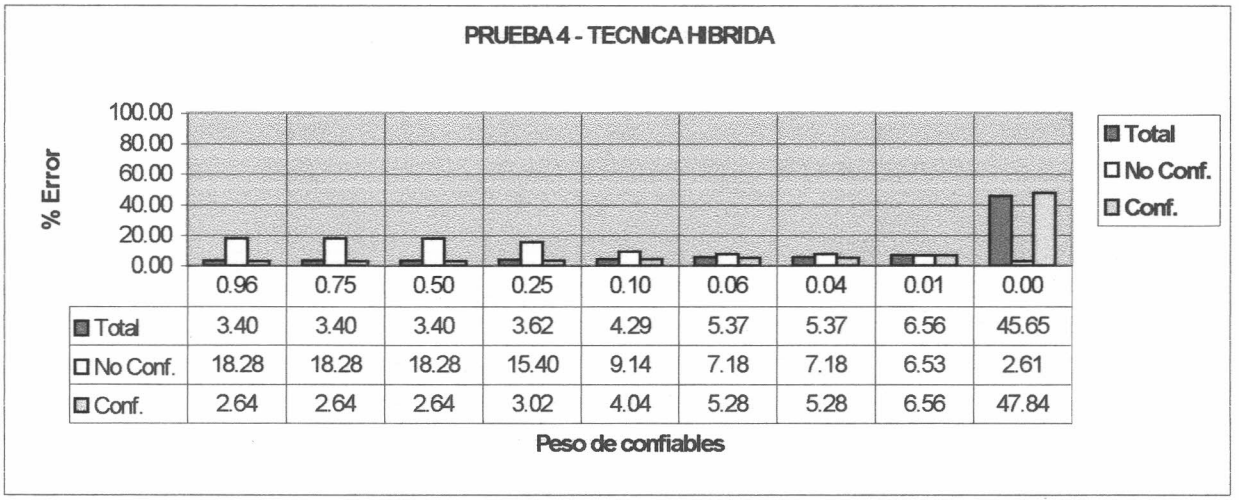
	0.96	0.75	0.50	0.25	0.10	0.06	0.04	0.01	0.00
Total	3.20	3.20	3.20	3.45	5.22	5.43	5.80	12.67	95.17
No Conf.	22.72	22.72	22.72	17.23	10.70	10.18	9.66	6.92	0.00
Conf.	2.21	2.21	2.21	2.75	4.94	5.19	5.61	12.96	100.00

Figura 6.5: Error de generalización en prueba 2.

Los resultados en esta experiencia, resultante de entrenar la red no supervisada con ejemplos no confiables, son muy distintos a los de la experiencia anterior. Los porcentajes de confiables y no confiables no acertados son más equilibrados para todos los p (excepto para el caso extremo de p=0). En las puntas mejora el error de predicción de una u otra clase y en el centro se ven proporciones más parejas. Aquí se hace notorio que el entrenamiento de la red

no supervisada es más efectivo con ejemplos no confiables, ya que aprende mejor ambas clases de patrones. En esta prueba se logra un error 0 en la predicción de no confiables (para p=0) pero la configuración resulta inútil, ya que se equivoca en todos los confiables.

La prueba 4 (No confiables con Regla de Oja) arrojó los siguientes resultados:



	0.96	0.75	0.50	0.25	0.10	0.06	0.04	0.01	0.00
Total	3.40	3.40	3.40	3.62	4.29	5.37	5.37	6.56	45.65
No Conf.	18.28	18.28	18.28	15.40	9.14	7.18	7.18	6.53	2.61
Conf.	2.64	2.64	2.64	3.02	4.04	5.28	5.28	6.56	47.84

Figura 6.6: Error de generalización en prueba 4.

En esta prueba los resultados que se obtienen son similares a la prueba anterior. Los errores mínimos son más altos pero se puede apreciar que en ningún caso se dispara tanto el error de la clase contraria y que hay un valor de p (0,01) donde todos los errores se mantienen en 6,5. De la comparación de las pruebas 3 y 4, ambas entrenando la red no supervisada con no confiables, es posible deducir que la Regla de Sanger da mejores valores de error mínimo pero, la Regla de Oja produce proporciones de errores más equilibrados.

Para hacer una comparación entre las 4 pruebas, los porcentajes de error que se muestran a continuación indican para qué proporción de ejemplos la respuesta de la red fue incorrecta. Las cuatro columnas de errores corresponden al mejor valor obtenido a lo largo de un entrenamiento, no necesariamente en la misma época.

Conjunto Prueba	% Error Entrenam.	% Error Testeo	% Error Confiab.	% Error No Confiab.
1	31,83	4,45	0,03	0,91
2	34,68	17,38	13,24	0,00
3	5,24	3,20	2,21	0,00
4	4,52	3,40	2,64	2,61

Tabla 6.2: Comparación de resultados de técnica híbrida.

En general, se puede notar, que los errores de confiables y no confiables son mucho menores que el error de testeo total, esto ocurre porque al bajar mucho el error de una clase sube considerablemente el otro y, si se desea una predicción equilibrada es necesario resignar la perfección de alguna de las clases. Este tema ya se estudió en el capítulo de Backpropagation.

En las pruebas donde el entrenamiento no supervisado se realizó con confiables resulta excelente la predicción de los confiables y lo mismo ocurre cuando se trata de los no confiables en la prueba inversa.

Los resultados obtenidos al aplicar la Regla de Sanger son mejores en cuanto a valores de error mínimo que los correspondientes a la Regla de Oja, pero en el análisis detallado se comprobó que los valores de los errores por clase resultan más equilibrados con la Regla de Oja (predice mejor ambos tipos de clases en vez de alguna en particular).

6.3 Explicación de respuestas

En el caso del aprendizaje hebbiano la explicación de respuestas estaría relacionada con el análisis de componentes principales (confrontar con *Tabla 6.1*).

6.4 Conclusiones

Reduciendo la dimensionalidad de los datos de 10 a 4 atributos por medio de aprendizaje no supervisado hebbiano y aplicando sobre el resultado un algoritmo de aprendizaje supervisado Backpropagation se obtuvo una

predicción acertada de 96,80% sobre alrededor de 15000 casos. En principio, estos resultados parecen ser menos precisos que los obtenidos utilizando el aprendizaje supervisado únicamente, cabe preguntarse entonces, desde el punto de vista de la relación entre complejidad y eficiencia, si esta técnica es conveniente. Al igual que en las experiencias descriptas en capítulos anteriores, es necesario tener en cuenta una función de selección que indique los costos de equivocarse en la predicción de cada una de las clases, y, en base a la misma, seleccionar la configuración de pesos que brinde el equilibrio deseado.

Los resultados más equitativos en predicción de ambas clases se obtuvieron entrenando la red no supervisada tomando ejemplos no confiables y aplicando la Regla de Oja.

Capítulo 7

7. EVALUACION DE RESULTADOS, DISCUSION Y CONCLUSIONES GENERALES

Finalmente, se comparan los resultados obtenidos con las experiencias realizadas: aprendizaje supervisado, aprendizaje no supervisado y una combinación de ambos. Se analizan las ventajas y desventajas de cada método y de las redes neuronales en general para el problema en estudio.

Para comparar los resultados de las tres técnicas estudiadas se muestran una tabla y un gráfico con los mejores resultados de cada una, entrenando sobre un conjunto de alrededor de 1000 patrones y testeando sobre el total de los casos (15000).

El indicador que se compara es:

% Error de predicción: indica qué porcentaje de los ejemplos de la muestra de testeo dio el resultado contrario al correcto (casos no acertados).

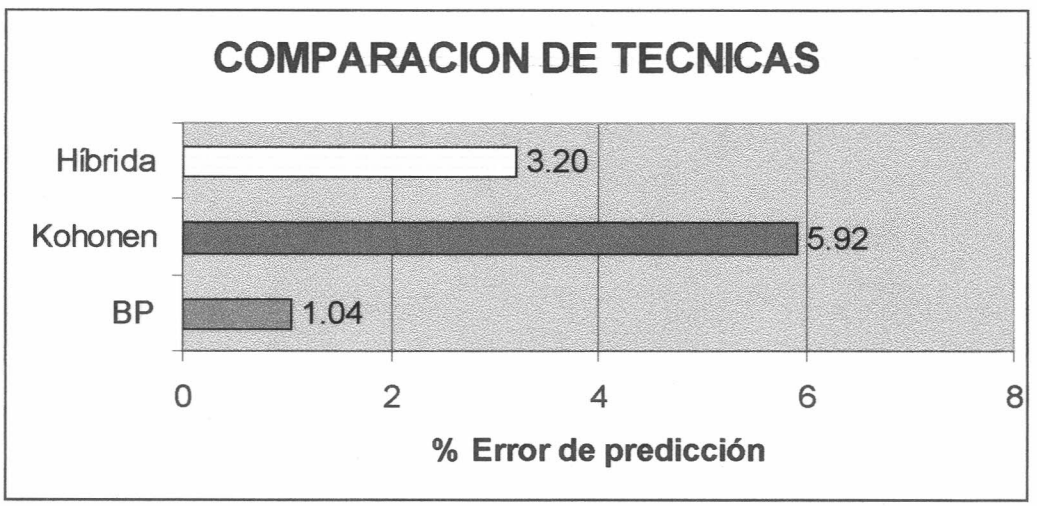


Figura 7.1: Comparación de errores de predicción de las distintas técnicas.

En primer lugar se comparan los resultados del aprendizaje supervisado Backpropagation sobre el conjunto de patrones de 10 atributos con la reducción de dimensionalidad de la técnica híbrida donde se utilizó también Backpropagation pero, esta vez, sobre entradas de 4 atributos. Se puede notar

que los resultados sobre el total de los datos son mejores en el caso inicial (el error de testeo es de 1,04% contra 3,20% de la técnica híbrida). Se puede afirmar así que la técnica de Backpropagation posee mayor capacidad de generalización que las otras técnicas estudiadas aquí. Pero si no se toma en cuenta el tiempo de entrenamiento, se deduce de los resultados obtenidos que no es conveniente aplicar la técnica híbrida. Dado que los datos de entrada a la red son temporales (dependen del momento en que se tomen) puede ocurrir que la red deba ser re-entrenada con bastante frecuencia. En este caso se haría necesario analizar la relación entre complejidad y eficiencia de ambas técnicas y considerar si el detrimento de eficiencia de la predicción puede ser justificado por la mejora en el tiempo de entrenamiento. En este trabajo no se estudian los tiempos de entrenamiento y predicción de cada una de las redes ya que dependen fuertemente del hardware utilizado y del lenguaje de programación de los algoritmos.

Utilizando Kohonen los valores de error son bastante más altos. La ventaja de este método es que los datos resultan distribuidos en clusters y se puede explicar los resultados analizando las características comunes de los patrones que componen un cluster. Es decir, el aprendizaje por mapeo de características permite explicar las respuestas más facilmente que las otras técnicas comparadas.

El objetivo que se persigue al aplicar redes neuronales en el problema estudiado es decidir de una manera lo más objetiva posible si se puede confiar financieramente en un individuo o no. Por más precisas que sean las respuestas de la red, ésta se volverá difícil de utilizar si no se puede explicar el por qué de un rechazo, por esta razón se torna importante la explicación de las respuestas de la red.

Es inútil insistir en que el diseño e implementación de técnicas de predicción para los fines presentados aquí, sólo tiene sentido si se coloca en un contexto preciso donde los costos de los distintos tipos de errores de decisión (dos, en este caso) pueden ser conocidos en su importancia relativa (confrontar con función de selección en la sección 4.3.4). Sólo de esta manera resulta posible evaluar la eficacia de una determinada técnica, ya que si se considera la reducción de la probabilidad de cada tipo de error por separado, el problema resulta trivial.

De los antecedentes registrados en relación con el estudio comparativo de las RNA y otras técnicas aplicadas a *credit scoring* concluímos que las primeras constituyen una herramienta justificable para este fin. Por ejemplo, el trabajo realizado en la Universidad de Harvard [Galindo, 1998] refleja que los resultados que se obtienen con RNA son competitivos y susceptibles de mejoras substanciales mediante el incremento del tamaño del conjunto de entrenamiento y del tiempo de entrenamiento.

Finalmente, vale la pena reiterar que ningún método garantiza por sí solo un resultado óptimo desde todo punto de vista. Los mejores resultados se basan en la combinación de modelos compensando las desventajas de una técnica con las ventajas de otra.

Capítulo 8

8. FUTUROS TRABAJOS

El alcance del presente trabajo ha comprendido implementar, analizar y comparar distintos algoritmos de redes neuronales para el problema de predecir si una persona puede resultar confiable para el otorgamiento de un crédito bancario teniendo como antecedente su pasado comercial y financiero.

Resultaría bastante útil también poder predecir una cierta *medida de confiabilidad*, es decir, un número que exprese el grado de confiabilidad comercial de la persona para futuros créditos y no sólo si es confiable o no confiable.

También podría medirse la confiabilidad para distintas etapas en el futuro, es decir, poder predecir el grado de confiabilidad o no confiabilidad de una persona para el próximo semestre, para el próximo año, para los próximos 5 años, para los próximos 10 años, etc.

Otro posible agregado que se podría plantear es, conociendo ciertas variables de situación del país en el período de tiempo que se analiza, otorgarle distinto peso a los atributos de antecedentes históricos según el momento en que ocurrieron (por ejemplo: en un período de hiperinflación o de recesión es más común que ocurran problemas financieros y deberían tener menos peso que en una época normal).

Cabe agregar también una idea aplicable al algoritmo supervisado de Backpropagation en el que podría introducirse un sesgo respecto del entrenamiento con probabilidad p al tomar ejemplos de una clase. Por ejemplo, teniendo en cada época una probabilidad p de entrenar con confiables y una probabilidad 1 – p de entrenar con no confiables.

Estas ampliaciones al trabajo aquí expuesto representan algunos lineamientos para futuras investigaciones en esta temática.

REFERENCIAS

Capítulo 1

Carbonell, J. G. y Langley, P. (1990). *Machine Learning*. In Shapiro, S.C. (ed), Encyclopedia of Artificial Intelligence, Vol. I, John Wiley, New York.

Cox, E. (1994). *The Handbook of Fuzzy Systems.* Academic Press. New York.

Davis L., (1991). *Handbook of Genetic Algorithms.* Van Nostrand Reinhold. New York.

Didner R. (1995). *Intelligent Systems at American Express*. Intelligent Systems for Finance and Business (31-37). Editado por Goonatilake y Treleaven. John Wiley & Sons.

Dutta, S. y Shekhar, S. (1988). *Bond Rating: A Non-conservative Application of Neural Networks*. Proceedings of IEEE International Conference on Neural Networks, San Diego.

Evans, J. (1988). *The Knowledge Elicitation Problem: A Psychological Perpective*. Behaviour and Information Technology, 7, 2, 111-130.

Galindo, J. y Tamayo P. (1998). *Credit Risk Assessment Using Statistical and Machine Learning: Basic Methodology and Risk Modeling Applications.* Publicado en Proceedings of Computation Economics '97 conference. Department of Economy, Harvard University.

Goldberg, D. E. (1989) *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison Wesley, Reading, MA.

Goonatilake S. (1995). *Intelligent Systems for Finance and Business: An Overview*. Intelligent Systems for Finance and Business (1-28). Editado por Goonatilake y Treleaven. John Wiley & Sons.

Graf, J. y Nakhaeisadeh, G. (1993). *Recent Development in Solving the Credit-scoring Problem*. Plantamura V. L., Soucek B. And Visagio, G. (eds.),

Logistic and Learning for Quality Software, Management and Manufacturing, John Wiley, New York.

Hetch-Nielsen, R. (1990). *Neurocomputing*, Addison-Weslwy, Reading MA.

Kosko, B. (1992). *Neural Networks and Fuzzy Systems*. Prentice Hall, Englewood Cliffs, NJ.

Leigh D. (1995). *Neural Networks for Credit Scoring*. Intelligent Systems for Finance and Business (61-69). Editado por Goonatilake y Treleaven. John Wiley & Sons.

Quinlan, J. R. (1986). *Induction of Decision Trees*. Machine Learning, 1, 81-106.

Rumelhart, D., McClelland, J. y PDP Research Group (1986*). Parallel Distributed Processing*. MIT Press, Cambridge, MA.

Schumann M. (1992). *Comparing Artificial Neural Networks with Others Methods within the field of Credit Scoring;* Proceedings of the 2nd. Pacific Rim International Conference on Artificial Intelligence, Korea.

Statlog (1993). *Machine Learning, Neural and Statistical Classification.* Deliverable 4.1 ESPRIT PROJECT 5170.

Wasserman, P. (1989). *Neural Computing*. Van Nostrand Rheinhold, New York.

Capítulo 4

Bryson, A.E y Y.C. Ho (1969) *Applied Optimal Control*. New York: Blaisdell.

Cater J. P. (1987) *Succesfully Using Peak Learning Rates of 10 (and Greater) in Back-Propagation Networks with the Heuristic Learning Algorithm*. In IEEE First International Conference on Neural Networks (San Diego 1987), eds. M. Caudill and C. Butler, vol. II, 645-651. New York: IEEE.

Franzini, M. A. (1987). *Speech Recognition with Backpropagation*. In Proceedings of the Ninth Annual Conference of the IEEE Engineering in Medicine and Biology Society (Boston 1987), 1702-1703. New York: IEEE.

Freeman J. A. y D. M. Skapura (1992). *Neural Networks: Algorithms, Applications and Programming Techniques*. Addison-WesleyPublishing Company.

Hertz J., Krogh A., Palmer R. (1990). *Introduction to the Theory of Neural Computation;* Santa Fé Institute Editorial Board.

Jacobs R. A. (1988). *Increased Rates of Convergence Through Learning Rate Adaptation*. Neural Networks 1,295-307.

Le Cun, Y. (1985). *Une Procédure d'Apprentissage pour Réseau à Seuil Assymétrique*. In Cognitiva 85: A la Frontière de l'Intelligence Artificielle des Sciences de la Connaissance des Neurosciences (Paris 1985), 599-604. Paris: CESTA.

Parker D.B. (1985). *Learning Logic*. Technical Report TR-47, Center for Computational Research in Economics and Manageament Science, Massachusetts Institute of Technology, Cambridge, MA.

Plaut, D., S. Nowlan, y G. Hinton (1986). *Experiment on Learning by Back Propagation*. Technical Report CMU-CS-86-126. Department of Computer Science, Carnegie Mellon University, Pittsburgh, PA.

Rumelhart D. E., Hinton G. E. y Williams R. J. (1986a). *Learning Representations by Back-Propagating Errors*. Nature 323, 533-536. Reimpreso en Anderson y Rosenfeld (1988).

Rumelhart D. E., Hinton G. E. y Williams R. J. (1986b). *Learning Internal Representations by Error Propagation*. In Parallel Distributed Processing, vol. 1, chap. 8. Reimpreso en Anderson y Rosenfeld (1988).

Vogl, T. P., Magis J. K., Rigler A. K., Zinc W. T. Y Alkon D. L. (1988) *Accelerating the Convergence of the Back-Propagation Method*. Biological Cybernetics 59, 257-263.

Werbos, P. (1974). *Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences*. Ph.D. Thesis, Harvard University.

Capítulo 5

Feldman, J. A. y Ballard D. H. (1982). *Connectionist Models and Their Properties*. Cognitive Science 6, 205-254. Reimpreso en Anderson y Rosenfeld [1988].

Grossberg, S. (1976a) *Adaptive Pattern Classification and Universal Recoding: I. Parallel Development and Coding of Neaural Features Detectors*. Biological Cybernetics 23, 121-134. Reimpreso en Anderson y Rosenfeld [1988].

Grossberg, S. (1980) *How Does the Brain Build a Cognitive Code?* Psychological Review 87, 1-51. Reimpreso en Anderson y Rosenfeld [1988].

Grossberg S. (1987b). *Competitive Learning: From Interactive Activation to Adaptive Resonance*. Cognitive Science 11, 23-63.

Kohonen, T. (1982). *Self-Organized Formation of Topologically Correct Feature Maps*. Biological Cybernetics 43, 59-69. Reimpreso en Anderson y Rosenfeld [1988].

Kohonen, T. (1989). *Self-Organization and Associative Memory* (3rd ed.). Berlin: Springer-Verlag.

Lippman, R. P. (1987). *An Introduction to Computing with Neural Nets*. IEEE ASSP Magazine, Abril 1987, 4-22.

Rumelhart D. E. y Zipser D. (1985). *Feature Discovery by Competitive Learning*. Cognitive Science 9, 75-112. Reimpreso en Parallel Distributed Processing, vol. 1, chap. 5.

Winters J. H. y C. Rose (1989). *Minimun Distance Automata in Parallel Networks for Optimum Classification*. Neural Networks 2, 127-132.

Capítulo 6

Oja E. (1982). *A Simplified Neuron Model As a Principal Component Analyzer*. Journal of Mathematical Biology 15, 267-273.

Oja E. (1989). *Neural Networks, Principal Components and Subspaces*. International Journal of Neural Systems 1, 61-68.

Sanger T. D. (1989a). *Optimal Unsupervised Learning in a Single-Layer Linear Feedforward Neural Network*. Neural Networks 2, 459-473.

Sanger T. D. (1989b). *An Optimality Principle of Unsupervised Learning*. In Advances in Neural Information Processing Systems I (Denver 1988), ed. D.S. Touretzky, 11-19. San Mateo: Morgan Kaufmann.

BIBLIOGRAFIA GENERAL

Freeman J. A. y Skapura D. M. (1992). *Neural Networks: Algorithms, Applications and Programming Techniques*. Addison-WesleyPublishing Company.

Goonatilake S. y Treleaven P. (1995). *Intelligent Systems for Finance and Business*. Editado por Goonatilake y Treleaven. John Wiley & Sons.

Hertz J., Krogh A., Palmer R. (1990). *Introduction to the Theory of Neural Computation;* Santa Fé Institute Editorial Board.

Welstead, S. (1994). *Neural Network and Fuzzy Logic Applications in C/C++*. John Wiley and Sons.

APENDICE A

IMPLEMENTACION DE ALGORITMO DE BACKPROPAGATION

Todas las implementaciones fueron hechas en FoxPro 2.5 para DOS.

Los programas se realizaron en este lenguaje porque permite un fácil manejo de los datos de entrada y de los resultados, que se mantienen en tablas. Puede ocurrir que la rapidez de los cálculos matemáticos sea menor que en otro lenguaje más utilizado para modelos matemáticos pero, dado que la finalidad del trabajo es comparar resultados entre diversas experiencias y todas están programadas en el mismo lenguaje, optimizar el tiempo que insume el entrenamiento no es de fundamental importancia.

A.1 ALGORITMO DE BACKPROPAGATION CON 1 NIVEL OCULTO

```
parameters pbasetest, pbasepat
* pbasetest: tabla con conjunto de datos para testeo
* pbasepat: tabla con conjunto de datos para entrenamiento

SET SAFETY OFF
SET DELE ON
SET TALK OFF
CLOSE ALL

* Los parámetros que siguen se cambian para cada corrida según lo que se desee
* probar
n = 0.2 && parámetro de velocidad de aprendizaje
beta = 0.9
alfa = 0.5 && parámetro de momentos
aa = 0.01
ab = 0.03
k_ok=4
k_mal=3

dim_input  = 11 && Unidades de entrada
dim_output = 1 && Unidades de salida
dim_hidden = 11 && Unidades nivel oculto

DIMENSION entrada (dim_input+1) && entradas y respuesta en el último lugar
DIMENSION p_inp_hid(dim_input, dim_hidden), delta_sal(dim_input, dim_hidden)
DIMENSION p_hid_out(dim_hidden)
STORE 0 TO inp_salida, salida, term_err, delta_sal
DIMENSION input_hid(dim_hidden)
DIMENSION output_hid(dim_hidden)
DIMENSION error_hid(dim_hidden), delta_hid(dim_hidden)
STORE 0 TO input_out, output_hid
STORE 0 TO entrada, p_inp_hid, p_hid_out, salida, input_hid, input_out
STORE 0 TO delta_hid, error_hid
STORE 0 TO mmalc, mmalnc, mtot_err

* Valores para la función de costo para seleccionar el peso de los casos
* confiables y no confiables en el resultado
DIMENSION mmaltant(5), mfcosto(5)
mmaltant = 999999
mfcosto(1) = 0.96
mfcosto(2) = 0.75
mfcosto(3) = 0.50
mfcosto(4) = 0.25
mfcosto(5) = 0.04
```

```
* Tabla de datos de entrenamiento
USE (pbasepat) IN 0 ALIAS patterns
* Tabla de resultados por época
USE resul2 IN 0
* Tabla de datos de testeo
USE (pbasetest) IN 0 ALIAS test
FOR i=1 TO 5
   mind = STR(i,1)
   USE pesoo&mind IN 0
   USE pesoh&mind IN 0
ENDFOR

SELE resul2
GO BOTT
mvez = vez + 1

cant_mal = 0
cant_bien = 0
tot_err = 0
tot_err_ant = 99999
STORE 0 TO err_ok, err_mal
STORE 9000 TO err_min_ant, err_min
STORE 0 TO memalc, memalnc

SELECT patterns
COUNT TO tot_casos
GO TOP

iter = 0
* Inicializa pesos
DO inic_pesos
* Pasa por todos los patrones de entrada mientras no se hayan aprendido todos
DO WHILE cant_bien < tot_casos
      SELECT patterns
      * Desordena los patrones de entrada
      DO mezcla
      GO TOP
      iter = iter + 1
      tot_err = 0
      SCAN
         * Lee patrón de entrada
         SCATTER TO entrada
         * Calcula salidas
         DO calc_salidas
         * Calcula errores
         DO calc_term_err
         * Adapta pesos
         DO adapt_pesos
      ENDSCAN

      * Cuenta casos acertados
      DO cuantos_bien
      * Guarda resultado de la época
      DO guarda_iter
      * Modifica parámetros según los resultados
      DO adaptive

      tot_err_ant = tot_err

      SELECT patterns
```

```
ENDDO


CLOSE ALL
RETURN


PROCEDURE mezcla
* Desordena patrones de entrada para cada época
PRIVATE msemilla
SET ORDER TO
msemilla = RAND(-1)
SCAN
    REPLACE orden WITH RAND() * 1000
ENDSCAN
SET ORDER TO orden
RETURN

FUNCTION calc_error
* Calcula error
RETURN ((entrada(dim_input+1) - salida)^2)

PROCEDURE inic_pesos
* Genera pesos iniciales en forma random
PRIVATE i,j, msemilla
msemilla = RAND(-1)
FOR i=1 TO dim_input
    FOR j=1 TO dim_hidden
       p_inp_hid(i,j) = ((RAND() - 0.5) / 1.5) + 0.01
    ENDFOR
ENDFOR
msemilla = RAND(-1)
FOR i=1 TO dim_hidden
    p_hid_out(i) = ((RAND() - 0.5) / 1.5) + 0.01
ENDFOR
RETURN


PROCEDURE calc_salidas
PRIVATE i,j
* Calcula valores de salida de capa oculta
FOR j=1 TO dim_hidden
   input_hid(j) = 0
   FOR i=1 TO dim_input
      input_hid(j) = input_hid(j) + (entrada(i) * p_inp_hid(i,j))
   ENDFOR
   output_hid(j) = tanh(input_hid(j))
ENDFOR

* Calcula valores de salida de capa salida
inp_salida = 0
FOR i=1 TO dim_hidden
   inp_salida = inp_salida + (output_hid(i) * p_hid_out(i))
ENDFOR

salida = tanh(inp_salida)

RETURN

PROCEDURE calc_term_err
```

```
* Calcula error para c/unidad de salida
term_err = (entrada(dim_input+1) - salida) * beta * (1 - (salida^2))

* Calcula errores para unidades ocultas
FOR i=1 TO dim_hidden
   p_hid_out(i)
   error_hid (i) = beta * (1 - (output_hid(i)^2)) * term_err * p_hid_out(i)
ENDFOR

RETURN

PROCEDURE adapt_pesos
private delta_act
* Actualiza pesos de la capa oculta a la salida
FOR j=1 TO dim_hidden
   delta_act = ( n * term_err * output_hid(j) ) + (alfa * delta_hid(j))
   p_hid_out(j) = p_hid_out(j) + delta_act
   delta_hid(j) = delta_act
ENDFOR

* Actualiza pesos de la capa entrada a la oculta
FOR i=1 TO dim_input
   FOR j=1 TO dim_hidden
      delta_act = ( n * error_hid(j) * entrada(i) ) + (alfa * delta_sal(i,j))
      p_inp_hid(i,j) = p_inp_hid(i,j) + delta_act
      delta_sal(i,j) = delta_act
   ENDFOR
ENDFOR


RETURN


FUNCTION tanh
* Calcula tangente hiperbólica
PARAMETERS x
RETURN (exp(beta * x) - exp(-beta * x) ) / (exp(beta * x) + exp(-beta * x) )


PROCEDURE  cuantos_bien
* Calcula resultados de la época
STORE 0 TO cant_bien, memalc, memalnc, tot_err
SELECT patterns
GO TOP
SCAN
   SCATTER TO entrada
   DO calc_salidas
   error_act = calc_error()
   tot_err = tot_err + error_act
   IF ABS(salida-entrada(dim_input+1)) < 1
      cant_bien = cant_bien + 1
   ELSE
      IF entrada(dim_input+1) = 1
         memalnc = memalnc + 1
      ELSE
         memalc = memalc + 1
      ENDIF
   ENDIF
ENDSCAN
tot_err = tot_err / 2
RETURN
```

```
PROCEDURE adaptive
* Modifica el parámetro de aprendizaje n según método de Parámetros Adaptivos
IF iter > 1
   IF tot_err <= tot_err_ant
      err_mal = 0
      err_ok = err_ok + 1
   ELSE
      err_mal = err_mal + 1
      err_ok = 0
   ENDIF
ENDIF

IF err_ok = k_ok
   n = n + aa
   err_ok = 0
ENDIF

IF err_mal = k_mal
   err_mal = 0
   n = n - (ab * n)
ENDIF

RETURN


PROCEDURE guarda_iter
* Guarda resultados de la época
PRIVATE i, mfc, mquepeso
DO testear
mquepeso = ''
FOR i=1 TO 5
      mfc = (mmalc * mfcosto(i)) + (mmalnc * (1 - mfcosto(i)))
      IF mfc < mmaltant(i)
         DO gpesos WITH STR(i,1)
         mmaltant(i) = mfc
         mquepeso = mquepeso + STR(i,1)
      ENDIF
ENDFOR
INSERT INTO resul2 ;
(iteracion, cantbien, error, n, vez, maltnc, ;
maltc, errortot, malec, malenc, pesos) ;
VALUES (iter, cant_bien, tot_err, m.n, mvez, mmalnc, ;
mmalc, mtot_err, memalc, memalnc, mquepeso)
?ITER, cant_bien, tot_err, str(mtot_err,12,6), 'Pesos ' + mquepeso
RETURN

PROCEDURE testear
* Testea resultados sobre datos de testeo
PRIVATE a, b, merror
SELECT test
GO TOP
STORE 0 TO a, b, mtot_err
SCAN
   SCATTER TO entrada
   DO calc_salidas
   merror = calc_error()
   mtot_err = mtot_err + merror
   if abs(test.resu - m.salida) >= 1
        if test.resu = 1
            b = b + 1
```

```
        else
            a = a + 1
      endif
   endif
ENDSCAN

mtot_err = mtot_err / 2
mmalnc = b
mmalc  = a
RETURN


PROCEDURE gpesos
PARAMETER pk
* Guarda los pesos de las mejores configuraciones
PRIVATE i, j, mj
SELECT pesoo&pk
ZAP
SELECT pesoh&pk
ZAP

SELECT pesoo&pk
APPEND BLANK
GATHER FROM p_hid_out

SELECT pesoh&pk
FOR i=1 TO dim_input
   APPEND BLANK
   FOR j=1 TO dim_hidden
      mj= ALLTRIM(STR(j))
      REPLACE p&mj WITH p_inp_hid(i,j)
   ENDFOR
ENDFOR

RETURN
```

A.2 ALGORITMO DE BACKPROPAGATION CON 2 NIVELES OCULTOS

```
parameters pbasetest, pbasepat
SET SAFETY OFF
SET DELE ON
SET TALK OFF
CLOSE ALL

* Estos parámetros se setean para cada corrida
n = 0.2 && parámetro de velocidad de aprendizaje
beta = 0.4
alfa = 0.1 && parámetro de momentos
aa = 0.01
ab = 0.02
k_ok=5
k_mal=2
momentos = "S" && si usa momentos
adaptive = "S" && si usa parámetros adaptivos


dim_input  = 11 && Unidades de entrada
dim_output = 1 && Unidades de salida
dim_hid1 = 10 && Unidades del primer nivel oculto
dim_hid2 = 5 && Unidades del segundo nivel oculto

DIMENSION entrada (dim_input+1) && entrada + 1 nodo de salida
DIMENSION p_inp_hid(dim_input, dim_hid1), delta_sal(dim_input, dim_hid1)
DIMENSION p_hid_out(dim_hid2), p_hid1_hid2(dim_hid1,dim_hid2)
STORE 0 TO inp_salida, salida, term_err, delta_sal
DIMENSION input_hid(dim_hid1),hid1_hid2(dim_hid2)
DIMENSION out_hid(dim_hid1),  out_hid2(dim_hid2)
DIMENSION error_hid1(dim_hid1),error_hid2(dim_hid2)
DIMENSION delta_hid1(dim_hid1,dim_hid2), delta_hid2(dim_hid2)
STORE 0 TO input_out, out_hid,p_hid1_hid2, out_hid2
STORE 0 TO entrada, p_inp_hid, p_hid_out, salida, input_hid, input_out
STORE 0 TO delta_hid1,delta_hid2, error_hid1,error_hid2,hid1_hid2
STORE 0 TO mmalc, mmalnc, mtot_err

* Tabla de datos de entrada
USE (pbasepat) IN 0 ALIAS patterns
* Tabla de pesos de unidades del nievl oculto 2 a las de salida
USE pesoso IN 0
* Tabla de pesos de unidades de entrada a unidades del nivel oculto 1
USE pesosh IN 0
* Tabla de pesos de unidades del nivel 1 al nivel oculto 2
USE pesosh2 IN 0
* Tabla de resultados por época
USE resul2 IN 0
* Tabla de datos de testeo
USE (pbasetest) IN 0 ALIAS test


SELECT resul2
GO BOTT
mvez = vez + 1

STORE 0 TO memalc, memalnc
cant_mal = 0
```

```
cant_bien = 0
tot_err = 0
tot_err_ant = 99999
STORE 0 TO err_ok, err_mal
STORE 9000 TO err_min_ant, err_min

SELECT patterns
COUNT TO tot_casos
GO TOP

iter = 0
* Inicializa pesos
DO inic_pesos
* Recorre patrones mientras no den todos bien
DO WHILE cant_bien < tot_casos
      SELECT patterns
      * Desordena patrones
      DO mezcla
      GO TOP
      iter = iter + 1
      tot_err = 0
      SCAN
         * Lee patrón de entrada
         SCATTER TO entrada
         * Calcula salidas
         DO calc_salidas
         * Calcula errores
         DO calc_term_err
         * Adapta pesos
         DO adapt_pesos
      ENDSCAN

      * Calcula resultados de la época
      DO cuantos_bien
      * Graba resultados de la época
      DO guarda_iter

      * Adapta parámetros según resultados
      IF adaptive = "S"
         DO adaptive
      ENDIF

      tot_err_ant = tot_err

      SELECT patterns
ENDDO

CLOSE ALL
RETURN

PROCEDURE guarda_iter
* Graba resultados de la época
DO testear
INSERT INTO resul2 ;
 (iteracion, cantbien, error, n, vez, maltnc, ;
maltc, errortot, malec, malenc) ;
VALUES (iter, cant_bien, tot_err, m.n, mvez, mmalnc, ;
mmalc, mtot_err, memalc, memalnc)
DO grab_pesos
?ITER, cant_bien, tot_err, STR(n,10,8), str(alfa,3,1), str(mtot_err,12,6)
RETURN
```

```
PROCEDURE mezcla
* Desordena patrones para que no se entrene en el mismo orden
PRIVATE msemilla
SET ORDER TO
msemilla = RAND(-1)
SCAN
    REPLACE orden WITH RAND() * 1000
ENDSCAN
SET ORDER TO orden
RETURN

FUNCTION calc_error
* Calcula error
RETURN ((entrada(dim_input+1) - salida)^2)

PROCEDURE inic_pesos
* Genera pesos iniciales en forma random
PRIVATE i,j, msemilla
msemilla = RAND(-1)
FOR i=1 TO dim_input
    FOR j=1 TO dim_hid1
       p_inp_hid(i,j) = ((RAND() - 0.5) / 1.5) + 0.01
    ENDFOR
ENDFOR
msemilla = RAND(-1)
FOR i=1 TO dim_hid1
    FOR j=1 TO dim_hid2
       p_hid1_hid2(i,j) = ((RAND() - 0.5) / 1.5) + 0.01
    ENDFOR
ENDFOR
msemilla = RAND(-1)
FOR i=1 TO dim_hid2
    p_hid_out(i) = ((RAND() - 0.5) / 1.5) + 0.01
ENDFOR
RETURN

PROCEDURE calc_salidas
PRIVATE i,j
* Calcula valores de salida de capas ocultas
FOR j=1 TO dim_hid1
    input_hid(j) = 0
      FOR i=1 TO dim_input
            input_hid(j) = input_hid(j) + (entrada(i) * p_inp_hid(i,j))
      ENDFOR
    out_hid(j) = tanh(input_hid(j))
ENDFOR
FOR j=1 to dim_hid2
    hid1_hid2(j) = 0
    FOR i=1 TO dim_hid1
        hid1_hid2(j) = hid1_hid2(j) + (out_hid(i) * p_hid1_hid2(i,j))
    ENDFOR
    out_hid2(j) = tanh(hid1_hid2(j))
ENDFOR

* Calcula valores de salida de capa salida
inp_salida = 0
FOR i=1 TO dim_hid2
      inp_salida = inp_salida + (out_hid2(i) * p_hid_out(i))
ENDFOR
```

```
salida = tanh(inp_salida)

RETURN


PROCEDURE calc_term_err
PRIVATE term_err2
* Calcula error para c/unidad de salida
term_err = (entrada(dim_input+1) - salida) * beta * (1 - (salida ^ 2))
* Calcula errores para unidades ocultas
FOR i=1 TO dim_hid2
  error_hid2 (i) = beta * (1 - (out_hid2(i) ^ 2)) * term_err * p_hid_out(i)
ENDFOR

FOR i=1 TO dim_hid1
      term_err2 = 0
      FOR j=1 TO dim_hid2
            term_err2 = term_err2 + (error_hid2(j) * p_hid1_hid2(i,j))
      ENDFOR
      error_hid1 (i) = beta * (1 - (out_hid(i) ^ 2)) * term_err2
ENDFOR
RETURN

PROCEDURE adapt_pesos
* Actualiza pesos de la capa oculta a la salida
FOR j=1 TO dim_hid2
   if momentos = "S"
         p_hid_out(j) = p_hid_out(j) + ( n * term_err * out_hid2(j) ) + ;
                     (alfa * delta_hid2(j))
         delta_hid2(j) = n * term_err * out_hid2(j) + (alfa * delta_hid2(j))
      else
         p_hid_out(j) = p_hid_out(j) + ( n * term_err * out_hid2(j) )
      endif
ENDFOR

* Actualiza pesos de la capa oculta1 a la oculta2
FOR i=1 TO dim_hid1
  FOR j=1 TO dim_hid2
   IF momentos = "S"
      p_hid1_hid2(i,j) = p_hid1_hid2(i,j) + ;
      ( n * error_hid2(j) * out_hid(i) ) + ;
                  (alfa * delta_hid1(i,j))
         delta_hid1(i,j) = n * error_hid2(j) * out_hid(i) + ;
         (alfa * delta_hid1(i,j))
      ELSE
      p_hid1_hid2(i,j) = p_hid1_hid2(i,j) + ;
      ( n * error_hid2(j) * out_hid(i) )
      ENDIF
 ENDFOR
ENDFOR

* Actualiza pesos de la capa entrada a la oculta
FOR i=1 TO dim_input
   FOR j=1 TO dim_hid1
      IF momentos = "S"
         p_inp_hid(i,j) = p_inp_hid(i,j) + ;
                        ( n * error_hid1(j) * entrada(i) ) + ;
                        (alfa * delta_sal(i,j))
         delta_sal(i,j) = n * error_hid1(j) * entrada(i) + ;
                         (alfa * delta_sal(i,j))
      ELSE
```

```
         p_inp_hid(i,j) = p_inp_hid(i,j) + ( n * error_hid1(j) * entrada(i) )
      ENDIF
   ENDFOR
ENDFOR
RETURN

PROCEDURE grab_pesos
* Guarda los pesos finales
PRIVATE i, j, mj
SELECT pesoso
ZAP
SELECT pesosh
ZAP
SELECT pesosh2
ZAP

SELECT pesoso
APPEND BLANK
GATHER FROM p_hid_out

SELECT pesosh
FOR i=1 TO dim_input
   APPEND BLANK
   FOR j=1 TO dim_hid1
      mj= ALLTRIM(STR(j))
      REPLACE p&mj WITH p_inp_hid(i,j)
   ENDFOR
ENDFOR

SELECT pesosh2
FOR i=1 TO dim_hid1
   APPEND BLANK
   FOR j=1 TO dim_hid2
      mj= ALLTRIM(STR(j))
      REPLACE p&mj WITH p_hid1_hid2(i,j)
   ENDFOR
ENDFOR
RETURN

FUNCTION tanh
* Calcula tangente hiperbólica
PARAMETERS x
RETURN (exp(beta * x) - exp(-beta * x) ) / (exp(beta * x) + exp(-beta * x) )

PROCEDURE testear
* Calcula y guarda resultados para el conjunto de testeo luego de cada época
PRIVATE a, b, merror
SELECT test
GO TOP
STORE 0 TO a, b, mtot_err
SCAN
   SCATTER TO entrada
   DO calc_salidas
   merror = calc_error()
   mtot_err = mtot_err + merror
   if abs(test.resu - m.salida) >= 1
      if test.resu = 1
         b = b + 1
      else
         a = a + 1
      endif
```

```
   endif
ENDSCAN
mtot_err = mtot_err / 2
mmalnc = b
mmalc  = a
RETURN



PROCEDURE  cuantos_bien
* Calcula cantidad de aciertos
STORE 0 TO cant_bien, memalc, memalnc, tot_err
SELECT patterns
GO TOP
SCAN
   SCATTER TO entrada
   DO calc_salidas
   error_act = calc_error()
   tot_err = tot_err + error_act
   IF ABS(salida-entrada(dim_input+1)) < 1
      cant_bien = cant_bien + 1
   ELSE
      IF entrada(dim_input+1) = 1
            memalnc = memalnc + 1
        ELSE
            memalc = memalc + 1
      ENDIF
   ENDIF
ENDSCAN
tot_err = tot_err / 2
RETURN

PROCEDURE adaptive
* Adapta el parámetro n con el método de Parámetros Adaptivos
IF iter > 1
   IF tot_err <= tot_err_ant
      err_mal = 0
      err_ok = err_ok + 1
   ELSE
      err_mal = err_mal + 1
      err_ok = 0
   ENDIF
ENDIF

IF err_ok = k_ok and n < 0.25
   n = n + aa
   err_ok = 0
ENDIF

IF err_mal = k_mal and n > 0.1
   err_mal = 0
   n = n - (ab * n)
ENDIF

RETURN
```

A.3 TABLAS UTILIZADAS EN BACKPROPAGATION

Tabla **PATTERNS**
Contiene los patrones de entrenamiento.
Su estructura es la siguiente:

	Nombre	Tipo	Long.	Dec.	Descripción
1	V1	Numeric	5	2	Sexo
2	V2	Numeric	5	2	Edad
3	V3	Numeric	5	2	Volumen de consultas
4	V4	Numeric	5	2	Antig. última consulta
5	V5	Numeric	5	2	Problemas 1
6	V6	Numeric	5	2	Problemas 2
7	V7	Numeric	5	2	Problemas 3
8	V8	Numeric	5	2	Problemas 4
9	V9	Numeric	5	2	Problemas 5
10	V10	Numeric	5	2	Relacionados
11	V11	Numeric	5	2	Variable adicional para umbral
12	RESU	Numeric	5	2	Resultado real
13	RESUBP	Numeric	6	2	Resultado BP
14	ORDEN	Numeric	6		Orden de proceso en época actual

Tabla **TEST**
Contiene los patrones de testeo.
Su estructura es la misma que la de PATTERNS.

Tabla **RESUL2**
Contiene los resultados por prueba y época.

Su estructura es la siguiente:

	Nombre	Tipo	Long.	Dec.	Descripción
1	ITERACION	Numeric	4		Nro. época
2	CANTBIEN	Numeric	4		Cantidad de aciertos en entrenam.
3	ERROR	Numeric	16	12	Error calculado en entrenam.
4	N	Numeric	16	12	Valor del parámetro eta.
5	VEZ	Numeric	4		Nro. de prueba
6	MALTNC	Numeric	5		Cantidad de no conf. mal en testeo
7	MALTC	Numeric	5		Cantidad de conf. mal en testeo
8	ERRORTOT	Numeric	16	12	Error calculado en testeo
9	MALEC	Numeric	5		Cantidad de conf. mal en entren.
10	MALENC	Numeric	5		Cantidad de no conf. mal en entren.
11	PESOS	Character	5		Indicación de que resultó mejor configuración y para qué casos

Tabla **RESULTAD**
Contiene los parámetros utilizados y un resumen de resultados por prueba para utilizar en las comparaciones.

Su estructura es la siguiente:

	Nombre	Tipo	Long.	Dec.	Descripción
1	NRO	Numeric	4		Nro. de prueba
2	DESCRIP	Character	60		Descripción
3	EJEMPLOS	Numeric	4		Cantidad de patrones en entrenam.
4	EJNC	Numeric	4		Cantidad de patrones NC en entr.
5	N	Numeric	6	2	Valor del parámetro eta inicial
6	ALFA	Numeric	6	2	Valor del parámetro alfa
7	BETA	Numeric	6	2	Valor del parámetro beta
8	ADAPTIVE	Logical	1		S/N usa parámetros adaptivos
9	HIDDEN1	Numeric	3		Cantidad de unidades ocultas nivel 1
10	HIDDEN2	Numeric	3		Cantidad de unidades ocultas nivel 2
11	MALNC	Numeric	6	2	Cantidad de no conf. mal entrenam.
12	MALC	Numeric	6	2	Cantidad de conf. mal entrenam.
13	MALT	Numeric	6	2	Cantidad total mal testeo
14	MALTNC	Numeric	6	2	Cantidad de no conf. mal testeo
15	MALTC	Numeric	6	2	Cantidad de conf. mal testeo

APENDICE B

IMPLEMENTACION DE ALGORITMO DE KOHONEN

B.1 ALGORITMO APRENDIZAJE DE KOHONEN

```
SET SAFETY OFF
SET DELE ON
SET TALK OFF
CLOSE ALL

* Tabla de datos de entrada
USE patterns alias patterns IN 0
* Tabla de pesos que resultan
USE pesos IN 0

dimmat = 10 && dimensión de la matriz
dimvec = 10 && dimensión del vector de entrada
DIMENSION pesos(dimmat*dimmat, dimvec)
DIMENSION entrada(dimvec)
winner = 0
pesos = 0
vecindad = dimmat

* Genera pesos iniciales en forma random
DO inicializar

SELECT patterns
* Mezcla los ejemplos de entrada para no tomarlos siempre en el mismo orden
DO mezcla

* Recorre cada uno de los patrones de entrada
GO TOP
nro = 1
SCAN
   ? nro
   tiempo = 1
   * Lee entrada
   SCATTER TO entrada
   * Actualiza los pesos hasta que haya 1 sóla unidad ganadora
   DO WHILE actualizar() > 1
      tiempo = tiempo + 1
   ENDDO
   SELECT patterns
   nro = nro + 1
ENDSCAN

* Vuelve a recorrer los casos de entrada 100 veces para que la red se estabilice
for i=1 to 100
   SELECT patterns
   SCAN
      ? i,nro
      tiempo = 1
      SCATTER TO entrada
      =actualizar()
      tiempo = tiempo + 1
      SELECT patterns
      nro = nro + 1
   ENDSCAN
endfor

* Graba pesos finales
DO grabpesos
CLOSE ALL
```

```
SET TALK ON
RETURN

PROCEDURE grabpesos
* Guarda los pesos finales
SELECT pesos
ZAP
FOR i=1 TO dimmat*dimmat
   APPEND BLANK
   FOR j=1 TO dimvec
      mj= ALLTRIM(STR(j))
      REPLACE p&mj WITH pesos(i,j)
   ENDFOR
ENDFOR
RETURN


PROCEDURE inicializar
* Genera pesos iniciales en forma random
PRIVATE i,j, msemilla
msemilla = RAND(-1)
FOR i=1 TO dimmat*dimmat
    FOR j=1 TO dimvec
       pesos(i,j) = RAND()
    ENDFOR
ENDFOR
RETURN

FUNCTION actualizar
* Actualiza los pesos de la vecindad de la unidad ganadora
PRIVATE unit, upd, i, j, finicial, ffinal, colinicial,;
        colfinal, fila, colum
winner = propagate()
upd = 0
fila = 0
colum = 0
DO transforma WITH winner,fila,colum

FOR i=1 TO dimmat
    FOR j=1  TO dimmat
        unit = (i - 1) * dimmat + j
        FOR k=1 TO dimvec
            mactu = fn() * ( fdelta(i, j, fila, colum) * ;
                    ( entrada(k) - pesos(unit, k) ))
            pesos(unit, k) = pesos(unit, k) + mactu
            IF round(mactu,15) # 0
               upd = upd + 1
            ENDIF
        ENDFOR
    ENDFOR
ENDFOR
RETURN upd


FUNCTION propagate
* Selecciona la unidad ganadora
PRIVATE smallest, i, mag, winner
smallest = 10000
winner = 0
FOR i=1 TO dimmat*dimmat
   mag = propag(i)
```

```
   IF mag < smallest
      winner = i
      smallest = mag
   ENDIF
ENDFOR

RETURN winner

FUNCTION propag
* Calcula distancia del vector de entrada a los pesos
PARAMETER unit
PRIVATE mag, i, suma
suma = 0
FOR i=1 TO dimvec
   suma = suma + (entrada(i) - pesos(unit, i)) ^ 2
ENDFOR
mag = SQRT(suma)
RETURN mag

FUNCTION fdelta
* Funcion que decrece con el tiempo para actualizar los pesos
PARAMETERS runi1, cuni1, fila, colum
PRIVATE aux, mresu, fila, colum, dist, sigma
dist = (runi1 - fila) ^ 2 + (cuni1 - colum) ^ 2
dist = SQRT(dist)
sigma = fn()
aux = - (dist ^ 2) / (2 * (sigma ^ 2))
mresu = EXP(aux)
RETURN mresu

FUNCTION fn
* Funcion que decrece con el tiempo
PRIVATE n
n = 1 / tiempo
RETURN n

PROCEDURE transforma
* Calcula ubicacion de una unidad en fila y columna de la matriz
PARAMETER unidad,fila,colum
fila = INT(( unidad -1 ) / dimmat) + 1
colum = MOD(( unidad -1 ) , dimmat) + 1
RETURN

PROCEDURE mezcla
* Mezcla los ejemplos de entrada para no tomarlos siempre en el mismo orden
PRIVATE msemilla
SET ORDER TO
msemilla = RAND(-1)
SCAN
    REPLACE orden WITH RAND() * 1000
ENDSCAN
SET ORDER TO orden
RETURN
```

B.2 ALGORITMO PARA UBICACIÓN DE PATRONES EN CLUSTERS

```
* Ubica los patrones en los clusters que resultan de los pesos guardados en el
* entrenamiento y asigna clases a los clusters según proporción de confiables y
* no confiables que caen en los mismos.
```

```
PARAMETERS ptablai, ptablao
* ptablai = tabla de entrada (se corre para la tabla con patrones de
* entrenamiento y luego para la tabla con patrones de testeo)
* ptablao = tabla de salida con un registro por cluster

SET DELE ON
SET TALK OFF
CLOSE DATA
CLEAR

dimvec = 10 & dimensión de los patrones
dimmat = 10 & dimensión de la matriz

DIMENSION entrada(dimvec+1), pesos(dimmat*dimmat,dimvec)

* En la tabla Pesos está la matriz de pesos
USE pesos IN 0 ALIAS pesos

* En la tabla ptablai están los patrones a ubicar
USE (ptablai) IN 0 alias test

* Lee pesos a memoria
DO leepesos

SELECT test
nro=1
* Recorre patrones
SCAN
     ?nro
     * Lee patrón
     SCATTER TO entrada
     * Calcula el resultado
     aa=propagate()
     entrada(dimvec+1)=aa
     * Guarda el resultado
     SELECT test
     replace resul2 WITH aa
     nro=nro+1
ENDSCAN

* Asigna clases a los clusters según proporciones de conf. y no conf.
DO resumen WITH ptablai, ptablao

CLOSE ALL
SET TALK ON
RETURN


PROCEDURE leepesos
* Lee pesos del archivo a memoria
SELECT pesos
i=0
SCAN
   i = i+1
   FOR j=1 TO dimvec
      mj= ALLTRIM(STR(j))
      pesos(i,j) = pesos.p&mj
   ENDFOR
ENDSCAN
RETURN
```

```
FUNCTION propagate
* Calcula unidad ganadora (con menor distancia al patrón de entrada)
PRIVATE smallest, i, mag, winner
smallest = 10000
winner = 0
FOR i=1 TO dimmat*dimmat
   mag = propag(i)
   IF mag < smallest
      winner = i
      smallest = mag
   ENDIF
ENDFOR
* Devuelve ganadora
RETURN winner


FUNCTION propag
* Calcula distancia de la unidad parámetro a la entrada
PARAMETER unit
PRIVATE mag, i, suma
suma = 0
FOR i=1 TO dimvec
   suma = suma + (entrada(i) - pesos(unit, i)) ^ 2
ENDFOR
mag = SQRT(suma)
RETURN mag


PROCEDURE resumen
PARAMETER ptablai, ptablao
* Cuenta confiables y no confiables por cluster
SELECT resul2 as resul, ;
SUM(iif(resu=1,1,0)) AS nconf, SUM(iif(resu=-1,1,0)) AS conf,;
000.00 as pc, 000.00 as pnc, ' ' as clase ;
FROM (ptablai) ;
GROUP BY resul2 ;
INTO DBF (ptablao)

* La signación de clases siguiente sirve sólo para los patrones del
* entrenamiento, para los del testeo se asigna de otra manera

* Calcula proporciones de cada clase
REPLACE ALL pc  WITH conf * 100 / (conf +nconf),;
            pnc WITH nconf * 100 / (conf +nconf)
* Asigna C (Confiable) o N (No confiable) al cluster
REPLACE clase WITH 'C' FOR pc >= 50
REPLACE clase WITH 'N' FOR pnc > 50
INDEX ON resul TAG resul
RETURN
```

B.3 ALGORITMO PARA CLASIFICACIÓN DE PATRONES EN CLASES

```
PARAMETERS ptablae, ptablat
CLOSE DATA
pdim = 10

USE (ptablat) ALIAS total IN 0
USE (ptablae) ALIAS entren ORDER resul IN 0
```

```
* Asigna clase a cada cluster según la que le tocó en el entrenamiento
SELECT total
SET RELATION TO resul INTO entren
REPLACE ALL clase WITH ''
REPLACE ALL clase WITH entren.clase
SET RELATION TO

* Guarda matriz resultante de entrenamiento en vectores
DIMENSION mfila(pdim)
SELECT entren
GO TOP
FOR i = 1 TO pdim
    mfila(i) = ''
      FOR col = 1 TO pdim
            IF SEEK (((i -1)*pdim) + col)
                  mfila(i) = mfila(i) + LEFT(clase,1)
            ELSE
                  mfila(i) = mfila(i) + ' '
            ENDIF
      ENDFOR
ENDFOR

* Asigna clases a clusters que no tenían casos en el entrenamiento
* Lo hace según las letras de las casillas vecinas

SELECT total
SCAN FOR EMPTY(clase)
      mrow = INT(resul / pdim) + 1
      mcol = IIF(MOD(resul,pdim)=0, pdim, MOD(resul,pdim))
      STORE 0 TO mbien, mmal
      IF mrow>1
            mresu = SUBSTR(mfila(mrow-1), mcol, 1)
            IF mresu = 'N'
                  mmal = mmal + 1
            ENDIF
            IF mresu = 'C'
                  mbien = mbien + 1
            ENDIF
      ENDIF
      IF mcol > 1
            mresu = SUBSTR(mfila(mrow), mcol-1, 1)
            IF mresu = 'N'
                  mmal = mmal + 1
            ENDIF
            IF mresu = 'C'
                  mbien = mbien + 1
            ENDIF
      ENDIF
      IF mrow<pdim
            mresu = SUBSTR(mfila(mrow+1), mcol, 1)
            IF mresu = 'N'
                  mmal = mmal + 1
            ENDIF
            IF mresu = 'C'
                  mbien = mbien + 1
            ENDIF
      ENDIF
      IF mcol < pdim
            mresu = SUBSTR(mfila(mrow), mcol+1, 1)
            IF mresu = 'N'
                  mmal = mmal + 1
```

```
            ENDIF
            IF mresu = 'C'
                  mbien = mbien + 1
            ENDIF
      ENDIF

      IF mbien > mmal
            REPL CLASE WITH 'C'
      ENDIF
      IF mbien < mmal
            REPL CLASE WITH 'N'
      ENDIF
ENDSCAN
RETURN
```

B.4 TABLAS UTILIZADAS EN KOHONEN

Tabla **PATTERNS**

Contiene los patrones de entrenamiento.
Su estructura es la siguiente:

1	NSEXO	Numeric	5	2	Sexo
2	EDAD	Numeric	5	2	Edad
3	VOLCONS	Numeric	5	2	Volumen de consultas
4	ULTCONS	Numeric	5	2	Antigüedad de la última consulta
5	P1	Numeric	5	2	Problemas tiempo 1
6	P2	Numeric	5	2	Problemas tiempo 2
7	P3	Numeric	5	2	Problemas tiempo 3
8	P4	Numeric	5	2	Problemas tiempo 4
9	P5	Numeric	5	2	Problemas tiempo 5
10	REL	Numeric	5	2	Relacionados
11	RESU	Numeric	5	2	Conf/No confiable
12	ORDEN	Numeric	4		Orden de proceso
13	RESUL2	Numeric	6	2	Nro. de cluster
14	CLASE	Character	2		Clase que le corresponde al cluster

Tabla **PESOS**

Contiene los pesos que resultan de la corrida
Su estructura es la siguiente:

	Nombre	**Tipo**	**Long.**	**Dec.**
1	P1	Float	20	15
2	P2	Float	20	15
3	P3	Float	20	15
4	P4	Float	20	15
5	P5	Float	20	15
6	P6	Float	20	15
7	P7	Float	20	15
8	P8	Float	20	15
9	P9	Float	20	15
10	P10	Float	20	15

La cantidad de variables de peso y la cantidad de registros de la tabla van a ser iguales a la dimensión de la matriz.

APENDICE C

IMPLEMENTACION DE ALGORITMO DE APRENDIZAJE HEBBIANO

C.1 ALGORITMO APRENDIZAJE HEBBIANO

```
PARAMETERS poper, pregla
* poper = 1 --> entrenar
* poper = 2 --> testear
* pregla = 1 --> Sanger
* pregla = 2 --> Oja

SET SAFETY OFF
SET DELE ON
SET TALK OFF
CLOSE ALL

* Dimvec es la dimensión del vector de entrada, Dimsal es la dimensión de la
* salida

dimvec = 10
dimsal = 4
eta = 0.3

*La tabla con alias DATOS contiene los patrones de entrenamiento
*La tabla PESOSOJA va a contener los pesos finales

USE datosnc alias datos IN 0
USE pesosoja IN 0

DIMENSION pesosoja(dimvec, dimsal)
DIMENSION entrada(dimvec)
DIMENSION v(dimsal)
v=0
pesosoja = 0

IF poper =1
   * Entrena la red
   DO entrenar
ELSE
   * Testea la red
   DO testear
ENDIF

CLOSE ALL
SET TALK ON
RETURN


PROCEDURE entrenar
* Inicializa pesos en forma random
DO inicializar
mporcmal = 0
SELECT datos
COUNT TO mcantidad
mfin = .f.
miter = 1
mok = 0
mcantmal = CEILING(mcantidad * mporcmal / 100)
* Recorre patrones de entrada hasta que la red aprenda los deseados
DO WHILE mok < mcantidad - mcantmal
   SELECT datos
   * Desordena datos de entrada
```

```
   DO mezcla
   GO TOP
   nro = 0
   mok = 0
   SCAN
      * Lee patrón de entrada
      SCATTER TO entrada
      * Actualiza pesos
      mresu = actualizar()
      IF mresu = 0
         * No fue necesario actualizar pesos
         mok = mok + 1
      ENDIF
      nro = nro + 1
   ENDSCAN
   eta = eta * 0.9
   ? "Iteracion " + str(miter), "Bien:" + str(mok), " Eta:" + str(eta,7,4)
      miter = miter + 1
ENDDO

* Graba pesos finales
DO grabpesos

RETURN

PROCEDURE mezcla
* Desordena datos de entrada
PRIVATE msemilla
SET ORDER TO
msemilla = RAND(-1)
SCAN
    REPLACE orden WITH RAND() * 1000
ENDSCAN
SET ORDER TO orden
RETURN

PROCEDURE grabpesos
* Guarda los pesos finales
SELECT pesosoja
ZAP
FOR i=1 TO dimsal
   APPEND BLANK
   FOR j=1 TO dimvec
      mj= ALLTRIM(STR(j))
      REPLACE p&mj WITH pesosoja(i,j)
   ENDFOR
ENDFOR
RETURN


PROCEDURE inicializar
* Genera pesos iniciales en forma random
PRIVATE i,j, msemilla
msemilla = RAND(-1)
FOR i=1 TO dimsal
    FOR j=1 TO dimvec
       pesosoja(i,j) = RAND()
    ENDFOR
ENDFOR
RETURN
```

```
FUNCTION actualizar
PRIVATE k, upd, mactu, i, j, msuma, limregla
* Calcula salidas de unidades de la red
DO actu_vec
upd = 0
* Actualiza los pesos
FOR i=1 TO dimsal
   * Aplica Regla de Sanger o de Oja
   limregla = IIF(pregla=1, i, dimsal)
   FOR j=1 TO dimvec
      msuma = 0
      FOR k=1 TO limregla
         msuma = msuma + (V(k) * pesosoja(k,j))
      ENDFOR
      mactu = eta * v(i) * ( entrada(j) - msuma )
      pesosoja(i,j) = pesosoja(i,j) + mactu
      IF round(mactu,4) # 0
         * Guarda si fue necesario actualizar los pesos
         upd = upd + 1
      ENDIF
   ENDFOR
ENDFOR
RETURN upd

PROCEDURE actu_vec
* Calcula salidas de unidades de la red
v = 0
FOR i=1 TO dimsal
   FOR j=1 TO dimvec
      V(i) = V(i) + (entrada(j) * pesosoja(i,j))
   ENDFOR
ENDFOR
RETURN

PROCEDURE testear
* Recorre datos de testeo y guarda el resultado
PRIVATE i, mi
USE testt alias test IN 0
DO leepesos
SELECT test
SCAN
   SCATTER TO entrada
   DO actu_vec
   FOR i=1 TO dimsal
       mi = ALLTRIM(STR(i))
       REPLACE vec&mi WITH v(i)
   ENDFOR
ENDSCAN
RETURN

PROCEDURE leepesos
* Lee pesos guardados en entrenamiento
PRIVATE i,j
SELECT pesosoja
GO TOP
i = 1
SCAN
   FOR j=1 TO dimvec
      mj= ALLTRIM(STR(j))
      pesosoja(i,j) = p&mj
```

```
   ENDFOR
   i=i+1
ENDSCAN
RETURN
```

C.2 TABLAS UTILIZADAS EN APRENDIZAJE HEBBIANO

Tabla **DATOS**
Contiene los patrones de entrenamiento.
Su estructura es la siguiente:

	Nombre	Tipo	Long.	Dec.	Descripción
1	V1	Numeric	5	2	Sexo
2	V2	Numeric	5	2	Edad
3	V3	Numeric	5	2	Volumen de consultas
4	V4	Numeric	5	2	Antig. última consulta
5	V5	Numeric	5	2	Problemas 1
6	V6	Numeric	5	2	Problemas 2
7	V7	Numeric	5	2	Problemas 3
8	V8	Numeric	5	2	Problemas 4
9	V9	Numeric	5	2	Problemas 5
10	V10	Numeric	5	2	Relacionados
11	RESU	Numeric	5	2	Resultado real
12	ORDEN	Numeric	6		Orden de proceso en época actual

Tabla **PESOSOJA**
Contiene los pesos que resultan de la corrida
Su estructura es la siguiente:

	Nombre	Tipo	Long.	Dec.
1	P1	Float	20	15
2	P2	Float	20	15
3	P3	Float	20	15
4	P4	Float	20	15
5	P5	Float	20	15
6	P6	Float	20	15
7	P7	Float	20	15
8	P8	Float	20	15
9	P9	Float	20	15
10	P10	Float	20	15

La cantidad de variables de peso es la que corresponde a la cantidad de atributos de entrada (10) y la cantidad de registros de la tabla va a ser la de la cantidad de salidas.

Tabla **TEST**
Contiene los patrones de de testeo.
Su estructura es la siguiente:

	Nombre	Tipo	Long.	Dec.	Descripción
1	V1	Numeric	5	2	Sexo
2	V2	Numeric	5	2	Edad
3	V3	Numeric	5	2	Volumen de consultas
4	V4	Numeric	5	2	Antig. última consulta
5	V5	Numeric	5	2	Problemas 1
6	V6	Numeric	5	2	Problemas 2
7	V7	Numeric	5	2	Problemas 3
8	V8	Numeric	5	2	Problemas 4
9	V9	Numeric	5	2	Problemas 5
10	V10	Numeric	5	2	Relacionados
11	RESU	Numeric	5	2	Resultado real
12	ORDEN	Numeric	6		Orden de proceso en época actual
13	VEC1	Numeric	16	10	Valor de la componente principal 1
14	VEC2	Numeric	16	10	Valor de la componente principal 2
15	VEC3	Numeric	16	10	Valor de la componente principal 3
16	VEC4	Numeric	16	10	Valor de la componente principal 4

APENDICE D

MATRICES RESULTANTES DE LA CLASIFICACION CON EL ALGORITMO DE KOHONEN

Se presentan los gráficos de las matrices resultantes de la corrida del algoritmo de Kohonen sobre matrices de diversas dimensiones.

C significa que el cluster se clasificó como confiable.
N significa que el cluster se clasificó como no confiable.

5x5:

```
 01234
0NNNNN
1CCCNN
2CCCNC
3CC NN
4CCCCN
```

7x7:

```
 0123456
0C NNNNC
1CCNNNNN
2CCCCNCC
3C CCCNN
4C  NNNN
5CC  NNN
6CC NNNN
```

8x8:

```
 01234567
0NNNNCCCN
1NNNNCCCN
2NNNCNCNN
3NNCNNNNN
4NCCNNNNN
5CCNNNCCC
6NNNNNCCC
7CCNNNCCC
```

9x9:

```
 012345678
0CCCCCCCCC
1CCNCCCCCC
2NNCNNCCCC
3NNNN CCCC
4NNNNNN CC
5N NNNN CN
6NN NNN NN
7N NNNNCCN
8NNNNNNCNN
```

10x10:

```
 0123456789
0CCCCCNNNNN
1CCCC  N NN
2CNCCCNNNNN
3CCCNNNN CC
4CCNNNCNNC
5NNNN NNCCC
6NNNNNNNNCC
7NCNNNNNNNN
8NNNNNNNNNN
9NCCNNNNNNN
```

11x11:

```
  01234567890
 0CCCCCCCCCCC
 1CCCCCCCC CC
 2NCCCC   CNC
 3NNNN  NNCCC
 4N N  NNNCCC
 5NNNNNCCNNNC
 6NNNNNCCNNNN
 7NNNNNNCNNNN
 8NNNNNCNC  N
 9N NC C N CN
10NNNCCCCNNNN
```

12x12:

```
  012345678901
 0CCNCCC CC NN
 1CNNN  N CC
 2CNNNC NNNCC
 3CNNNNN NNNNC
 4CNNNNNNNNNCC
 5NNNNNNN CCCC
 6NNNNNNN CCCC
 7NNNNNNC CNCC
 8 NNNNNCC  CC
 9N NNNNNNCCCC
10NNNN NNNCCCC
11NNNNNNNNCCCC
```

13x13:

```
  0123456789012
 0NNNNNNNNNNNN
 1NNNNN  NNNNNN
 2CNNN NNNNN NN
 3   NNNNNNNNNN
 4NCNNNNN NNNNN
 5CCCNNNNNNN NN
 6CCNNN NNN CCN
 7CCNNNCCCCCCNN
 8CCCC  C CCCNN
 9C  CC C     N
10CCN CC   NNNN
11CCC    CC CNN
12CCCCNNCNCCCCN
```

15x15:

```
  012345678901234
 0CCC NNNNNCCCCCC
 1CCCNNNNNCCC CCC
 2NCNNNNNCCCCC CC
 3N CNNNNCCCCC CC
 4CCCCNNCNCCN CCC
 5C CCC CCCC CCCC
 6CCC NNCC   CNCC
 7NNCCC NNN CNCNN
 8NNNNNNNNN NN CC
 9N NNNN N  NNC N
10NNNNNN NN NNCCN
11NNNNNNNN NNNCCN
12NNNN NNN N N NN
13NNNNNNNNN C NNN
14NNNN NNN CCNNNN
```

16x16:

```
  0123456789012345
 0NNNNNCCNCCCCCCCC
 1NN  N NNCNCCCCCN
 2CCNCCCC CNNNCCCC
 3NNNNC CNC NNC  C
 4NNNNC CCCC N NNC
 5NNNCCC  CCCCCNNN
 6NNNCCC  CCCCC CC
 7NNNNCCCCCNC  NNCC
 8NNNNNNNNNN NNNNC
 9NNNNNNNNNN NNNCC
10N   NNN NNN NNCN
11NNCNN  NNNN NCCC
12N CNN NNNN  NCCC
13NN NNNNNN   CCCC
14NNNNNN N NN  CCC
15NNNNNNNNNNNCCCCC
```

18x18:

```
  012345678901234567
 0NNNNNNNNNNNNNNNNN
 1NNNNNNNNNNN NNNNNN
 2NNNNNNNNN NNNNNNNN
 3   NNNNN NN NNN  N
 4NNNNNNNN NN N  NN
 5NNNNN     NNNN  NNN
 6NNNNNNN       CNNCC
 7NNNNCNNCC  CCCCCC
 8NNNNNN C N CCCCC
 9NNNNNNCCCCCCC   CC
10NCCNNNCC CCN     NN
11NCCN NNNCCC  CC NN
12NCCNN NN     CCCN N
13CCCN NC      CC N N
14CCCCCC  CCNNCCN N
15CCN  CCCC CCC C  N
16CC CCCCCCCCC CCNNN
17C CCCCCCCCCCCCCNNN
```

20x20:

```
  01234567890123456789
 0CCCNNNNNNNNNNNNNC NN
 1NNNNNNNNNN NNNNNC NN
 2 CCC NN NN NNNNNC NN
 3CCC NNNNN   N  N N N
 4C  CCNNNNNN NNNNN  C
 5 C C NNNNNNNNNNN
 6C C CCCNNNNN  N NCCC
 7CCCCCNCNNN N  NNCCCC
 8NCCC  CNNNN  NNN CCC
 9NNNN C NCNNN N NN CC
10NNNN NNN NNNNNNNNNCN
11CNNNNNNNNN N N  NNNN
12CNNNNNNNN N NNNN C N
13C NNNNNNNNN NN  C  N
14CCCCCCCCCNNNN N NCC C
15CCCCCCCCCNNNNNNNNCCC
16NCCCCC C NNNN NNC  C
17CCCCCCCCCCN N NCNNNN
18CCCC CCCC  NCCCCCNNN
19CCCCCCCCCCCCCCCCCCNNN
```