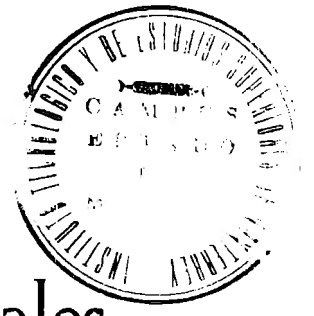
INSTITUTO TECNOLOGICO Y DE ESTUDIOS SUPERIORES DE MONTERREY

CAMPUS ESTADO DE MEXICO

Entrenamiento de la Madaline mediante el algoritmo de Pastoreo para la corrección automática de ruido en la transmisión de texto



Trabajo de investigación que, para obtener el grado de

Maestro en Ciencias Computacionales

Presentó el Ing. Jesús Benjamín Rodríguez García

Siendo integrado el jurado por:

Dr. Sergio Chapa Vergara	Presidente
Dr. Jesús Figueroa Nazuno	Secretario
Dr. Oscar Chavoya Aceves	Sinodal 1
Dr. Jesús L. Soto Sumuano	Sinodal y Asesor

Diciembre 1992

INSTITUTO TECNOLÓGICO Y DE ESTUDIOS SUPERIORES DE MONTERREY

CAMPUS ESTADO DE MÉXICO

Entrenamiento de la Madaline mediante el algoritmo de Pastoreo para la corrección automática de ruido en la transmisión de texto

Trabajo de investigación que, para obtener el grado de

MAESTRO EN CIENCIAS COMPUTACIONALES

presentó el **Ing. Jesús Benjamín Rodríguez García**

Siendo integrado el jurado por:

Dr. Sergio Chapa Vergara	*Presidente*
Dr. Jesús Figueroa Nazuno	*Secretario*
Dr. Oscar Chavoya Aceves	*Sinodal 1*
Dr. Jesús L. Soto Sumuano	*Sinodal y asesor*

DICIEMBRE 1992

Prefacio

Una de las más grandes maravillas de la naturaleza es, sin duda alguna, el poder del cerebro humano. La capacidad de adaptarse y responder a diferentes situaciones y estímulos; el poder reconocer objetos, figuras y colores que se encuentran mezclados en una sola imagen; la facilidad de aprender por experiencia; y la posibilidad de distinguir entre clases diferentes de objetos similares, son algunas de las cualidades del cerebro que aún no han podido sustituir satisfactoriamente las computadoras más modernas de la actualidad capaces de realizar hasta 100 millones de operaciones de punto flotante por segundo.

La justificación a lo anterior podemos obtenerla partiendo del hecho de que la arquitectura del cerebro humano es significantemente diferente a la arquitectura de una computadora convencional. Esto se debe a que comúnmente las computadoras realizan las operaciones de manera secuencial, mientras que en el cerebro humano existe una gran interconectividad entre las neuronas que lo constituyen; permitiendo así que las operaciones se realicen de manera paralela.

Los sistemas y métodos inspirados en la naturaleza constituyen un enfoque diferente para la Inteligencia Artificial. Intentan abstraer los principios básicos de los procesos naturales y aplicarlos a la solución de problemas. Entre estos métodos podemos listar a los algoritmos genéticos, vida artifical y redes neuronales.

Todos los sistemas y procedimientos inspirados en la naturaleza tienen en común una característica fundamental que los distingue de otros tipos de enfoques: la aplicación repetida y paralela de unos pocos principios básicos aparentemente sencillos. Sin embargo, a partir de estos principios simples emergen complejas propiedades de conjunto.

Esta tesis es un esfuerzo que intenta sumarse al de otras personas que también han incursionado en una tecnología recientemente resucitada en los últimos años: los sistemas neuronales artificiales, más comúnmente conocidos como **redes neuronales**. Por ello, a lo largo de este trabajo se presenta un análisis de la teoría general sobre redes neuronales, se desarrolla un nuevo algoritmo (que es el objetivo principal de la tesis) para el entrenamiento de un tipo particular de red y se implementa la misma como una forma alternativa de corrección automática de errores en la transmisión de texto.

Índice

Lista de figuras.

Lista de Tablas.

Introducción

Si la única herramienta que nosotros tenemos es una computadora secuencial, entonces naturalmente intentaremos resolver todos los problemas en términos de algoritmos secuenciales. Sin embargo, existen algunos problemas que son difíciles de resolver de esta manera.

Consideremos la técnica que emplearíamos si fueramos a programar una computadora convencional para reconocer objetos en una fotografía. Lo primero que nuestro programa intentaría hacer es localizar las áreas de interés en la figura. Es decir, intentaríamos dividir los puntos en grupos, de tal forma que cada grupo pudiera ser asociado únicamente con un objeto. Para ello se tendría que revisar la imagen punto por punto para determinar la formación de bordes. Una vez detectado un borde, es probable que los puntos siguientes pertenezcan a otro objeto o a una deformación del objeto que se está analizando actualmente, para determinar en cual de los dos casos nos encontramos es necesario mantener información adicional sobre las características de cada uno de los posibles objetos a encontrar. Algunas de las características a considerar son: el color, la textura y el tamaño. Debido a esto último es necesario utilizar una cantidad considerable de recursos y tiempo para el reconocimiento de los objetos, y también se reduce la generalidad del algoritmo.

Algo muy interesante sucede cuando observamos una fotografía en la cual se presenta un objeto difícil de reconocer. Una vez que logramos

identificarlo, si volvemos a observar la fotografía podemos reconocerlo más rápidamente. Mientras que con el procedimiento secuencial anterior, es necesario recorrer todos los puntos de la fotografía cada vez que se intenta reconocer al objeto y, por lo tanto, el tiempo requerido es constante. Con lo anterior podemos pensar que el esquema de almacenamiento y recuperación de información en el cerebro humano, permite hacer ajustes en la estrucutura que registra la información, de tal manera que se requiere menos esfuerzo para reconocer objetos que fueron identificados previamente.

Un ejemplo más sencillo de reconocimiento de patrones, es la identificación de caracteres escritos a mano. Una forma de identificar cada caracter consiste en colocarlo sobre una retícula de *m* renglones y *n* columnas. Para determinar la representación del caracter en la retícula se puede utilizar un vector de mXn bits donde el bit correspondiente a un pixel (un cuadro individual de la retícula) toma un valor de 1 si el pixel se encuentra total o parcialmente obscurecido y toma un valor de 0 en caso contrario. Una vez obtenido el vector de representación del caracter, utilizando un procedimiento secuencial, es necesario buscar dentro de un conjunto de vectores previamente almacenados comparando bit por bit hasta encontrar uno que corresponda exactamente al vector de representación, el vector encontrado tiene un caracter asociado que corresponde al caracter escrito a mano. Considerando que en el mundo existen millones de personas y cada una de ellas tiene su propio estilo para trazar caracteres; entonces, dependiendo además de la resolución de la retícula, existen una gran cantidad de representaciones distintas para un caracter en particular. Sin embargo, cuando observamos un caracter

parecido al número 3; aún cuando éste se encuentre bastante distorcionado con respecto a nuestro trazado ideal de dicho número, pero se asemeja más al 3 que a cualquier otro caracter, entonces decidimos que el caracter que estamos observando es el número 3. ¿La facultad que tiene el ser humano para realizar lo anterior se debe a que tiene almacenadas en el cerebro todas representaciones posibles para cada caracter? Considerando que constantemente leemos documentos escritos a mano por personas a las cuales nunca antes habiamos visto escribir, pero podemos leer e identificar cada uno de los caracteres escritos por esas personas; entonces la manera en que nuestro cerebro realiza la identificación de caracteres escritos a mano no corresponde al procedimiento secuencial descrito anteriormente, en el cual se deben tener almacenadas con anterioridad todas las representaciones posibles de cada caracter.

Entonces, ¿cómo es posible que los humanos podamos reconocer patrones bastante complejos más rápidamente que una computadora convencional, aún y cuando dichos patrones se encuentren parcial o totalmente distorcionados? Esta cuestión es aún más interesante si consideramos que los tiempos de operación en las computadoras electrónicas actuales son alrededor de siete veces más rápidos que la operación de las células que componen nuestro sistema neurobiológico. La respuesta a esta interrogante podemos obtenerla considerando que la arquitectura del cerebro humano es significativamente diferente de la arquitectura de una computadora convencional. Esto se debe a que las computadoras convencionales realizan las operaciones de manera secuencial; mientras que en el cerebro humano existe una alta interconectividad entre las neuronas que lo constituyen, permitiendo así

que las operaciones se realicen de manera paralela y, por consiguiente, se requiera poco tiempo para el reconocimiento de un patrón.

En muchas aplicaciones del mundo real, deseamos que nuestras computadoras resuelvan problemas de reconocimiento de patrones complejos, como en el caso anterior. Dado que las computadoras convencionales no son capaces de atacar satisfactoriamente este tipo de problemas, se hace necesaria una nueva tecnología que explota las características de la fisiología del cerebro humano como la base para los nuevos modelos de procesamiento. Dicha tecnología ha sido llamada sistemas neuronales artificiales, o simplemente redes neuronales.

Las redes neuronales, al igual que los demás sistemas y métodos inspirados en la naturaleza, constituyen un enfoque diferente para la Inteligencia Artificial. Intentan abstraer los principios básicos de los procesos naturales y aplicarlos a la solución de problemas. Esto último les permite presentar una característica fundamental que los distingue de otros tipos de enfoques: la aplicación repetida y paralela de unos pocos principios básicos aparentemente sencillos. Sin embargo, a partir de estos principios simples emergen complejas propiedades de conjunto útiles para la solución de problemas de alto grado de dificultad.

Considerando que la tecnología de redes neuronales es un campo de la I.A. que ha tomado bastante importancia en los últimos años después de su reciente resurgimiento, resulta atractivo investigar y tratar de aportar conocimientos que puedan facilitar otras investigaciones en esta misma ruta.

Es por ello que el objetivo principal de esta tesis consiste en **desarrollar un nuevo algoritmo para el entrenamiento de la red neuronal Madaline**. Para comprobar el funcionamento de dicho algoritmo se entrena con él una Madaline; la cual se implementa en una aplicación que comunica dos computadoras, con el propósito de corregir el ruido simulado que se introduce al pasar un caracter de una máquina a otra.

La estructura de esta tesis se compone de 5 capítulos. En el primero se presentan los hechos más importantes relacionados con el surgimiento de las redes neuronales y los primeros intentos de análisis del cerebro como un organismo computacional. En el segundo capítulo se describen los aspectos necesarios para comprender el funcionamiento de las redes neuronales; éstos son: los conceptos fundamentales, las características principales, las topologías y los tipos de entrenamiento existentes. En el tercer capítulo se describen la estructura y el funcionamiento de la Adaline y la Madaline, así como el algoritmo MR II para el entrenamiento de la Madaline. A lo largo del cuarto capítulo se presenta el desarrollo del objetivo principal de esta tesis; es decir, se describe el algoritmo de Pastoreo para el entrenamiento de la Madaline. El quinto capítulo consiste en la documentación de una aplicación que comunica dos computadoras mediante el puerto serial; en la computadora receptora se implementa una Madaline entrenada con el algoritmo de Pastoreo, con el propósito de corregir el ruido simulado que se introduce al pasar un caracter de la máquina emisora a la receptora.

Capítulo 1

Antecedentes de redes neuronales

1.1 Inspiración en los modelos biológicos

Las redes neuronales están constituidas en base a los modelos biológicos; es decir, los investigadores generalmente estan pensando en la organización del cerebro cuando construyen configuraciones de redes y algoritmos para operar sobre ellas. El conocimento acerca de la operación global del cerebro es limitado, por lo que los diseñadores deben ir más allá del conocimiento biológico actual, buscando estructuras que puedan realizar funciones de utilidad. Es por ello que al construir redes neuronales se ignoran muchas características reales del cerebro y se toman únicamente las que son de utilidad en el ámbito computacional.

1.2 Aspectos fisiológicos del sistema nervioso

El sistema nervioso del ser humano está constituido por células llamadas neuronas, su estructura es de una alta complejidad. Aproximadamente 10^{11} neuronas participan en unas 10^{15} interconexiones formando rutas de transmisión de información que pueden medir hasta más de un metro de longitud. Esta estructura en forma de rutas permite

considerar al cerebro humano como una compleja red neuronal biológica. Cada neurona comparte muchas características con las otras células del cuerpo, pero tienen la única función de recibir, procesar, y transmitir señales electroquímicas a través de las rutas neurales que componen el sistema de comunicación del cerebro.

La Figura 1 muestra la estructura de un par de neuronas biológicas típicas. Las dendritras son extensiones del cuerpo de la célula en forma de ramificaciones, éstas sirven como medio de conexión receptor con otras neuronas. El axón es un conducto que parte desde el cuerpo de la célula y tiene ramificaciones en su parte terminal, éste sirve como medio de conexión emisor con otras neuronas.

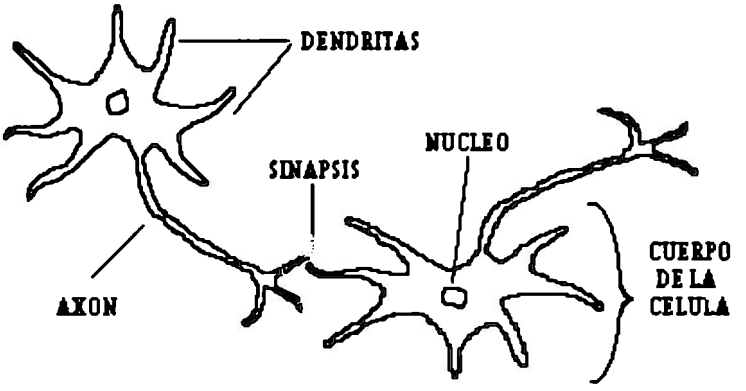


Figura 1. Conexión entre dos neuronas

La figura describe, desde un punto de vista fisiológico simplificado, la estructura de las neuronas (células nerviosas) que se encuentran en el cerebro y la manera en que están interconectadas.

Una neurona recibe señales electroquímicas a través de las dendritas, dichas entradas son conducidas hacia el cuerpo de la célula. Allí, éstas son sumadas, algunas señales tienden a excitar a la célula mientras que otras

tienden a inhibirla. Cuando la sumatoria excede a un cierto valor de umbral, la célula se dispara enviando un impulso eléctrico al exterior a través del axón. Las ramificaciones en la parte terminal del axón establecen conexiones con las dendritas de otras células. A este punto de unión se le conoce como sinapsis. De esta forma, el impulso eléctrico que viaja por el axón de una célula es distribuido a las demás células con las que mantiene contacto sináptico.

1.3 Surgimiento de los sistemas neuronales artificiales

El conocimiento fisiológico desarrollado acerca del sistema nervioso y la actividad cerebral, y la aparición de las modernas computadoras digitales en la época de los 40's, despertaron el interés de entusiastas investigadores; los que intentaron modelar la actividad cerebral considerada como un sistema neuronal biológico. Estos modelos son conocidos como sistemas neuronales artificiales. Desde entonces, se han presentado diferentes teorías sobre la modelación de los sistemas neuronales biológicos. Estas teorías, a pesar de ser diferentes, coinciden en representar dichos sistemas como un conjunto de neuronas conectadas entre sí. A este esquema de representación se le conoce con el nombre de red neuronal, cuya definición completa se presenta más adelante, primeramente analizaremos algunos de los sistemas neuronales artificiales que marcaron la pauta para el desarrollo de este campo de la Inteligencia Artificial.

1.4 El modelo de McCulloch y Pitts

EL primer intento significativo de análisis del cerebro como un organismo computacional, fue presentado en 1943 por McCulloch y Pitts. Esta teoría considera la formación de rutas aleatorias de interconexión entre las neuronas que forman la red y se basa en 5 suposiciones (ref. 10):

1.- La actividad de una neurona es un proceso de todo o nada.

2.- Durante un período de sumación latente se debe de excitar un cierto número fijo (mayor que uno) de sinapsis, para disparar la neurona en algún momento dado.

3.- El único retardo significativo dentro del sistema nervioso es el retardo sináptico.

4.- La actividad de una sinapsis inhibitoria prohibe absolutamente la excitación de la neurona en ese momento.

5.- La estructura de una red neuronal no cambia con el tiempo.

La primer asunción define a la neurona como una entidad binaria; es decir, se mantiene en un estado de encendido (1), indicando que la neurona se disparó; o de apagado (0), indicando que la neurona no se disparó. En esta teoría, la actividad de la red neuronal se desarrolla en un tiempo discreto, de tal manera que el estado (0 ó 1) de las neuronas en el tiempo t determinarán el estado de la red en el tiempo $t+1$. De esta forma podemos definir el predicado $N_i(t)$, el cual denota la afirmación de que la neurona i dispara al tiempo t. La notación $\neg N_i(t)$, denota la afirmación de que la neurona i no dispara al tiempo t. Usando esta notación podemos describir

el comportamiento de ciertas redes usando lógica proposicional. La Figura 2 muestra cinco redes sencillas de este modelo.

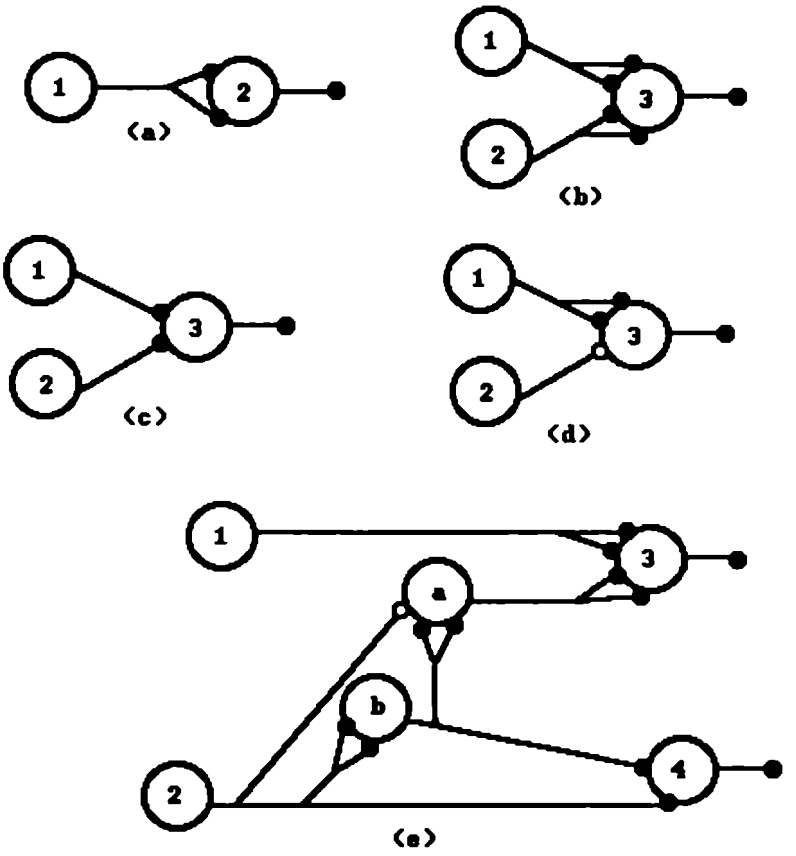


Figura 2. Ejemplos de redes sencillas del modelo de McCulloch y Pitts

Los círculos etiquetados representan los cuerpos de las neuronas. Los círculos pequeños sombreados representan conexiones excitatorias y los que no están sombreados representan conexiones inhibitorias.

Las redes 2(a) a la 2(d) definen ciertas funciones cuyas expresiones se describen en la Tabla 1. Uno de los resultados más importantes expuestos por esta teoría es que cualquier red sin retroalimentación puede ser descrita en términos de combinaciones de estas cuatro simples expresiones. Por ejemplo, la red 2(e) puede ser descrita mediante las siguientes expresiones:

$$N_3(t) = N_1(t - 1) \vee N_a(t - 1)$$

$$N_4(t) = N_2(t - 1) \& N_b(t - 1)$$

$$N_a(t) = N_b(t - 1) \& \neg N_2(t - 1)$$

$$N_b(t) = N_2(t - 1)$$

Tabla 1. Descripción de redes en base a expresiones

Se presentan las cuatro configuraciones básicas de redes en el modelo de McCulloch y Pitts

Red	Función	Descripción en base a expresiones
2(a)	Precesión	$N_2(t) = N_1(t-1)$
2(b)	Disyunción	$N_3(t) = N_1(t-1) \vee N_2(t-1)$
2(c)	Conjunción	$N_3(t) = N_1(t-1) \& N_2(t-1)$
2(d)	Negación conjunta	$N_3(t) = N_1(t-1) \& \neg N_2(t-1)$

El inconveniente fundamental de este modelo es, precisamente, la quinta suposición; la cual contrasta con la característica básica de los sistemas neuronales. Dicha característica consiste en el hecho de que éstos no nacen completamente programados, sino que de alguna manera desarrollan nuevas habilidades. Aunque la teoría de McCulloch y Pitts fue desechada, no deja de ser un modelo aproximado de la actividad cerebral, y su importancia no puede ser ignorada; ya que ésta ha sido una gran influencia para muchas personas que han incursionado en las ciencias computacionales modernas.

1.5 Aprendizaje Hebbiano

Los sistemas neuronales biológicos no nacieron preprogramados con todo el conocimiento y habilidades con que cuentan; más bien, esto es el resultado de un fenómeno de aprendizaje que modifica, de alguna forma, dichos sistemas neuronales para incorporar nueva información. El aprendizaje se logra mediante la práctica repetida de la habilidad o conocimiento que se desea adquirir. A este proceso de repetición se le conoce como entrenamiento.

Una teoría básica sobre el aprendizaje fue propuesta en 1949 por Donald Hebb en su libro Organization of Behavior .

La idea principal consiste en asumir lo siguiente: *cuando el axón de la célula A está los suficientemente cerca para excitar a la célula B y repetidamente o persistentemente contribuye para que ésta se dispare, algún proceso de crecimiento o cambios metabólicos tienen lugar en una o ambas células tal que la eficiencia de A, como una de las células que provocan el disparo de B, es incrementada* (ref. 3).

Dado que la conexión entre dos neuronas es a través de la sinapsis, esto nos invita a pensar que cualquier cambio que ocurre durante el aprendizaje tiene lugar precisamente en la sinapsis. Hebb propone que el área de conexión sináptica se incrementa. Algunas teorías más modernas establecen que hay un aumento en la velocidad de liberación de las sustancias químicas estimulantes por parte de la célula que envía el impulso eléctrico. En cualquier caso, el proceso de aprendizaje afecta las sinapsis

entre las neuronas. Esto último es la base para muchos de los algoritmos de entrenamiento de redes neuronales, incluyendo el algoritmo que se presentará más adelante como objetivo principal de esta tesis.

1.6 El perceptrón de Rosenblatt

En la década de los 50´s, Frank Rosenblatt presenta otro intento de modelación de la actividad cerebral. Su trabajo consiste de un dispositivo conocido como el perceptrón, o mejor dicho, el fotoperceptrón (ref. 13). Rosenblatt creía que la conectividad que desarrollan las redes biológicas tiene un alto grado de aleatoriedad. Por ello consideró que la herramienta más apropiada para el análisis de la organización cerebral sería la teoría de la probabilidad. Él desarrolló su teoría de la **separabilidad estadística** que usó para caracterizar las propiedades básicas de las redes aleatoriamente interconectadas de su modelo.

EL fotoperceptrón es un dispositivo que responde a patrones ópticos. Su estructura se muestra en la Figura 3. La luz hace contacto con los puntos del área sensorial (S). Cada punto de S transforma la intensisdad analógica de la luz que se refleja en un valor discreto de 0 ó 1. Los impulsos generados por los puntos de S son transmitidos a las unidades del área de asociación (A). Cada unidad de A está conectada a un conjunto aletorio de puntos de S, llamado conjunto fuente de esa unidad de A, y las conexiones pueden ser excitatorias o inhibitorias. Las conexiones pueden tomar los valores de +1, -1, y 0. Cuando se presenta un patrón en el área sensorial, una unidad de A puede activarse si la suma de las entradas excede cierto

valor de umbral. Al activarse una unidad de A, ésta produce una salida que es enviada a la siguiente capa de unidades.

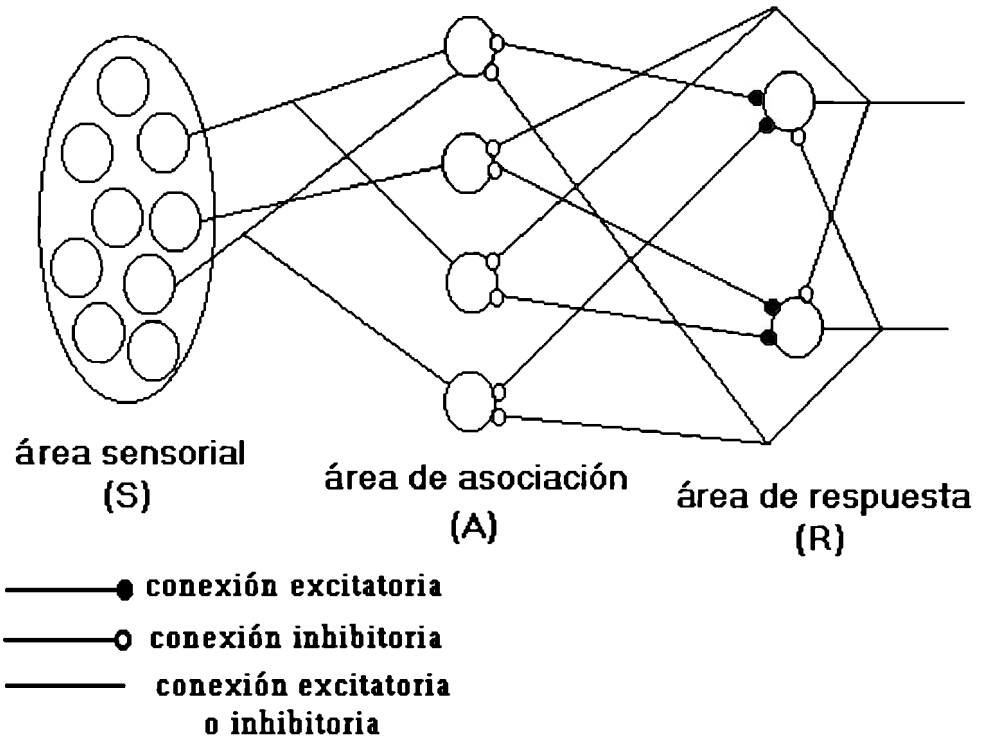


Figura 3. Estructura del fotoperceptrón

Se pueden apreciar las rutas aleatorias de conexión que se forman entre las capas.

Las unidades de A están conectadas a las unidades de la capa de respuesta (R). Las rutas de interconexión entre ambas capas también están distribuidas de manera aleatoria; pero además, existen conexiones inhibitorias de retroalimentación de la capa de respuesta a la capa de asociación, y conexiones inhibitorias entre las unidades de R. El esquema completo de interconectividad, para un perceptrón sencillo de dos unidades en R, es presentado en forma de diagrama de Venn en la Figura 4.

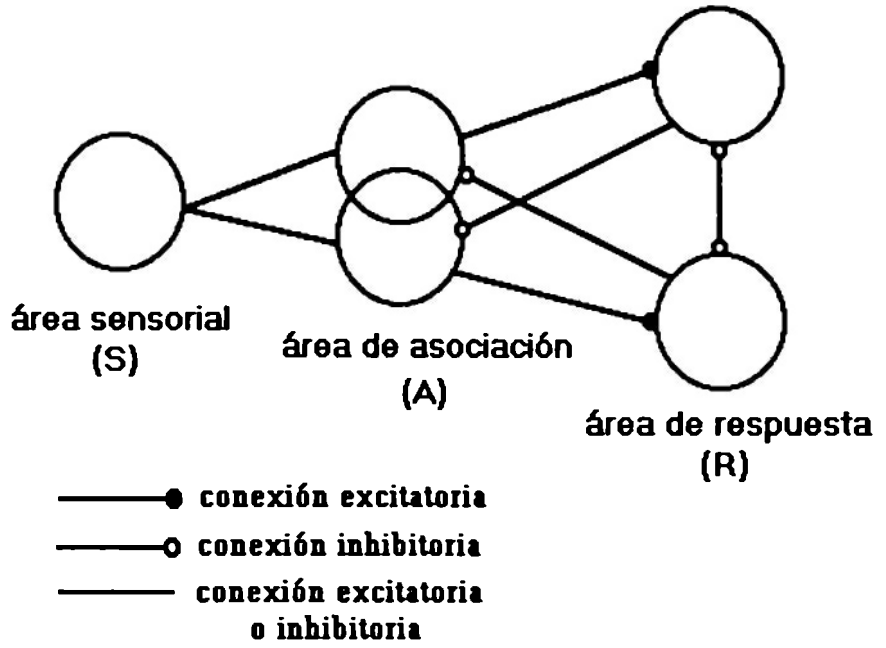


Figura 4. Representación del fotoperceptrón mediante diagramas de Venn

Nótese como cada unidad de R inhibe al resto de ellas y al complemento de su conjunto fuente. Se puede apreciar, además, que algunas unidades de A están en el conjunto fuente de diferentes unidades de R.

En la Figura 4, puede apreciarse que cada unidad de R inhibe las unidades de A que forman el complemento del conjunto fuente de dicha unidad de R. Además, cada unidad de R inhibe al resto de ellas. Estos factores contribuyen al establecimiento de una única unidad ganadora de R para cada patrón que se presenta en el área sensorial. Las unidades de R responden con un +1 si la suma de sus entradas excede un cierto valor de umbral, y con un -1 en caso contrario.

Este modelo de redes es capaz de clasificar los patrones presentados al área sensorial dentro de categorías, de acuerdo a las unidades de respuesta del sistema. Los patrones que son suficientemente parecidos

deberán excitar la misma unidad de R; mientras que patrones diferentes deberán excitar una unidad distinta en R.

El perceptrón fue considerado como un dispositivo de aprendizaje. Ya que en su configuración inicial, fue incapaz de distinguir ciertos patrones de interés; pero a través de un cierto proceso de entrenamiento, éste pudo aprender a distinguirlos. En esencia, el entrenamiento consiste en un proceso de reforzamiento mediante el cual la salida de las unidades de A es incrementada o decrementada dependiendo de si o no las unidades de A contribuyen a la respuesta correcta del perceptrón para un patrón dado. Cuando un patrón es aplicado al área sensorial y el estímulo se propaga a través de las capas hasta que una unidad de respuesta es activada, si dicha unidad es la correcta, la salida de las unidades contribuyentes de A es incrementada. Si la unidad activada de R no es la correcta, la salida de las unidades contribuyentes de A es decrementada. Una variante de este esquema consiste en que al mismo tiempo que se incrementa el valor de la salida de las unidades activas; restarle el valor totalizado de dicho incremento, de manera equitativa, a los valores de salida de las unidades inactivas. El análisis matemático de la inferencia en el perceptrón es bastante completo, pero no se presenta aquí porque queda fuera de los objetivos del presente trabajo.

Rosenblatt presenta un teorema muy importante en 1962 acerca del aprendizaje del perceptrón. La prueba de dicho teorema demostraba que un perceptrón podía aprender cualquier cosa que pudiera representar. Aquí es importante distinguir entre representación y aprendizaje. La representación consiste en la habilidad del perceptrón (o cualquier otra estructura) para

simular una función específica. El aprendizaje requiere la existencia de un procedimiento sistemático para ajustar la estructura de tal forma que simule la función.

1.7 El perceptrón simplificado

El entusiasmo de los investigadores en el campo de las redes neuronales disminuyó bastante con la aparición, en 1969, del libro Perceptrons: An Introduction to Computational Geometry, escrito por Marvin Minsky y Seymour Papert (ref. 11). Ellos presentaron un astuto y detallado análisis del perceptrón en términos de sus capacidades y limitaciones. Para dicho análisis, Minsky y Papert hicieron a un lado el tratamiento probabilístico presentado por Rosenblatt, y retomaron las ideas del cálculo de predicados e idealizarón el modelo del perceptrón que aparece en la Figura 5. El conjunto $P = \{p_1, p_2, .., p_n\}$ es el conjunto de predicados. Cada predicado simple p_i toma un valor de uno si el *i*-ésimo punto de la retina está encendido, y toma un valor de cero en caso contrario. Cada uno de los predicados de entrada es ponderado por un número del conjunto $\{a_1, a_2, ..,a_n\}$ asociado a él. La salida (S) toma un valor de uno si y sólo si la sumatoria de los productos a_ip_i, para i igual a 1 hasta n, es mayor que cierto valor de umbral (U).

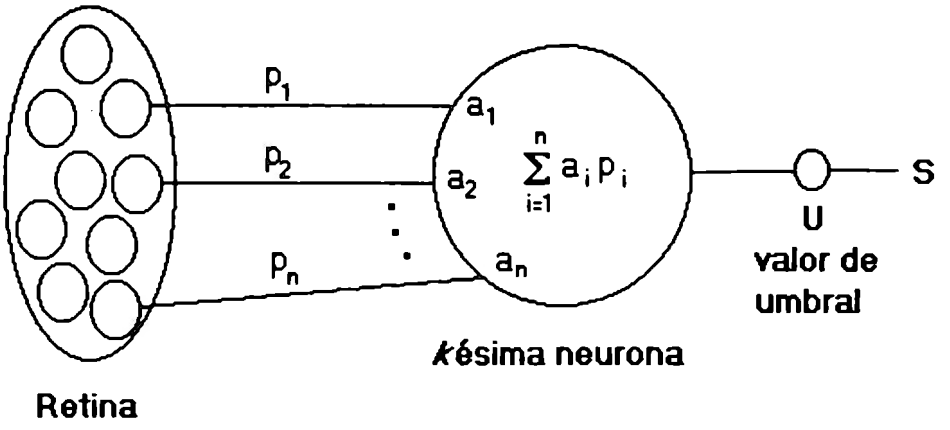
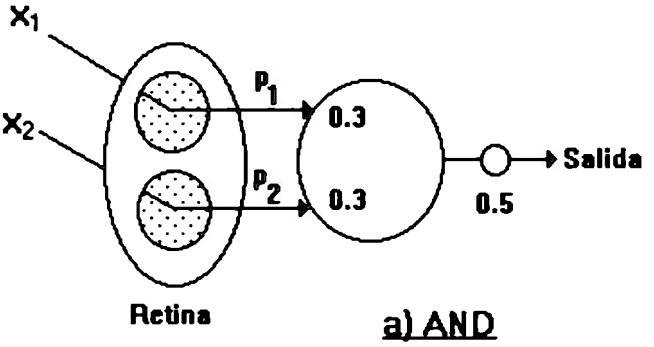


Figura 5. Representación del perceptrón simplificado

La salida está en función de la sumatoria y del valor de umbral.

Bajo el modelo del perceptrón presentado en la Figura 5, es posible simular algunas funciones como el AND y el OR, que se muestran en la Figura 6(a) y 6(b) respectivamente. Sin embargo, otra función igualmente sencilla como el XOR no se puede representar bajo esta estructura, tal y como se aprecia en la Figura 6(c).



a) AND

X_1	X_2	Salida
0	0	0
0	1	0
1	0	0
1	1	1

Tabla de verdad

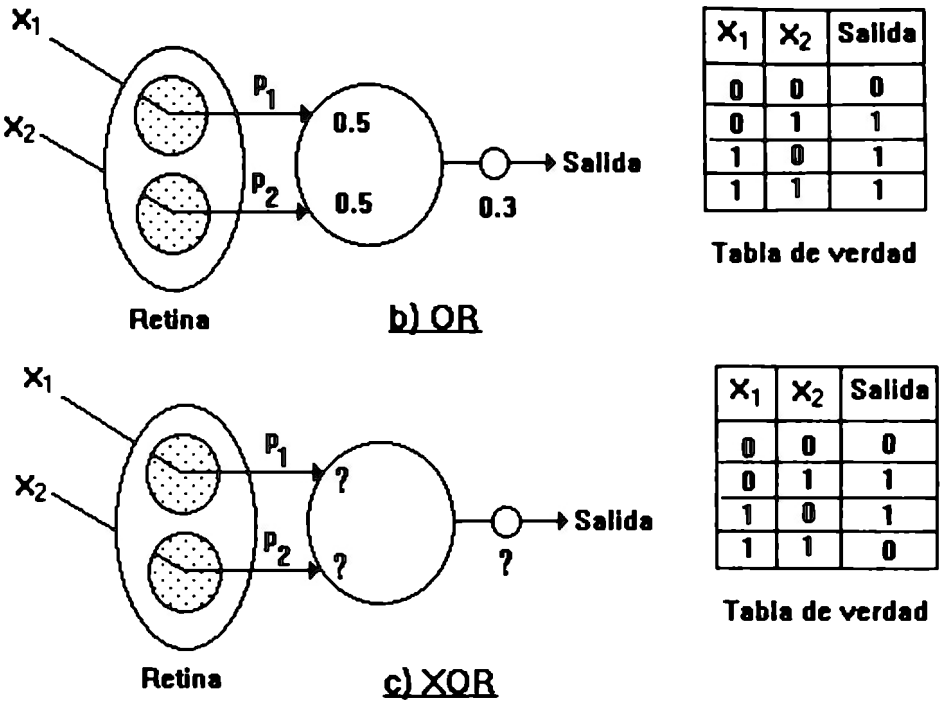
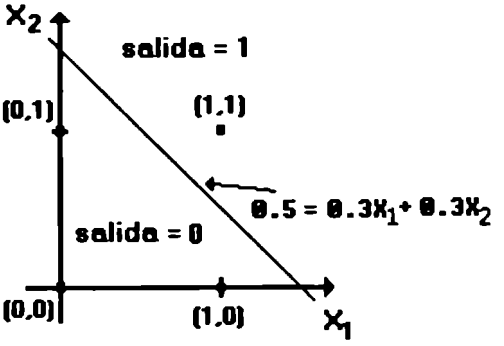


Figura 6. Representación de funciones mediante el perceptrón

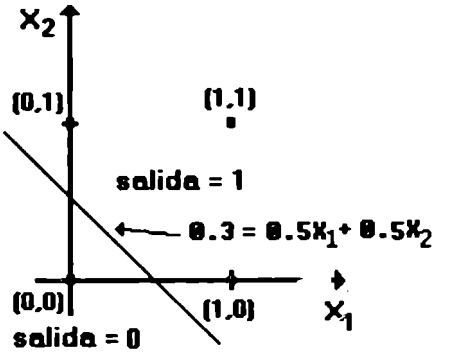
Se presentan los valores de ponderación y de umbral necesarios para que al aplicar las entradas X_1 y X_2 se obtenga la salida deseada tal y como se muestra en las tablas de verdad.

Precisamente, uno de los puntos discordantes más importantes presentados en el libro de Minsky y Papert es la idea de que existen ciertas restricciones en la clase de problemas que el perceptrón puede tratar. Se llegó a la conclusión de que el perceptrón puede clasificar patrones sólo si son **linealmente separables**. Dado que existen muchos problemas de clasificación que no poseen clases linealmente separables, entonces existen restricciones en la aplicación del perceptrón. El problema del XOR es uno de estos casos. El concepto de separabilidad lineal puede apreciarse en la Figura 7. En las figuas 7(a) y 7(b), correspondientes a la función AND y OR respectivamente, se puede observar que existe una línea que divide al plano X_1-X_2 en dos regiones. Una en la cual todos los valores de X_1 y de X_2 que

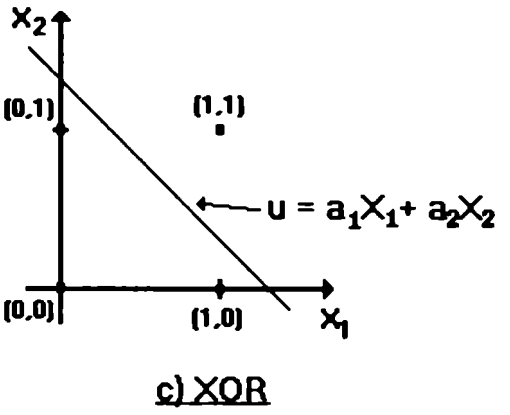
pertenezcan a ella, arrojarán un valor de uno a la salida del perceptrón. Y otra en la que, cualquier valor de X_1 y de X_2 que pertenezcan a ella, arrojarán un valor de cero a la salida del perceptrón. Entonces, mediante esa línea que separa las dos regiones es posible clasificar a todos los patrones que se puedan aplicar a la entrada del perceptrón. Y de aquí el nombre de separabilidad lineal. En el caso del XOR presentado en la Figura 7(c), podemos apreciar que no existe forma de colocar la línea que agrupa los posibles valores de entrada, de tal manera que se puedan definir una región de salida igual a uno y otra de salida igual a cero que cumplan con la tabla de verdad de la función.



a) AND



b) OR

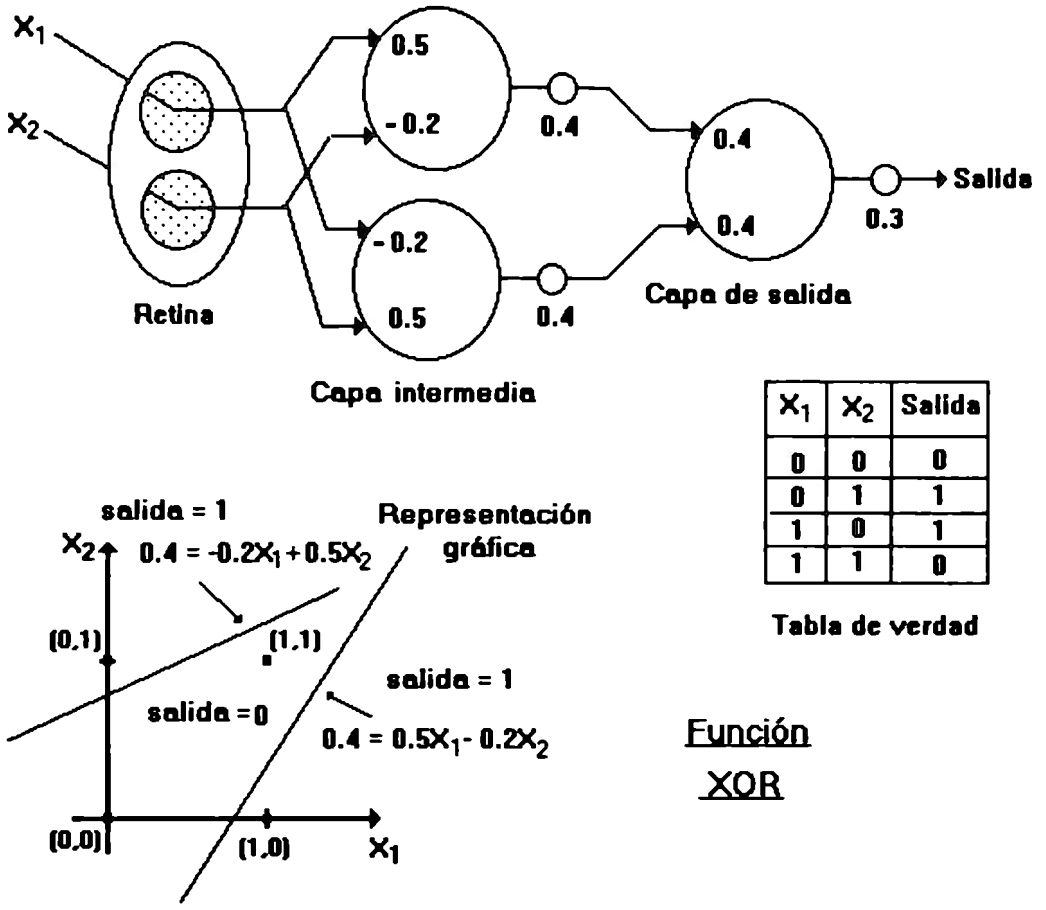


c) XOR

Figura 7. Representación de la separabilidad lineal de patrones

Se puede apreciar la línea fronteriza que permite formar las regiones que cumplen con la tabla de verdad de la función AND y OR, pero en el caso del XOR no existe forma de encontrar dicha frontera.

Pudiera pensarse que el problema del XOR es irresoluble mediante el perceptrón. Sin embargo, recordando lo que dice el teorema de Rosenblatt, "el perceptrón puede aprender todo aquello que puede representar", podemos observar que X_1 XOR X_2 = $(X_1 \& \neg X_2) \vee (\neg X_1 \& X_2)$. Dado que tanto $(X_1 \& \neg X_2)$ como $(\neg X_1 \& X_2)$ son posibles de representar con el perceptrón, entonces basta agregar una capa más al perceptrón para que realice un OR con las salidas de las representaciones de los dos términos anteriores. Esto se puede apreciar en la Figura 8.



X_1	X_2	Salida
0	0	0
0	1	1
1	0	1
1	1	0

Tabla de verdad

Figura 8. Solución al problema de XOR mediante el perceptrón

Para poder representar la función del XOR mediante el perceptrón, es necesario agregar una capa intermedia a su estructura.

Las dos neuronas que componen la capa intermedia permiten la definición de dos rectas sobre el plano X_1-X_2 de tal forma que se generan tres regiones, dos de las cuales arrojarían un valor de uno a la salida del perceptrón para caulquier punto que esté dentro de ellas, y en la otra se obtendría un valor de cero a la salida para cualquier punto situado en ella. Mediante los conceptos de hiperespacio e hiperplanos, podemos observar que, en general, en un espacio de n dimensiones se puden definir hiperplanos de n-1 dimensiones de tal forma que se generen regiones que

puedan agrupar patrones que pertenezcan a una misma clase. En el ejemplo del XOR, tenemos un espacio X_1-X_2 de dos dimensiones y podemos definir dos hiperplanos de una dimensión, las rectas $-0.2X_1+0.5X_2=0.4$ y $0.5X_1-0.2X_2=0.4$, que generan las tres regiones que agrupan a los posibles patrones de entrada {(0,0), (0,1), (1,0), (1,1)} en las regiones adecuadas para poder producir la salida (0 ó 1) indicada, en la función XOR, para cada uno de ellos.

Aunque el problema del XOR pudo ser resuelto mediante la adición de una capa intermedia, existen otras consideraciones presentadas por Minsky y Papert que aún no han sido resueltas satisfactoriamente. Sin embargo, el perceptrón ha servido de inspiración para muchos de los modelos neuronales, existentes en la actualidad, que han resuelto problemas de importancia considerable.

Capítulo 2

Fundamentos de redes neuronales

2.1 La neurona artificial

El funcionamiento de la neurona del modelo del perceptrón simplificado ha sido aprovechado, bajo ciertas modificaciones, por los constructores de muchos modelos de redes neuronales que existen en la actualidad. Por ello se presenta la estructura general de una neurona artificial; a partir de la cual pueden considerarse al resto de las representaciones de neuronas como casos particulares de dicha estructura general.

La neurona artificial, también llamada unidad de procesamiento, fue diseñada con el propósito de representar las características básicas de una neurona biológica. En esencia, un conjunto de entradas son aplicadas (X_1, X_2,..., X_n), cada una de las cuales representa la salida de otras neuronas. Cada entrada es multiplicada por un peso correspondiente (W_1, W_2,..., W_n), en analogía a la estrechez sináptica entre ambas neuronas, y la suma de dichas multiplicaciones, llamada entrada neta (net), determina si la neurona receptora se dispara o no. La Figura 9 muestra un modelo que implementa esta idea.

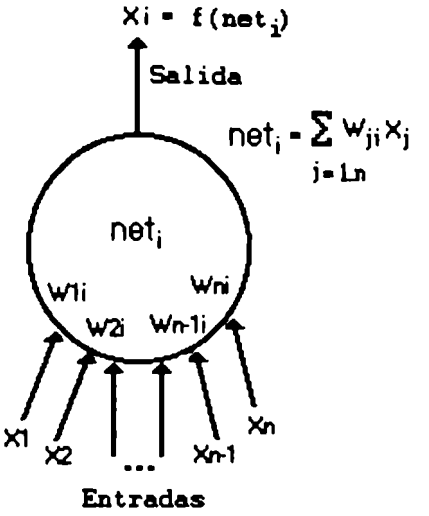


Figura 9. Neurona artificial

La estructura representa una neurona artificial (unidad de procesamiento) básica.

La salida de una unidad de procesamiento está en función del valor de net; es decir, $X_i = f(net_i)$. De tal forma que para determinar la salida X_i de la neurona i tenemos que calcular net_i en base a la sumatoria de los productos $W_{ji} X_j$ para j de 1 hasta n. EL valor de net también puede obtenerse a partir del producto punto de los vectores $\mathbf{W}=(W_1,W_2,...,W_n)$ y $\mathbf{X}=(X_1,X_2,...,X_n)$; es decir, net = **WX** .

Formalmente, una neurona es un elemento simple de procesamiento que mapea Q->R. Donde R es el conjunto de los números reales y Q es el conjunto de conjuntos *n*-arios de números reales. Al mismo tiempo, f(net) mapea R->R. A f(net) se le denomina **función de activación.**

Existen diferentes tipos de funciones de activación. Las más comunes son la función lineal, escalón-binaria, escalón-bipolar, sigmoidal y la tangente hiperbólica. Estas funciones han sido utilzadas por los diseñadores

de las diferentes topologías de redes neuronales en sus algoritmos de entranamiento. La Tabla 2 muestra como se obtienen los valores de cada función.

Tabla 2. Descripción de las funciones de activación

Presenta la manera en que se evaluan las difentes funciones de activación

Tipo de Función	Descripción
Lineal	$f(x) = x$
Escalón - binaria	$f(x,\theta) = \begin{cases} 1 & \text{si } x \geq \theta \\ 0 & \text{si } x < \theta \end{cases}$
Escalón - bipolar	$f(x,\theta) = \begin{cases} 1 & \text{si } x \geq \theta \\ -1 & \text{si } x < \theta \end{cases}$
Sigmoidal	$f(x) = \frac{1}{1+e^{-x}}$
Tangente hiperbólica	$f(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}$

A continuación se presentan los nombres de algunas redes en cuyas neuronas se utilizan las funciones de activación descritas en la Tabla 2. Cabe mencionar que estas funciones se utilizan en otras redes que no se presentan aquí.

La función lineal y la función sigmoidal se pueden utilizar alternativamente en las salidas de las neuronas de la red de backpropagation (BPN), formalizada primeramente por Werbos (ref. 15), posteriormente

por Parker (ref. 12) y después por Rummelhart y McClelland (ref. 7) . La función escalón binario se utiliza en el perceptrón y en la red de la teoría de resonancia adaptativa (ART) presentada por Carpenter y Grossberg (ref. 1). La función escalón bipolar se utiliza en la Adaline propuesta por Widrow y Hoff (ref. 17), en la red de memoria asociativa bidireccional (BAM, ref. 3) y la red Madaline que se presentará más adelante. La función tangente hiperbólica se utiliza en una versión de la red de Hopfield (ref. 9).

2.2 Definición general de una red neuronal

Hasta el momento, se ha analizado el funcionamiento completo de una neurona y se han mostrado algunos ejemplos de redes neuronales; sin embargo, no se ha presentado la definición precisa de una red neuronal.

Las redes neuronales son simuladores del comportamiento del cerebro humano a nivel de sus elementos más sencillos, las neuronas. Estos simuladores, generalmente, se implementan en hardware y, por ello, también son conocidas como neurocomputadoras. Una red es un conjunto de neuronas altamente interconectadas, donde cada neurona se conecta a muchas más, pero la ruta formada entre la neurona x y la neurona y es diferente a la ruta entre la neurona x y la neurona z. Las líneas de conexión entre las neuronas pueden ser unidireccionales o bidireccionales y su función consiste en transmitir la señal de salida de una de ellas a una entrada de la otra. Generalmente, existe un valor de ponderación para cada una de las señales que entran a una neurona. Cada una de las entradas de la neurona i (una de las muchas que forman la red) sirven para recibir la

salida de alguna neurona específica *j*, entonces el valor de ponderación W_{ji} se fija con el propósito de medir la facildad para excitar o inhibir que tiene la neurona *j* sobre la neurona *i*, por lo cual a W_{ji} también se le llama **peso de conexión**. La manera en que están distribuidas las neuronas y las rutas de conexión entre ellas, definen la topología de la red y, por consecuencia, el comportamiento de la misma. Cualquier red contiene una capa de entrada formada por *n* neuronas, mediante la cual se aplican los patrones a la red; y una capa de salida constituida por *m* neuronas, a través de la cual se pueden tomar los patrones producidos por la red. Donde *m* no necesariamente debe ser distinta de *n*, pero ambas deben ser mayor o igual que uno. Entre la capa de entrada y la capa de salida puede haber una o más capas ocultas. En ocasiones, la capa de entrada y la capa de salida son una misma. La operación básica de una red consiste en que al aplicarse un patrón formado por *n* bits (en este concepto un bit puede tomar tres valores: -1, 0 ó 1) en la capa de entrada, dichos bits son propagados a través de las neuronas, mediante las trayectorias formadas entre ellas, hasta obtener un patrón de *m* bits, en la capa de salida, correspondiente al patrón aplicado en la capa de entrada.

Formalmente, una red neuronal es una estructura de procesamiento de patrones que mapea E->S. Donde E es el conjunto de todos los patrones de *n* bits (n >= 1) que se puden aplicar a la entrada de la red y S es el conjunto de todos los patrones de *m* bits (m >= 1) que se pueden obtener a la salida de la misma. El conjunto E recibe el nombre de conjunto de entrenamiento y el conjunto S recibe el nombre de conjunto de salidas deseadas.

2.3 Características de las redes neuronales

Las redes neuronales, por su propia naturaleza, poseen ciertas características esenciales que las hacen atractivas para ciertas aplicaciones. A continuación se describen dichas características:

2.3.1 Representación distribuida. No se puede señalar a ciertas neuronas como las que almacenan un determinado dato, sino que el conocimiento se encuentra distribuido a través de las conexiones que existen entre todas la neuronas que constituyen la red. Lo cual difiere de la mayoría de los sistemas de almacenamiento y recuperación de información, en los cuales los datos se encuentran en alguna determinada localidad.

2.3.2 Alto grado de paralelismo. Debido a su propia estructura, es posible que la información presentada en capa de entrada se distribuya al mismo tiempo a todas las neuronas de la red. Logrando, con ello, pequeños tiempos de respuesta y, por consiguiente, una alta velocidad de procesamiento de patrones. Por ejemplo, para una red que contenga miles de neuronas agrupadas en tres capas secuenciales (entrada, oculta y salida) tendrá un tiempo de respuesta de tres veces el tiempo de procesamiento de una sola neurona.

2.3.3 Generalización. Es capaz de tratar con patrones incompletos o distorsionados y asociar a ellos un patrón de salida que correspondería a patrones similares previamente aprendidos por la red.

2.3.4 Auto-actualización. Cuando se presenta un patrón que no ha sido aprendido previamente, la red puede ser modificada para agrupar este nuevo patrón junto con el resto de los patrones que ya habían sido aprendidos y que pertenecen a la misma clase, sin necesidad de reprogramación de la aplicación.

2.4 Topologías de redes neuronales

El comportamiento de una red se rige, básicamente, por su topología; es decir, por la manera en que están distribuidas las neuronas que la componen, y las rutas de conexión que se forman entre ellas.

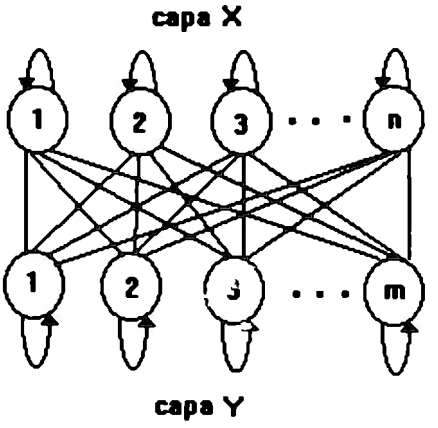
En la actualidad, existen muchas estructuras de redes neuronales que han sido presentadas por diversos investigadores con el propósito de resolver, de la manera más adecuada, problemas específicos de ciertas áreas. Sin embargo, todas estas redes pueden agruparse en dos clasificaciones generales que son: la red ampliamente interconectada y la red multicapa. Cada una de estas estructuras generales posee sus propias características que las hacen atractivas para resolver cierto tipo de problemas.

2.4.1 Red ampliamente interconectada. En esta topología, todas las neuronas están conectadas, cada una, hacia el resto de ellas. Aquí las rutas de conexión pueden ser unidireccionales o bidireccionales, por lo cual se deduce que puede existir retroalimentación entre las neuronas. Los patrones son aplicados solamente a un conjunto de las neuronas y el patrón

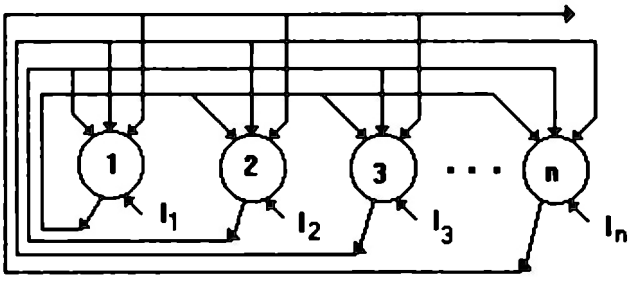
de salida se obtiene en otro conjunto, no necesariamente disjunto o diferente. Este tipo de estructura es el que presentan la mayoría de las memorias asociativas como la BAM, la memoria de Hopfield y la máquina de Boltzmann cuyas gráficas de presentan el la Figura 10(a), 10(b) y 10(c) respectivamente (ref. 3).

Este tipo de redes son excelentes clasificadores de patrones distorsionados o incompletos; sin embargo, dado que el número de neuronas de la red está en proporción con el tamaño de los patrones, se presentan problemas de saturación cuando la cantidad de patrones diferentes a clasificar es muy grande.

a) Bidirectional Associative Memory (BAM)



b) Memoria de Hopfield



c) Máquina de Boltzmann

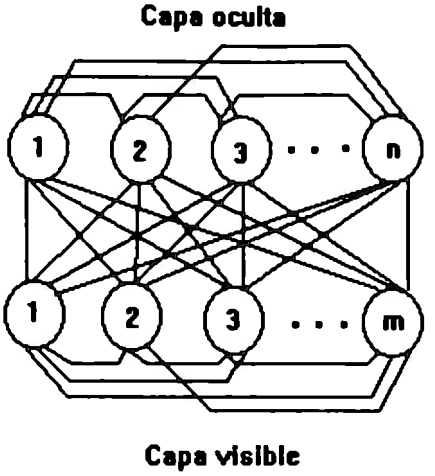


Figura 10. Ejemplos de redes ampliamente interconectadas

Las redes ampliamente interconectadas, generalmente, funcionan como memorias asociativas

2.4.2 Red multicapa. La estructura de esta red consiste de una capa de entrada, una o más capas ocultas y una capa de salida. Cada capa, a su vez, está constituida por una o más neuronas. Generalmente, las unidades de la capa de entrada solo tienen la función de distribuir los bits de los patrones de entrada hacia la primer capa oculta. Las capas no necesariamente son del mismo tamaño; es decir, pueden tener direfentes cantidades de neuronas. Los conjuntos de neuronas que forman cada capa son disjuntos. Las unidades de una capa transmiten su salida a las unidades de la capa del nivel inmediato superior y, en algunos casos, al inferior; pero nunca existe conexión directa entre neuronas de capas no adyacentes. El sentido de las rutas de conexión puede ser unidireccional o bidireccional. Existe, además, la posibilidad de interconexión entre las neuronas de una misma capa. Un ejemplo típico de esta topología es el perceptrón de Rosenbaltt.

Dentro de esta misma categoría existe una derivación muy importante. Dicha derivación es la red neuronal **"feed forward"**. Esta red tiene la particularidad de que las unidades de una capa únicamente pueden transmitir su señal hacia las unidades de la capa inmediata superior, no existe transmisión hacia la capa inmediata inferior. Esto significa que el sentido de las conexiones es unidireccional hacia adelante.

Por la propia estructura de esta red; los patrones presentados en la capa de entrada, son propagados hacia adelante a través de las salidas de las unidades de esta capa. Las señales de salida son recibidas por las entradas de cada una de las unidades de la capa oculta, las cuales a su vez, las propagarán hacia las entradas de las unidades de la siguiente capa oculta o a la capa de salida. Durante este proceso de alimentación hacia adelante (feed forward), el patrón presentado en la entrada se va deformando, tanto en su tamaño como en sus bits, hasta que en la salida se obtiene un patrón que depende completamente del patrón en la entrada, del número de neuronas en cada capa y de los valores de ponderación de las conexiones entre las neuronas. Como ejemplo de esta estrucutura podemos citar a la backpropagation network (BPN), formalizada primeramente por Werbos (ref. 15), posteriormente por Parker (ref. 12) y después por Rummelhart y McClelland (ref. 7), cuya estructura aparece en la Figura 11. Otra red que cae dentro de esta clasificación es la red Madaline, cuya descripción se presentará más adelante como parte fundamental de este trabajo.

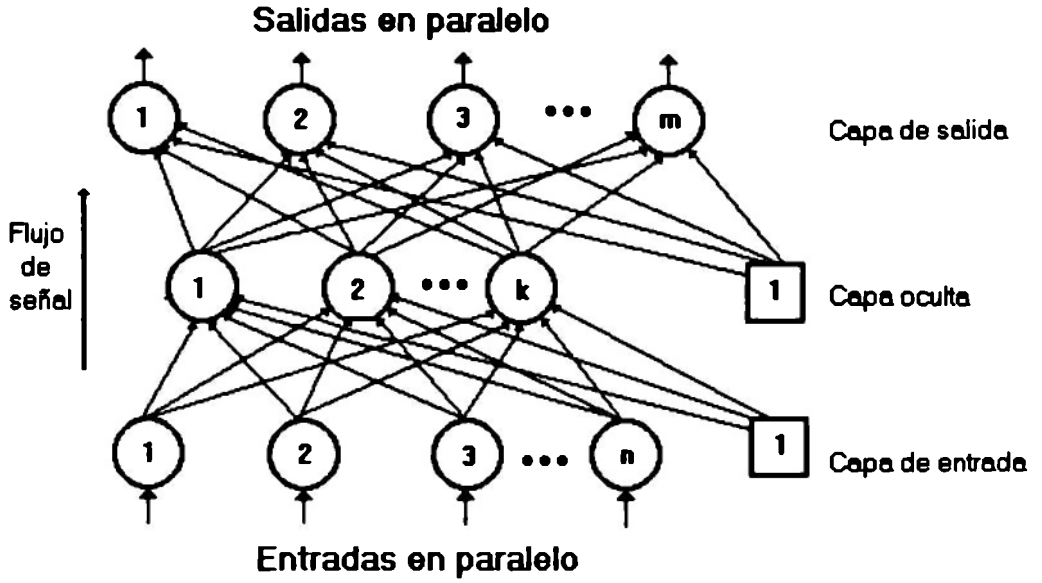


Figura 11. La backpropagation network (BPN)

Los patrones presentados en la capa de entrada son propadagados hacia adelante y, en la capa de salida, se obtiene el patrón correspondiente.

2.5 Tipos de entrenamiento

La estructura general de cualquier tipo de red neuronal, le da un gran poder de reconocimiento de patrones a ésta, siempre y cuando sea sometida a un proceso de entrenamiento que le permita realizar inferencia sobre ella misma al momento de presentarle patrones en su entrada. Los algoritmos de entrenamiento pueden clasificarse como supervisados y no-supervisados.

2.5.1 Entrenamiento supervisado. En estos algoritmos, se requiere que la red se comporte como un sistema de entrada/salida. En otras palabras, la red recibe un vector **X** y arroja un vector **Y**. Para lograr el entrenamiento se deben considerar las parejas de todos los patrones del

conjunto de entrenamiento con su respectivo patrón considerado como la salida deseada. La operación, básicamente, consiste en aplicar un patrón en la entrada de la red y propagarlo a través de la misma hasta obtener un patrón de salida. Dicho patrón de salida, se compara con la salida deseada para el patrón aplicado y, mientras exista diferencia, se debe ajustar la red hasta considerar (dado que en algunos casos se maneja cierto margen de error) que se ha obtenido la salida deseada. El ajuste de la red, generalmente, se hace mediante la modificación de los pesos de las conexiones entre las neuronas. El entrenamiento concluye cuando la red entrega los patrones que se consideran como las salidas deseadas para cada uno de los patrones del conjunto de entrenamiento.

En el caso de las redes multicapa, además de modificar los pesos de conexión; en algunas ocasiones, es necesario agregar una capa oculta más, o simplemente aumentar la cantidad de neuronas en las ya existentes. También, bajo ciertas circunstancias, se pueden crear nuevas conexiones especiales entre neuronas, tal es el caso de los contactos axo-axónicos para lograr los fenómenos de *inhibición* y *facilitación* presináptica.

2.5.2 <u>Entrenamiento no-supervisado</u>. El principal inconveniente que presenta el entrenamiento supervisado, es su desviación de los sistemas biológicos, dado que es difícil de creer que exista un mecanismo en el cerebro que compare las salidas obtenidas y las salidas deseadas y con ello ajustar el sistema neuronal. Aún cuando esto fuera posible, ¿de dónde provienen los patrones considerados como la salida deseada? Por esta razón, se han desarrollado algoritmos de entrenamiento no-supervisado, en

ocasiones llamados auto-organizables, que son más apegados a los modelos de aprendizaje de los sistemas biológicos. En el proceso de entrenamiento no se requieren las salidas deseadas para los patrones que se pueden introducir a la red. El algoritmo de entrenamiento modifica la red, generalmente mediante sus pesos de conexión, para producir patrones de salidas consistentes. Esto significa que si se aplica dos veces el mismo patrón a la red, en ambas ocasiones se obtiene el mismo patrón a la salida. Además, patrones lo suficientemente parecidos pueden arrojar el mismo patrón de salida. El proceso de entrenamiento, por lo tanto, extrae las características elementales del conjunto de entrenamiento y agrupa a los patrones dentro de diferentes clases. El entrenamiento termina cuando la red se estabiliza después de la realización de ciertas operaciones con el conjunto de entrenamiento. En algunos casos el entrenamiento es constante, ya que conforme se van presentando nuevos patrones a la red, ésta se altera automáticamente y reagrupa los patrones previamente introducidos considerando las características del nuevo patrón.

Este tipo de entrenamiento es característico de las redes ampliamente interconectadas; ya que las conexiones de retroalimentación entre las neuronas, hacen posible que los patrones sean propagados de manera continua, hacia adelante y hacia atrás, hasta que se logra la estabilización de la red.

Capítulo 3

Las neurocomputadoras Adaline y Madaline

3.1 La Adaline

Es un dispositivo que representa un tipo especial de neurona. Su nombre proviene de la contracción de ADAptive LInear NEuron, aunque también se dice que proviene de ADAptive LINear Element. Es importante analizar su estructura y funcionamiento, dado que es el tipo de neurona utilizado en la red neuronal Madaline que se presentará más adelante. Esta unidad de procesamiento fue presentada en 1960 (ref. 17) por Bernard Widrow y un grupo de estudiantes (el más destacado Marcian E. Hoff; quien, tiempo después, participaría en el invento del microprocesador). Su estructura y funcionamiento son similares a los del modelo de la neurona artificial típica, presentada previamente. Solo que, además, se le agrega una entrada adicional X_0, cuya señal siempre tiene un valor de uno, pero su peso de conexion W_0, puede variar durante el proceso de entrenamiento. Esta entrada adicional se conoce como término de sesgo o valor de alumbramiento y tiene la función de aumentar el poder de representación de la Adaline. Las señales de entrada pueden tomar únicamente los valores -1 y 1. La función de activación que utiliza esta neurona, es la función escalón bipolar, también conocida como función signo. La Figura 12, presenta la estructura de la Adaline. El valor de *net* se calcula, igual que en el caso de

la neurona artificial, mediante la suma de los productos de las señales de entrada por su peso asociado, pero se adiciona el valor de W_0. El valor de la salida de la Adaline toma un valor de uno si el valor de *net* es mayor o igual que cero; y toma un valor de menos uno, en caso contrario.

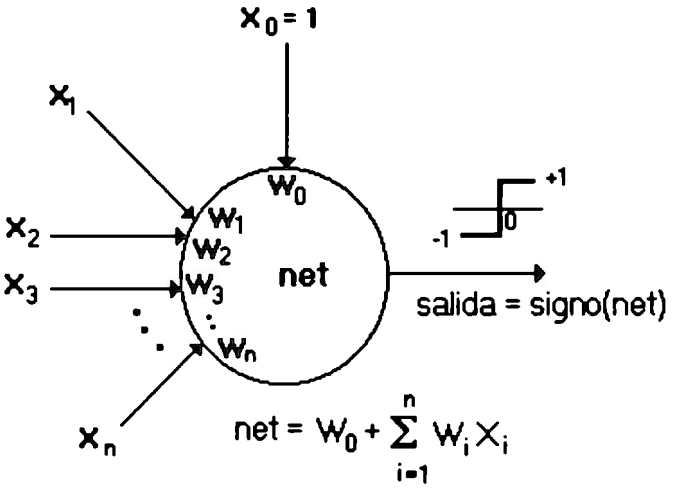


Figura 12. Estructura de la Adaline

El valor de net se obtiene como en la neurona artificial típica, pero se le agrega el valor de W_0. La función de activación que determina la salida de la neurona, es la función escalón-bipolar o signo.

3.2 El algoritmo LMS para el entrenamiento de la Adaline

La Adaline es adaptativa en el sentido de que existe un procedimiento de entrenamiento para modificar los pesos con el propósito de permitirle entregar el valor de salida correcto para una determinada entrada. Dicho algoritmo de entrenamiento es conocido como LMS (Least Mean Squared), y su funcionamiento consiste en encontrar el vector de pesos (W) que pueda satisfacer, considerando cierta tolerancia, las salidas deseadas para

cada patrón del conjunto de entrenamiento. Para lograr encontrar el vector de pesos óptimo (W^*) que produzca las salidas exactas para todo el conjunto de entrenamiento, se debe minimizar el error caudrado medio, calculado mediante la sumatoria del cuadrado de las diferencias entre las salidas deseadas (d) y las salidas netas obtenidas (net) para todos los patrones del conjunto de entrenamiento, dividida entre el número de patrones (L) que forman éste último.

$$\langle \varepsilon_k^2 \rangle = \frac{1}{L} \sum_{K=1}^{L} (d_k - net)^2$$

Describiendo a net de manera vectorial, net = $W^T X$, podemos definir al error cuadrado medio como una función de W.

$$\langle \varepsilon_k^2 \rangle = \langle (d_k - W^T X_k)^2 \rangle \quad \text{entonces} \quad \langle \varepsilon_k^2 \rangle = f(W)$$

$$\langle \varepsilon_k^2 \rangle = \langle d_k^2 \rangle + W^T \langle X_k X_k^T \rangle W - 2 \langle d_k X_k^T \rangle W$$

Es posible encontrar el valor mínimo de esta función moviendose en la dirección negativa del gradiente de la misma. Este procedimiento se conoce como el método de descenso paso a paso. Sin embargo, el procedimiento para calcular el gradiente, además de complicado, es costoso. Por lo cual, en el algoritmo de entrenamiento de la Adaline, se utiliza una aproximación del gradiente que puede ser determinada a partir de información que es conocida en cada iteración. Dicho algoritmo de entrenamiento, se presenta a continuación:

0. Asignar un valor real aleatorio, entre -1 y 1, a todos los pesos de conexión y hacer k = 1.

1. Aplicar el patrón X_k, de n bits, a las entradas de la Adaline.

2. Determinar el cuadrado del error para el patrón k, usando el valor actual del vector de pesos W.

$$\varepsilon_k^2 = (d_k - W^T X_k)^2$$

3. Considerar al cuadrado del error para el patrón k, como una aproximación del error total y determinar la dirección del gradiente mediante:

$$\delta = \text{sign}(d_k - W^T X_k)$$

4. Actualizar el vector de pesos de acuerdo a lo siguiente:

$$W = W + 2\delta\, m\, \varepsilon_k X_k$$

Donde m es el factor de magnitud de cambio (en ocasiones llamado amortiguamiento)

5. Si el error cuadrado medio no ha sido reducido hasta un valor aceptable, hacer k = k + 1 y regresar al paso 1.

La actualización del vector de pesos se hace buscando el mínimo del error total cuadrado medio. La utilización del factor de magnitud de cambio m, se fija en un valor menor que 0.1 (por ello, también se le considera como amortiguamiento), y determina la velocidad de aproximación hacia el mínimo del error y evita la posibilidad de oscilación sobre el punto óptimo. La

manera en que este algoritmo se aproxima a la solución óptima describe una curva escalonada, debido a que se utiliza una aproximación del gradiente al determinar la dirección del próximo movimiento. Por lo anterior, no se puede asegurar el alcance exacto del punto óptimo, y es por ello que se considera cierto margen de error para detener el algoritmo de entrenamiento. Esto último se muestra en la Figura 13.

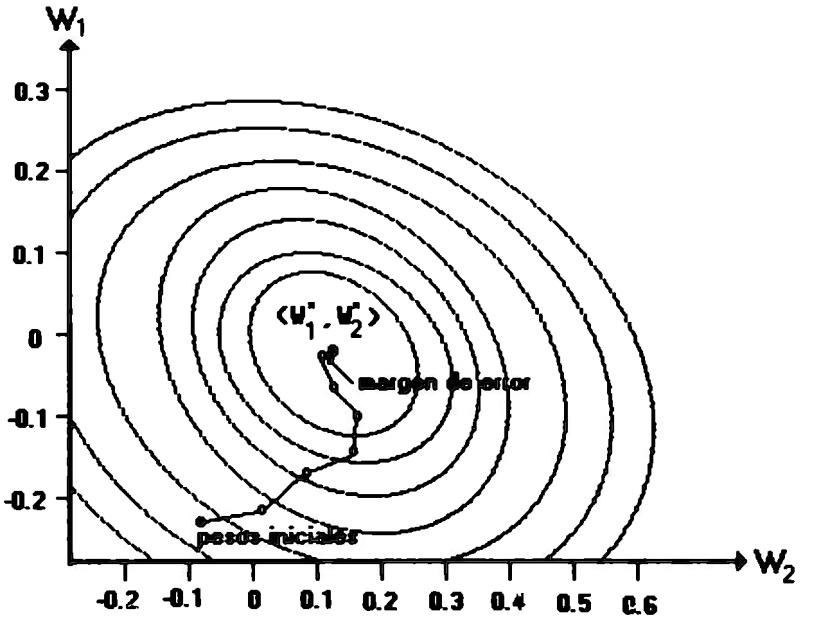


Figura 13. Búsqueda del mínimo error mediante el algoritmo LMS

Se presenta el camino hipotético descrito por los valores un vector de pesos de dos dimensiones en la búsqueda del mínimo error. Se puede apreciar que no es una curva bien definida dado que se utiliza una aproximación del gradiente.

En el apéndice B se presenta la ejecución de un software que entrena una Adaline mediante el algoritmo LMS, para tratar de representar ciertas funciones.

3.3 La Madaline

Una Adaline con n entradas y una salida, es capaz de implementar ciertas funciones lógicas. Hay 2^n posibles patrones de entrada. Cualquier implementación lógica es capaz de clasificar cada patrón con un -1 o +1, de acuerdo a la salida deseada. Por lo tanto, existen 2 elevado a la 2^n posibles funciones lógicas conectando n entradas a una sola salida. Debido a que la Adaline es una variante del perceptrón, presentado previamente, también presenta sus mismas limitaciones. Esto significa que solo puede implementar un subconjunto de todas las combinaciones de funciones posibles. Este subconjunto esta formado por todas las funciones lógicas linealmente separables. Entonces funciones que no caen en este rango, como lo es el XOR, no se puden representar en una simple Adaline. Pero, al igual que con el perceptrón, es posible combinar varias Adalines para resolver dicho problema. Esto da lugar a la creación de la red neuronal Madaline, cuyo nombre proviene de la contracción Many Adalines. La Figura 14 muestra dicha estructura.

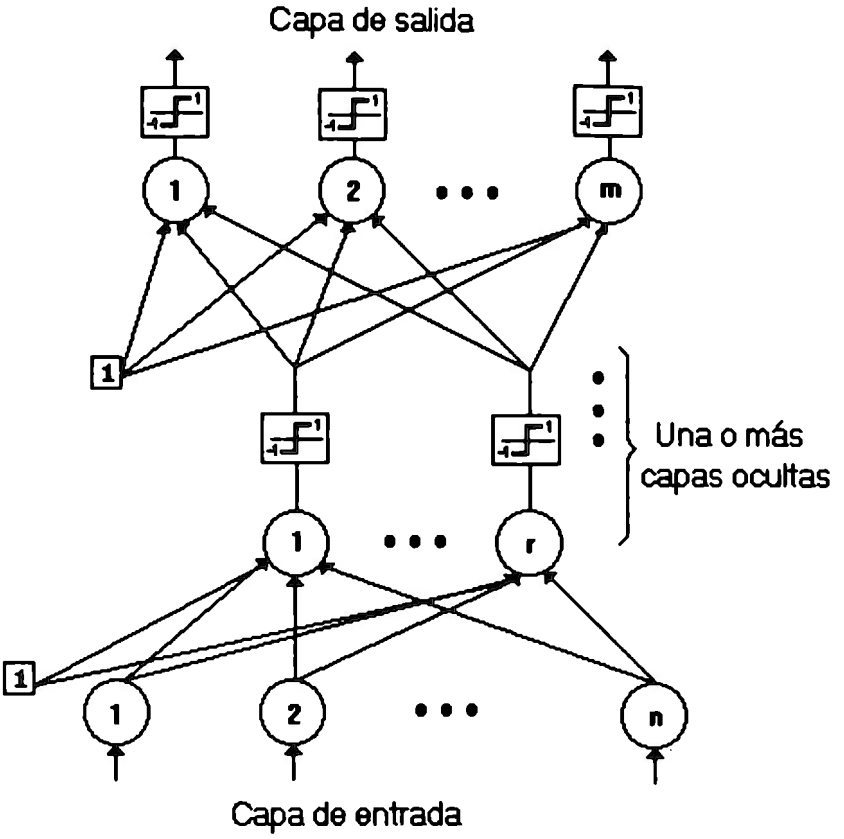


Figura 14. Arquitectura de la red neuronal Madaline

Cada unidad de procesamiento es una Adaline. La unidades de la capa de entrada únicamente tienen la función de distribuir los bits del patrón de entrada hacia la capa oculta.

Dado que cada unidad de procesamiento es una Adaline; entonces, las salidas y las entradas de cada una de ellas, pueden tomar únicamente valores de -1 o +1. La estructura de esta red consiste de una capa de entrada, una o más capas ocultas y una capa de salida. Cada capa, a su vez, está constituida por una o más Adalines. Las unidades de la capa de entrada únicamente tienen la función de distribuir los bits del patrón de entrada hacia la primer capa oculta. Las capas no necesariamente son del mismo tamaño. Los conjuntos de Adalines que forman cada capa son

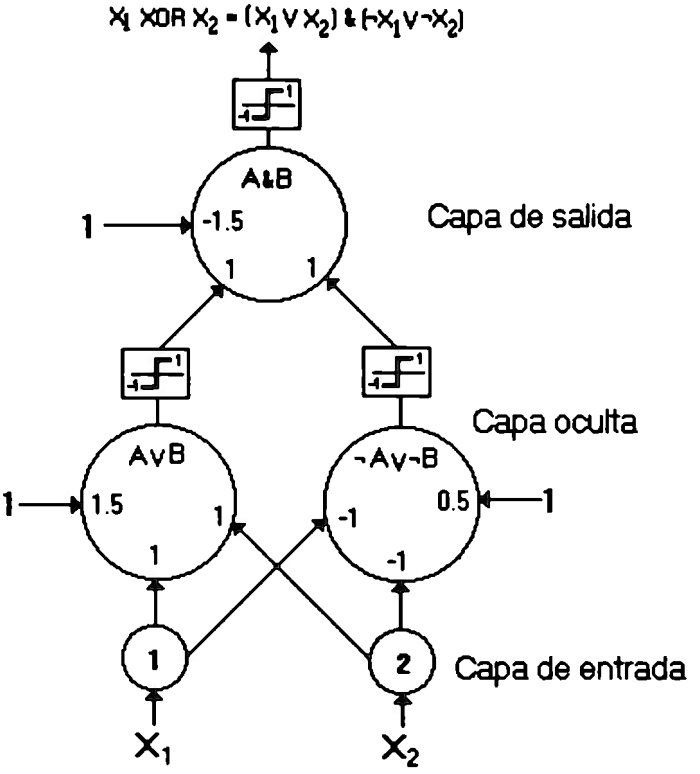


Figura 15. Madaline que resuelve el problema del XOR

Las entradas X_1 y X_2 pueden tomar valores de ±1 y representan los bits del patrón X.

3.5 La Adaline y la Madaline, consideradas las primeras neurocomputadoras comerciales

La Adaline y la Madaline fueron las primeras neurocomputadoras comerciales en el mundo (ref. 16). La Memistor Corporation, fundada y manejada por Bernard Widrow, ofrecía estas máquinas a la venta comercial entre 1962 y 1965. En estos dispositivos, los valores de los pesos de conexión eran representados mediante resistencias variables, de tres terminales, llamadas *memistor*, cuya estructura se presenta en la Figura 16. El memistor es una celda electroquímica en miniatura dentro de la cual se

coloca cobre, obtenido de una solución de sulfato de cobre ($CuSO_4$), sobre una barra de carbón en respuesta a una corriente directa que circula a través de un electrodo que se conecta a una de las terminales. Las otras dos terminales están conectadas, cada una, a los extremos de la barra de carbón y son utilizadas para que circule una corriente alterna a través de la resistencia constituida por la barra de carbón revestida de cobre. De esta forma, es posible ajustar electrónicamente los pesos de la neurocomputadora.

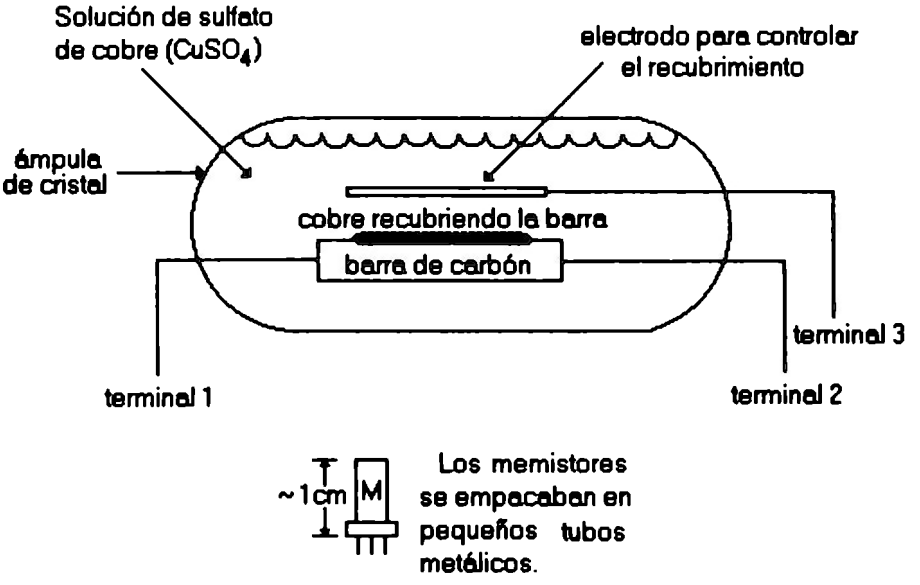


Figura 16. Estructura del memistor

Esta resistencia electroquímica variable se utilizaba en las neurocomputadoras Adaline y Madaline de Bernard Widrow. El electrodo de la terminal 3 ajustaba la cantidad de cobre sobre la barra para controlar la resistencia del memistor y con ello ajustar los pesos de la neurocomputadora.

3.6 Un esquema electrónico para el entrenamiento de la Adaline

Un esquema electrónico completo de entrenamiento para la AdaLine, constaría, además de los memistores, de un circuito de comparación entre el valor de net y la salida deseada, y un circuito adaptativo que reciba la diferencia obtenida por el circuito de comparación y ajuste los memistores de acuerdo al algoritmo de entrenamiento. Este esquema se presenta en la Figura 17.

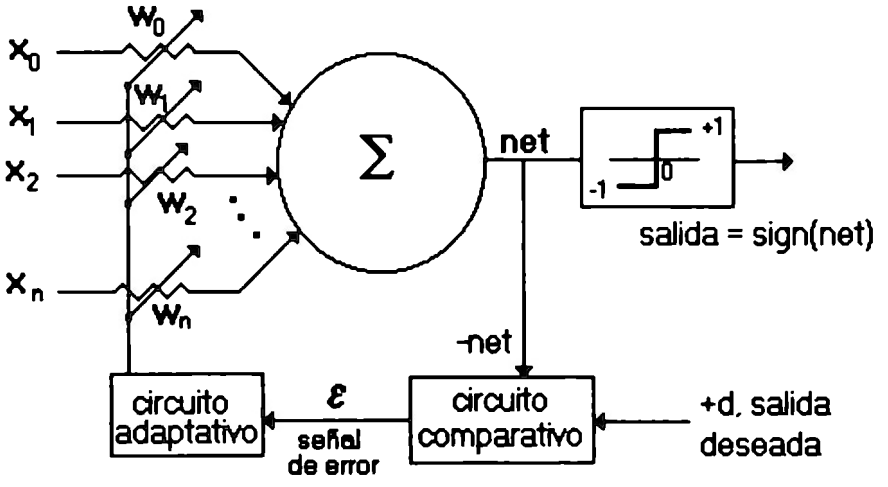


Figura 17. Esquema electrónico de entrenamiento de la Adaline

El circuito comparativo determina el error entre la salida deseada y el valor de net, el circuito adaptativo toma el error y lo utiliza para actualizar la resistencia de los memistores.

3.7 La necesidad de utilizar redes neuronales de unidades con salida digital

La función de activación de una Adaline es discontinua en cero; por lo tanto, no es diferenciable y por ello no es posible determinar, mediante el

cálculo diferencial, el efecto que tiene la variación de los parámetros de una unidad de la capa oculta sobre el error total del sistema. Pero, si la función de activación de cada Adaline se cambiara por una función continua y, por ello, diferenciable; entonces se podría aplicar una generalización del algoritmo LMS para el entrenamiento de la Madaline. Esto es, precisamente, lo que se logra al cambiar la función de activación por una que entrega una salida sigmoidal de comportamiento similar pero continuo, dando lugar a la creación de la red neuronal backpropagation y su algoritmo de entrenamiento conocido como la "regla delta generalizada" (ref. 15, 12, 7).

Existen ciertas aplicaciones, como la implementación de redes neuronales para compresión y transmisión instantánea de imágenes, en las cuales la salidas de las unidades de una capa debe viajar largas distancias al ser enviada hacia las entradas de las unidades de la siguiente capa. Aquí se debe considerar que las señales transmitidas se ven afectadas por factores como atenuación, debido a la resistencia del propio medio de transmisión, y distorsión provocada por el ruido del medio ambiente. Entonces, si se emplea una red de neuronas con salida analógica; la alteración sufrida por la señal recibida puede provocar graves problemas de inconsistencia en el sistema, dado que el valor de la señal recibida en la entrada *i* de la neurona *j* no es igual al valor de salida de la neurona *i*. En cambio, con un modelo de neuronas con salida digital, es posible emplear dispositivos de reajuste para eliminar cualquier alteración en la señal recibida, de tal manera que el valor de la señal en la entrada *i* de la neurona *j* es el mismo valor de salida de la neurona *i*.

Debido a lo anterior, resulta necesario y atractivo tratar de desarrollar algoritmos de entrenamiento para las redes cuyas neuronas trabajan con salidas digitales, como en el caso de la Madaline. Ese es, precisamente, el objetivo principal de esta tesis: presentar un algoritmo propio para el entrenamiento de la Madaline.

3.8 El algoritmo MR II para el entrenamiento de la Madaline

Dado que la arquitectura de la Madaline no presenta retroalimentación entre las Adalines que la constituyen, entonces sus algoritmos de entrenamiento deben ser del tipo supervisado. El método más comúnmente utilizado para entrenar la Madaline, es el conocido como Madaline Rule II (MR II).

El algoritmo MR II (ref. 18), sigue una política de prueba y error, pero realiza una búsqueda inteligente basada en el principio de **mínima perturbación**. Al aplicar un patrón en la capa de entrada y propagarlo, a través de las capas ocultas, hacia la capa de salida; se obtendrá un patrón constituido por las salidas bipolares de todos los Adalines de la capa de salida. Entonces, el entrenamiento intenta reducir el error considerado como el número de Adalines cuya salida difiere del bit correspondiente de la salida deseada para el patrón aplicado a la Madaline. El principio de mínima perturbación consiste en el hecho de que se deben seleccionar primero aquellos Adalines que con el menor cambio de sus pesos logren reducir el error. El algoritmo general se presenta a continuación (ref. 3):

1. Aplicar un patrón de entrenamiento a las entradas de la Madaline y propagarlo hasta las unidades de salida.

2. Contabilizar el número de bits incorrectos en la capa de salida; llamar a este número el error.

3. Para todas las unidades en la capa de salida:

3.1. Seleccionar la primera unidad, no seleccionada anteriormente, cuyo valor de net es más cercano a cero. (Esta unidad es la que puede invertir su salida bipolar con el menor cambio en sus pesos, de acuerdo al criterio de *mínima perturbación*).

3.2. Cambiar los pesos de la unidad seleccionada, tal que su salida bipolar se invierta.

3.3. Propagar el patrón de entrada hacia adelante, desde la entrada hasta la salida, nuevamente.

3.4. Si el cambio de los pesos provoca una disminución del número de errores, aceptar la modificación; en caso contrario, restaurar los pesos originales.

4. Repetir el paso 3 para cada una de las capas ocultas.

5. Si no existe error, ir al paso 8.

6. Para todas las unidades en la capa de salida:

6.1. Seleccionar la pareja $\{U_1, U_2\}$ de unidades, no seleccionada anteriormente, cuyo valor de net es más cercano a cero.

6.2. Cambiar los pesos de ambas unidadades, tal que que sus salidas bipolares se inviertan.

6.3. Propagar el patrón de entrada hacia adelante, desde la entrada hasta la salida, nuevamente.

6.4. Si el cambio de los pesos provoca una disminución del número de errores, aceptar la modificación; en caso contrario, restaurar los pesos originales.

7. Repetir el paso 6 para cada una de las capas ocultas.

8. Repetir los pasos 1 al 7 con el siguiente patrón de entrenamiento, hasta que se hallan obtenido las salidas deseadas para todo el conjunto de entrenamiento.

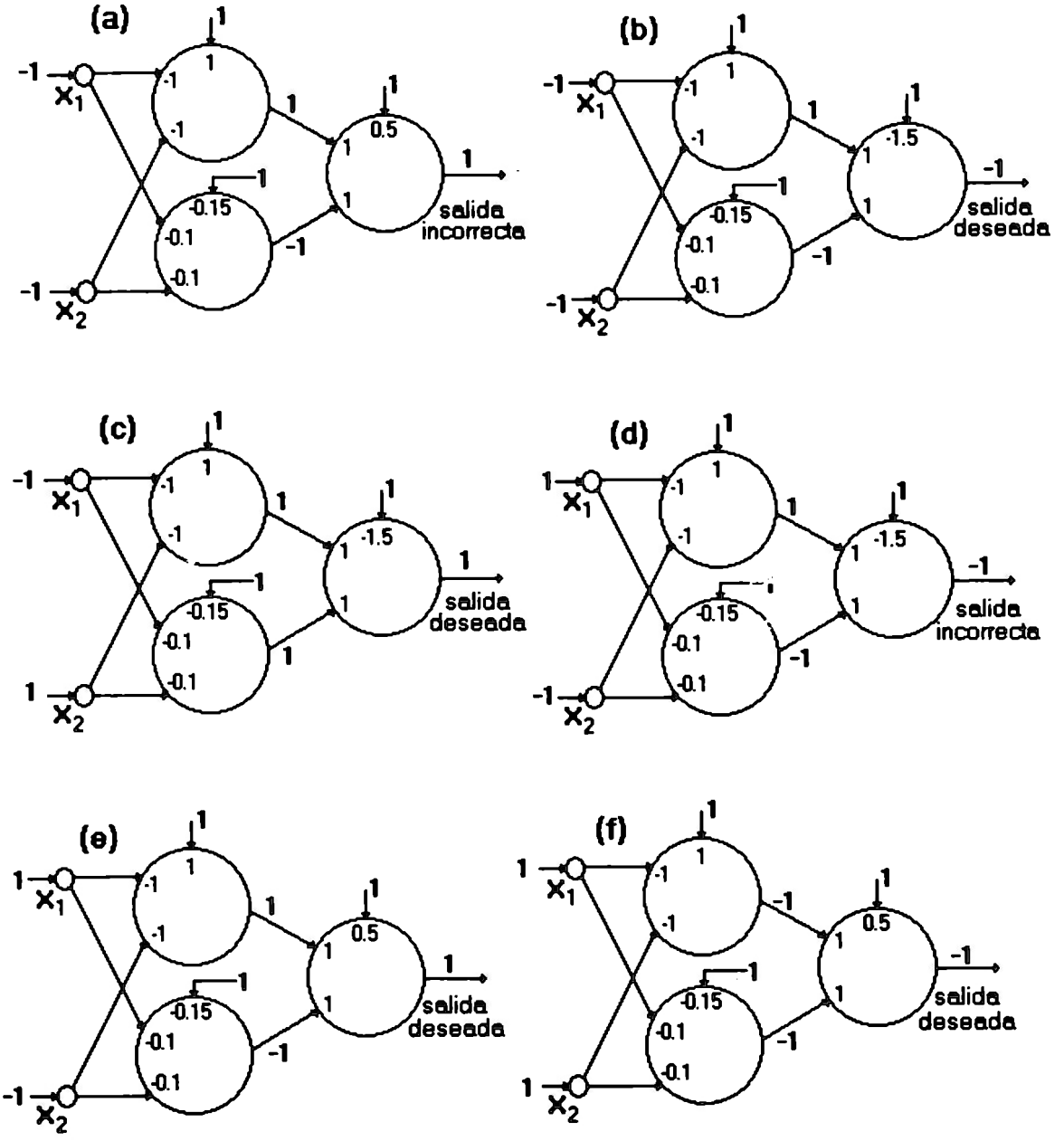
En ocasiones, el hecho de llegar a seleccionar pares de Adalines en los pasos 6 y 7 no es suficiente para lograr el entrenamiento, generalmente en Madalines con más de 25 unidades en alguna capa. Por ello, puede ser

necesario repetir dichos pasos con ternas, cuartetas e incluso combinaciones más grandes para conseguir el entrenamiento.

3.9 La necesidad de marcar las Adalines como seleccionadas previamente

Al analizar detalladamente el algoritmo MR II, se puede observar que la necesidad de marcar las Adalines como previamente seleccionadas, se debe al propio criterio de mínima perturbación que rige su funcionamiento. Al seleccionar la unidad *i* que tiene el valor de net más cercano a cero y hacer el menor cambio en sus pesos de tal forma que se invierta su salida bipolar, para reducir el número de bits erroneos en la capa de salida, sucede que su valor de net cambia de signo, pero queda nuevamente muy cercano, generalmente más que antes, a cero. Por este motivo, existe una gran probabilidad de que al tratar de ajustar la Madaline para el siguiente patrón, en una iteración sucesiva, la misma unidad *i* sea elegida nuevamente para reducir el número de errores en la salida. Lo anterior da lugar a que los pesos de la unidad *i* sean modificados en cada iteración al ajustar la red para cada uno de los patrones del conjunto de entrenamiento, dado que su valor será el más cerrado a cero. Dando lugar a la posibilidad de un ciclo infinito y, por consiguiente, el entrenamiento nunca termine a pesar de que la aplicación sea factible. Otra forma en que se puede producir un ciclo, si se permite seleccionar una Adaline previamente modificada, es cuando se tiene una sola unidad en la capa de salida. Dado que el algoritmo trabaja primeramente con la capa de salida, es posible que con ajustes a los pesos de la Adaline de salida se logre obtener la salida deseada para cada patrón del conjunto de entrenamiento y, por lo tanto, nunca sea necesario

modificar las unidades de la capa oculta; sin embargo, esto puede provocar que los pesos de la unidad de salida comiencen a oscilar y, debido a ello, el entrenamiento nunca termine. La Figura 18 presenta un caso sencillo, donde se produce un ciclo infinito al tratar de entrenar la red para simular la función XOR sin considerar la unidades como previamente seleccionadas.



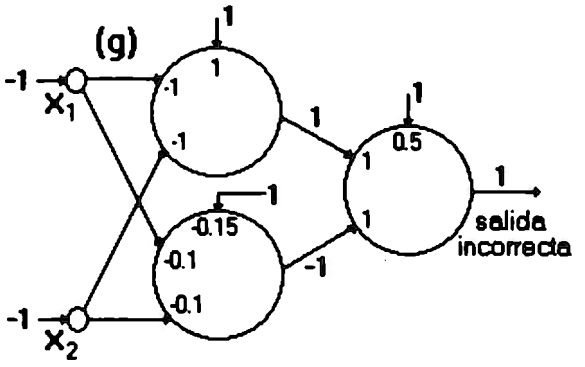


Tabla de verdad de la función XOR

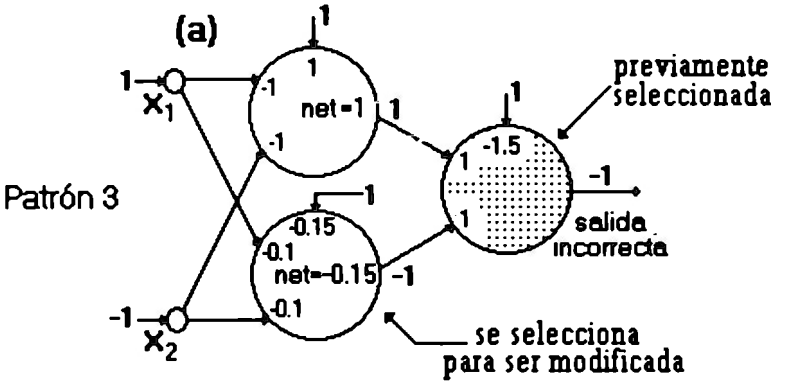
#	X_1	X_2	Salida deseada
1	-1	-1	-1
2	-1	1	1
3	1	-1	1
4	1	1	-1

Figura 18. Intento fallido de entrenamiento de la Madaline para simular la función XOR

Si no se consideran a la unidades como previamente seleccionadas, se puede presentar un ciclo al estar eligiendo repetidamente la misma Adaline para ajustar la red. En este caso siempre se elige la unidad de salida.

La Figura 18(a) muestra que el patrón 1 falla con los pesos originales, por ello, en la Figura 18(b) se presenta la red con los pesos de la unidad de salida modificados para obtener la respuesta deseada para el patrón 1. La Figura 18(c) muestra como se obtiene la salida correcta para el patrón 2 con los pesos actuales. En la Figura 18(d) se puede observar que el patrón 3 falla. Nuevamente, como se pude observar en la Figura 18(e), la unidad de salida es seleccionada y se corrigen sus pesos de tal manera que se obtine la respuesta deseada para el patrón 3. Enseguida, en la Figura 18(f), se puede observar como se obtiene la salida deseada para el patrón 4, con los pesos actuales de la red. Sin embargo, al probar nuevamente el patrón 1, en la Figura 18(g), podemos apreciar que regresamos al mismo estado que tenía la red al iniciar el entrenamiento y, por lo tanto, el procedimeinto de entrenamiento entró en un ciclo infinito. La manera de evitar que ésto suceda consiste en no permitir que las unidades que hallan sido utilizadas al

ajustar la red para obtener la salida deseada de un determinado patrón sean nuevamente seleccionadas para lograr la respuesta deseada de otro. Esto es lo que que hace el algoritmo MR II. Retomando el problema anterior, utilizando la técnica de marcar las Adalines como previamente seleccionadas; al llegar al estado mostrado en la Figura 18(d), donde la red falla para el patrón 3, en vez de seleccionar nuevamente la unidad de salida para ajustar la red, se deberá tomar la unidad inferior de la capa intermedia, dado que ésta tiene el valor de net más cercano a cero, de acuerdo al criterio de mínima perturbación. Al modificar, en esta ocasión, la unidad inferior de la capa oculta, tal como se muestra en la Figura 19, el entrenamiento quedará concluido, dado que la Madaline entregará la salida deseada para los cuatro patrones.



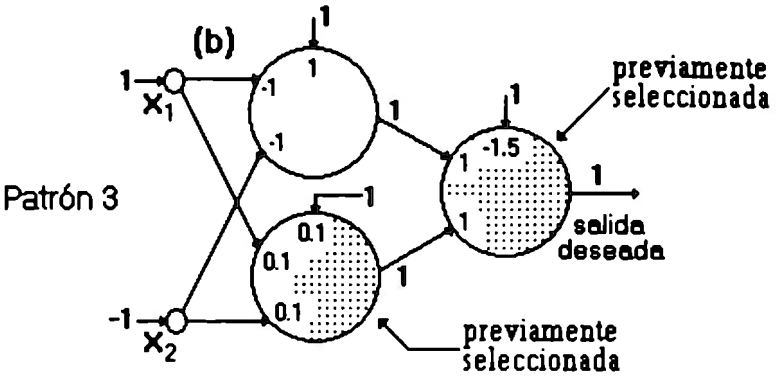


Figura 19. Entrenamiento de la Madaline para simular la función XOR

La figura 19(a) corresponde al estado presentado en la figura 18(d), sólo que, en esta ocasión, la Adaline de la capa de salida está marcada como previamente seleccionada y, por lo tanto, se elige la unidad inferior de la capa oculta para ser modificada, de acuerdo al criterio de mínima perturbación. Al realizar la modificación, como se muestra en la figura 19(b), la Madaline queda entrenada.

3.10 La necesidad de tomar combinaciones de Adalines para el entrenamiento

Como se puede observar, es necesario mantener el control de las Adalines que han sido modificadas para lograr el entrenamiento de la Madaline; sin embargo, esto trae consigo la necesidad de tomar parejas, ternas, cuartetas o combinaciones mayores de Adalines. Lo anterior se debe a que al modificar los pesos de una unidad para ajustar la red a un patrón dado y marcar dicha unidad como previamente seleccionada, llega un momento en que todas están marcadas o que las disponibles no provocan cambios favorables en la salida de la Madaline para el patrón aplicado en la entrada; entonces se intenta invertir la salida de dos unidades al mismo

tiempo para tratar de obtener cambios favorables en la salida. De igual forma, cuando se analizan todas las posibles parejas y no se logra disminuir el número de bits erroneos en la capa de salida, se toman combinaciones de tres Adalines y se invierten sus salidas para intentar disminuir el número de errores. Al agotarse todas los posibles ternas, se intenta con cuartetas y así sucesivamente.

Capítulo 4

Desarrollo del algoritmo de pastoreo para el entrenamiento de la Madaline

4.1 Justificación

Resumiendo el funcionamiento del algoritmo MR II, descrito en el capítulo anterior; podemos considerarlo como un algoritmo iterativo que utiliza una política de prueba y error para encontrar el conjunto de pesos de solución con los cuales se logrará que la Madaline entregue las salidas deseadas para todos los patrones del conjunto de entrenamiento. Dicha política consiste en elegir una Adaline e invertir su salida bipolar con el propósito de reducir el error para un determinado patrón aplicado en la entrada.

Para determinar cual unidad será modificada, se sigue el criterio de mínima perturbación consistente en elegir primero aquellas unidades en que se puede invertir su salida con el menor cambio en sus pesos; es decir, aquellas que tienen el valor de net más cercano a 0. Con lo anterior podemos decir que la forma en que el algoritmo se aproxima a los pesos de solución se basa en una "busqueda inteligente", donde dicha "inteligencia" proviene del hecho de considerar el criterio de mínima perturbación para elegir la siguiente Adaline de prueba. Dado que dicho criterio establece que se debe elegir la Adaline cuyo valor de net es más cercano a cero, es

posible que una misma unidad i sea seleccionada para reducir el error en iteraciones sucesivas, abriendose la posibilidad de que se genere un ciclo y el entrenamiento nunca termine.

Para evitar la formación de ciclos; se establece que las unidades que se hayan utilizado para reducir el error de un determinado patrón, queden marcadas como seleccionadas previamente de tal forma que no puedan utilizarse en iteraciones posteriores para reducir el error de otro patrón. Como consecuencia de lo anterior, llega un momento en que al intentar reducir el error de un determinado patrón, no es posible lograr cambios favorables en la salida al invertirse las respuestas bipolares de las Adalines disponibles (que no han sido utilizadas para reducir el error de otros patrones). En tal situación, se deben invertir simultaneamente las salidas de la pareja de unidades $\{U_i, U_j\}$ cuyos valores de net sean los más cercanos a 0. Una vez que se utiliza un pareja, debe ser marcada como seleccionanda previamente, de la misma forma que en caso de una sola unidad. Entonces, llega un momento en que no es posible reducir el error con las parejas de unidades disponibles y, por lo tanto, se hace necesario tomar tripletas de Adalines. Siguiendo este mismo esquema, es posible que posteriormente se requiera tomar cuartetas o combinaciones mayores de unidades para concluir el entrenamiento.

Los aspectos más importantes que se deben tomar en cuenta al analizar la eficiencia de un algoritmo son: la cantidad de recursos que consumen y el tiempo requerido para encontrar la solución del problema planteado.

En el algortimo MR II, la necesidad de seleccionar parejas, ternas, cuartetas o combinaciones mayores de Adalines para intentar reducir el número de bits erróneos en la capa de salida de la Madaline; implica la utilización de memoria adicional para llevar el control de las unidades o combinaciones de ellas, que han sido seleccionadas previamente. Este problema, en realidad, no es tan serio, ya que se pueden utilizar estructuras de almacenamineto adecuadas y algoritmos eficientes para administrarlas.

Partiendo de que cualquier método de prueba y error desperdicia tiempo en intentos infructuosos para lograr estados más favorables; entonces con el propósito de incrementar la eficiencia del algoritmo, es importante tratar de reducir el tiempo de procesamiento desperdiciado inutilmente.

En el caso de MR II, los factores que contribuyen a incrementar el tiempo requerido para concluir el entrenamiento son: la explosión combinatoria al tomar parejas, tripletas o combinaciones mayores de Adalines para reducir el error; y la cantidad de patrones "olvidados" cada vez que la Madaline "aprende" uno nuevo.

Para determinar la explosión combinatoria, utilizamos la fórmula $C = n! / (r! (n - r)!)$, donde C es el número de combinaciones de tamaño r que se pueden obtener de n unidades. Dado que *indicaciones preliminares establecen que es adecuado llegar a utilizar parejas de Adalines para reducir el error en redes de tamaño modesto con hasta 25 unidades por capa* (ref. 18), entonces podemos pensar que con redes de mayor tamaño se hace necesario llegar al nivel de tripletas o cuartetas. Por ejemplo, al analizar el

caso de una aplicación que requiera una Madaline con 500 unidades en una de sus capas; antes de comenzar a probar con parejas de Adalines, primero se deben invertir las salidas bipolares de todas las unidades que no han sido seleccionadas. El tiempo desperdiciado inútilmente hasta aquí no es tan grande dado que solamente se invierten las salidas de un número menor o igual que 500 Adalines. Sin embargo, cuando se hace necesario llegar hasta el nivel de tripletas, se tuvieron que haber invertido las salidas de 124,750 parejas de Adalines antes de llegar a dicho nivel. Además, debe considerarse que el tiempo necesario para invertir las salidas de un par de unidades, obviamente, es el doble que para una sola Adaline. El problema es más grave al momento de tener que llegar a la cuartetas ya que para ello se debieron invertir las salidas de 20,708,500 tripletas de Adalines. Entonces, el tiempo desperdiciado crece considerablemente conforme se van agotando las unidades y sus posibles combinaciones disponibles para reducir el número de errores en la salida.

La necesidad de utilizar parejas, tripletas o combinaciones mayores de unidades, también trae consigo la posibilidad de afectar demasiado a los patrones previamente aprendidos, cada vez que la red aprende uno nuevo en base a la modificación de los pesos de una combinación de Adalines. Por intuición; la probabilidad de afectar a un patrón previamente aprendido, si se modifican simultaneamente los pesos de dos o más unidades, es mayor que si se modifican los de una sola.

En base a lo expresado por Freeman y Skapura, después de presentar el algoritmo MR II en su libro Neural networks: algorithms, applications, an programming techniques (Octubre, 1991); *Al momento de realizar este*

escrito, el MR II estaba bajo experimentación para determinar sus características de convergencia y otras propiedades. (ref 3. pag. 75). Entonces resulta atractivo, además de analizar el MR II, tratar de desarrollar un nuevo algoritmo que supere algunos inconvenientes encontrados en él; ya que existe poca investigación sobre esta ruta y es posible aportar algo con el propósito de allanar el camino de investigaciones posteriores en esta rama específica de la Inteligencia Artificial.

4.2 Desarrollo

Tratando de superar, en algo, los factores que contribuyen a incrementar el tiempo requerido para el entrenamiento en algoritmo MR II, se desarrolló el **algoritmo de pastoreo para el entrenamiento de la Madaline.** Su nombre se deriva del orden en que se aplican los patrones del conjunto de entrenamiento a la red. Este algoritmo, al igual que el MR II, sigue una política de prueba y error con una búsqueda inteligente dirigida por el principio de mínima perturbación. Pero, para la dirección de la búsqueda, además, se considera el criterio de lograr el mayor cambio en el estado actual con el menor esfuerzo y, al mismo tiempo, causar el menor efecto sobre los patrones previamente aprendidos. Dicho algoritmo se describe a continuación:

0. Hacer k = 1. Para todas las Adalines de la red: asignar pesos arbitrarios en el rango [-1,1] y marcarlas como no seleccionadas previamente.

1. Aplicar el patrón k, a las unidades de entrada de la Madaline y propagar la señal hasta la capa de salida.

2. Determinar el error como el número de Adalines, de la capa de salida, cuya salida bipolar sea difierente del bit correspondiente de la salida deseada para el patrón aplicado a la Madaline.

3. Si el error es igual a cero, ir al paso 14.

4. Determinar el orden en que serán probadas las unidades de la capa oculta siguiendo el criterio de mínima perturbación. Esto es, construir una lista que apunte a cada Adaline de la capa oculta, de manera ascendente, en base a su valor absoluto de net. Mover el apuntador de dicha lista al inicio de la misma.

5. Avanzar el apuntador de la lista hasta encontrar una unidad que no haya sido seleccionada previamente o se haya recorrido toda la lista. Si el apuntador llegó al final, ir al paso 11. En caso contrario, considerar a la unidad en que se detuvo el apuntador como la Adaline de prueba.

6. Modificar los pesos de la Adaline de prueba, de tal forma que se invierta su salida bipolar.

7. Propagar la nueva salida bipolar de la unidad de prueba hasta la capa de salida.

8. Calcular nuevamente el error, de la manera descrita en el paso 2.

9. Si el error es mayor o igual que antes, restaurar los pesos originales, volver a propagar la salida bipolar y regresar al paso 5.

10. Marcar a la unidad de prueba como seleccionada previamente y actualizar el nuevo valor del error. Si el error es mayor que cero regresar al paso 5. En caso contrario, ir al paso 14.

11. Si el error es diferente de cero y existen más capas ocultas, repetir los pasos 4 al 10 para la siguiente capa oculta.

12. Para todas las Adalines de la capa de salida cuya respuesta bipolar es diferente del bit correspondiente de la salida deseada y que no han sido marcadas como seleccionadas previamente: modificar sus pesos de tal forma que se invierta su salida bipolar, marcarla como seleccionada previamente y decrementar el error.

13. Si no se pudo reducir el error original, obtenido en el paso 2, con las Adalines disponibles (que no han sido seleccionadas previamente); entonces, liberar todas las unidades de la Madaline (marcarlas como no seleccionadas previamente) y volver al paso 4.

14. Hacer i = 1.

15. Aplicar el patrón i, a las unidades de entrada de la Madaline y propagar la señal hasta la capa de salida.

16. Calcular el error como en el paso 2.

17. Si el error es mayor que cero, hacer k = i y regresar al paso 1.

18. Hacer i = i + 1. Si i es menor o igual que el número de patrones del conjunto de entrenamiento, regresar al paso 15. En caso contrario, terminar.

4.3 Descripción y análisis

El algoritmo debe partir de una Madaline en la cual todas las unidades están disponibles para ser elegidas con el propósito de reducir el error en la capa de salida. Además, los pesos de todas las Adalines pueden tomar valores negativos y positivos para representar entradas inhibitorias y excitativas respectivamente. Para evitar que dichos pesos se disparen (crezcan o disminuyan demasiado durante el entrenamiento), se inicializan con un valor aleatorio y pequeño, entre -1 y 1. Entonces, el paso 0 se utiliza unicamente como inicialización de la Madaline. La operación de hacer k = 1, es necesaria para que se aplique el primer patrón del conjunto de entrenamiento en el paso 1.

El paso 1 se refiere a la aplicación del patrón determinado por el valor de k en ese momento. La frase **aplicar el patrón k**, indica que se deben de forzar las salidas bipolares de cada una de las unidades de la capa de entrada al valor correspondiente de cada bit del patrón k. Cabe recordar que las unidades de la capa de entrada unicamente sirven para distribuir cada bit del patrón aplicado hacia todas las Adalines de la primer capa oculta. Por

eso es que esta capa, generalmente, se implementa como una lista cuyos elementos almacenan los bits (-1 o 1) del patrón en cuestión. La acción de **propagar** se refiere al hecho de que cada uno de los bits del patrón aplicado, son recibidos por las entradas asociadas a ellos, en cada una de las Adalines de la primer capa oculta, y multiplicados por el peso correspondiente. Esto es con la finalidad de obtener el valor de net de cada unidad y poder determinar su salida bipolar. Una vez obtenidas las salidas bipolares de cada una de las Adalines de la capa oculta, éstas serán distribuidas hacia las entradas de las unidades de la siguiente capa oculta o de la capa de salida. El procedimiento de propagación se detiene cuando se obtienen las salidas bipolares para cada una de las Adalines de la capa de salida. Durante este proceso de propagación hacia adelante, **feed forward**, el patrón presentado en la entrada se va deformando, tanto en su tamaño como en sus bits, hasta que en la salida se obtiene un patrón que depende completamente del patrón en la entrada, del número de neuronas en cada capa y de los pesos de conexión entre las Adalines.

Para una Madaline con una sola capa oculta. Consideremos la siguiente notación:

n -> número de unidades de la capa de entrada

r -> número de Adalines de la capa oculta

m -> número de Adalines de la capa de salida

$\mathbf{E_i}$ -> el *i*-ésimo bit del patrón de entrada

$\mathbf{O_i.W_k}$ -> el *k*-ésimo peso de la *i*-ésima Adaline de la capa oculta

$\mathbf{O_i.net}$ -> el valor de net de la *i*-ésima Adaline de la capa oculta

$\mathbf{O_i.sb}$ -> la salida bipolar de la *i*-ésima Adaline de la capa oculta

$S_i.W_k$ -> el *k*-ésimo peso de la *i*-ésima Adaline de la capa de salida

$S_i.net$ -> el valor de net de la *i*-ésima Adaline de la capa de salida

$S_i.sb$ -> la salida bipolar de la *i*-ésima Adaline de la capa de salida

Entonces, podemos describir el procedimiento de propagación, de la siguiente manera:

1. Para i = 1 hasta r :

1.1. $O_i.net = O_i.W_0$

1.2. Para j = 1 hasta n : $O_i.net = O_i.net + E_j * O_i.W_j$

1.3. Si $O_i.net >= 0$, $O_i.sb = 1$. En caso contrario, $O_i.sb = -1$

2. Para i = 1 hasta m :

2.1. $S_i.net = S_i.W_0$

2.2. Para j = 1 hasta r : $S_i.net = S_i.net + O_j.sb * S_i.W_j$

1.3. Si $S_i.net >= 0$, $S_i.sb = 1$. En caso contrario, $S_i.sb = -1$

Al obtener las respuestas bipolares para cada una de las Adalines de la capa de salida, podemos considerarlas como los elementos de un vector $Y = (y_1,y_2,..,y_m)$. Representando la salida deseada, para el patrón aplicado en la entrada, como el vector $D = (d_1,d_2,..,d_m)$ y considerando que tanto los elementos de Y como los de D solo pueden tomar los valores de -1 o 1; entonces podremos expresar el error de la siguiente manera:

$$error = \sum_{j=1}^{m} \frac{|d_j - y_j|}{2}$$

Dado que el valor absoluto de $d_j - y_j$ arrojará un valor de 2 cuando los bits sean diferentes, entonces es necesario dividirlo entre 2 para que el error se incremente en 1. Cuando los bits sean iguales, el valor absoluto de $d_j - y_j$ arrojará un valor de 0 y, por lo tanto, el error no se incrementará. Aunque, ésta es una manera elegante de expresar el error, al momento de implementarse en software es más recomendable emplear una rutina en la cual el error se inicialice en 0 y comparar todos los elementos de los dos vectores, cuando sean diferentes incrementar el error. Esto último es lo que se hace en el paso 2, basándose en el hecho de que el tiempo requerido para realizar una resta, obtener el valor absoluto de la misma y dividirlo entre 2; es mucho mayor que una simple comparación. Si el patrón obtenido a la salida de la Madaline es igual a la salida deseada, entonces el error tendrá un valor de 0; en cuyo caso es necesario realizar cierto procedimiento, que se explica más adelante, para determinar cual será el siguiente patrón a aplicarse en la entrada. Esto último implica la necesidad del paso 3.

Considerando que cualquier método de prueba y error desperdicia tiempo en intentos infructusos por lograr estados más favorables, es deseable que cualquier esfuerzo logre el mayor cambio en el estado actual. Motivo por el cual, en este algoritmo se intenta primeramente seleccionar las Adalines de la primer capa oculta. De tal forma que, al invertir su salida y propagarla a través de la red, se inviertan las salidas del mayor número posible de unidades de la capa de salida.

Es posible suponer que al inicio del entrenamiento, es más probable que las respuestas obtenidas en la Madaline para cada patrón difieran en muchos bits de la salida deseada para el mismo. Por lo tanto, al invertir las salidas de las Adalines de la primer capa oculta y provocar la modificación de las respuestas de un determinado número, posiblemente grande, de unidades de salida; entonces existe una mayor probabilidad de que se reduzca, en vez de aumentar, en número de bits erroneos.

Cada vez que una unidad de la capa oculta se utiliza para reducir el error de un determinado patrón, dicha unidad queda marcada como seleccionada previamente. En base a lo anterior, y considerando que tanto el tamaño de la capa de entrada como el de la capa de salida dependen directamente de los patrones de entrada y salida respectivamente, entonces una buena medida para conseguir el entrenamiento es utilizando una cantidad considerable de Adalines en las capas ocultas e intentar primero con ellas reducir el error de los patrones aplicados a la Madaline.

Puede pensarse que el hecho de lograr que se modifiquen un número considerable de bits de la salida de la Madaline cada vez que se invierte la salida de una unidad de las capas ocultas, traiga consigo problemas al momento de llegar a la fase final del entrenamiento en la cual la diferencia entre las salidas deseadas y las salidas obtenidas, para los patrones aplicados, sea mínima. Sin embargo, debemos recordar que las unidades quedan marcadas como previamente seleccionadas cada vez que se utilizan para reducir el error de un patrón. Entonces, para las últimas iteraciones del entrenamiento, lo más probable es que la mayoría de las Adalines de las

capas ocultas no se encuentren disponibles y, por ello, el entrenamiento recaiga en las unidades de la capa de salida. Algo muy importante en esto último es que se debe invertir la respuesta de las unidades de la capa de salida de tal forma que no se realicen cambios muy bruscos en los pesos. Esto es con el propósito de no afectar a los patrones previamente aprendidos en base al ajuste de las unidades de las capas ocultas.

Con lo anterior se justifica el por qué seleccionar primeramente las unidades de las capas ocultas. Una vez determinada la capa de unidades sobre la que se seleccionarán las Adalines, se debe determinar el orden en que serán elegidas las unidades de dicha capa. Esta selección se basa en el criterio de mínima perturbación, el cual consiste en seleccionar primero para modificación, la Adaline en la cual se logre invertir su salida bipolar con el menor cambio en sus pesos. Entonces, dado que la salida bipolar esta en función del valor de net, una manera sencilla de establecer dicho orden es construyendo una lista cuyos elementos contengan, cada uno, el número de una unidad de la capa en cuestión. La manera de determinar qué número de Adaline va en cada elemento es siguiendo un orden ascendente de acuerdo al valor absoluto de net de cada unidad. De tal forma que el elemento uno tendrá el número de la Adaline que tenga el menor valor absoluto de net, en el segundo elemento se encontrará la unidad cuyo valor absoluto de net sea el más cercano al de la Adaline contenida en el elemento 1, y así sucesivamente. La Figura 20 muestra la manera en que se construye dicha lista.

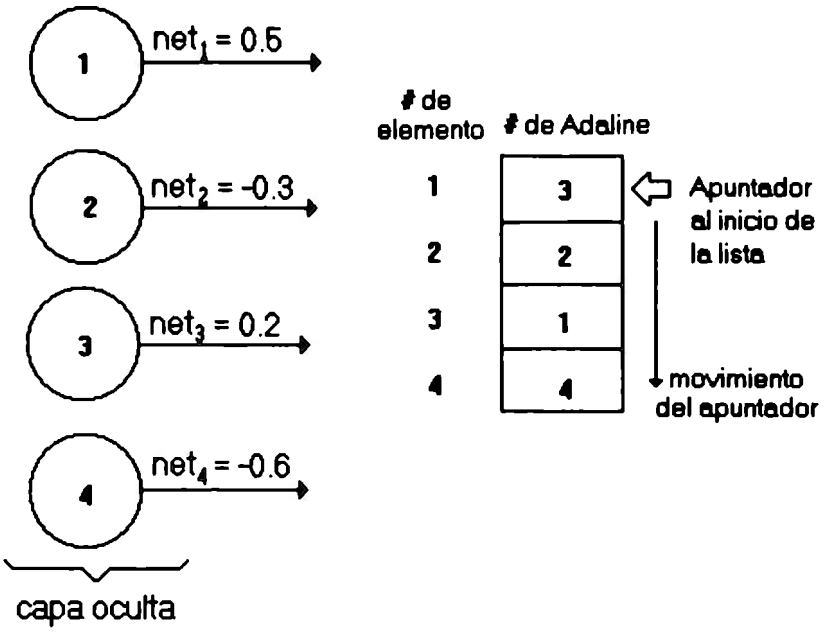


Figura 20. Determinación del orden en que serán probadas las unidades de una capa oculta

Cada círculo representa una Adaline. Los números dentro de cada círculo representan el número de la Adaline. Los elementos de la lista contienen los números de Adaline de tal forma que se establece un orden de menor a mayor valor de net. Inicialmente, el apuntador de la lista apunta al inicio de la misma. El número de unidad contenido en el elemento indicado por el apuntador, es el número de la Adaline a probar. El apuntador se desplaza en el sentido indicado por la flecha de movimiento del mismo.

El considerar el valor absoluto de net se debe a que se invertirá la salida bipolar de la unidad seleccionada y dado que la salida bipolar se obtiene de acuerdo a la función de activación de la Adaline que es la función escalón bipolar o signo, entonces los valores de net más cercanos a 0, independientemente de que sean negativos o positivos, son los más faciles de incrementar o decremantar, mediante un cambio en los pesos de la unidad, de tal manera que cambien de signo y, por lo tanto, la salida bipolar

de la Adaline se invertirá. Más adelante se explica detalladamente la manera en que se logra invertir la salida de una unidad.

La determinación del orden en que se probarán las Adalines de una capa oculta y, por lo tanto, la construcción de la lista se hace en el paso 4. Aquí mismo se mueve el apuntador de la lista al inicio de la misma.

Una vez construida la lista, es necesario recorrerla para intentar reducir el error de un determinado patrón con las Adalines que se encuentran disponibles (que no han sido seleccionadas previamente); esto es precisamente lo que se realiza en el paso 5. La manera en que se realiza lo anterior, es analizando el estado de la Adaline *i*, donde *i* es el número almacenado en el elemento indicado por el apuntador de la lista en ese momento, para determinar si está disponible. Si la Adaline no se encuentra disponible, se debe avanzar el apuntador hasta localizar una que sí lo esté o hasta que se haya encontrado el final de la lista. En el caso de que se haya encontrado una unidad disponible, ésta deberá considerarse como la **Adaline de prueba**. Esto significa que se deberá invertir la salida de dicha unidad, tal como se indica en el paso 6, para posteriormente propagar únicamente la nueva respuesta de la Adaline de prueba, de acuerdo a lo establecido en el paso 7.

Como ya se mencionó anteriormente, es importante elegir un procedimiento adecuado al momento de invertir la salida bipolar de una Adaline en base al ajuste de los pesos de la misma. Al tratar de invertir una salida tenemos dos posiblidades; la primera es que su valor actual sea -1 y se debe invertir a 1, la segunda es que su valor actual sea 1 y se debe

invertir a -1. En el primer caso, es necesario aumentar el valor de todos aquellos pesos que esten asociados a entradas que en ese instante se encuentren con un valor de 1 y se deberán disminuir aquellos en los cuales las señales de entrada asociadas tengan un valor de -1. En el segundo caso, es necesario disminuir el valor de todos aquellos pesos que esten asociados a entradas que en ese instante se encuentren con un valor de 1 y se deberán aumentar aquellos en los cuales las señales de entrada asociadas tengan un valor de -1. A continuación se presenta un procedimiento iterativo que utiliza un cierto valor de incremento para lograr invertir la respuesta de una Adaline como la que se muestra en la Figura 12:

Caso I (de -1 a 1):

1. $W_0 = W_0 + inc$

2. Para i = 1 hasta n: $W_i = W_i + (X_i * inc)$

3. $net = W_0$

4. Para i = 1 hasta n: $net = net + (X_i * W_i)$

5. Si net < 0, regresar al paso 1

6. **Salida** = 1

Caso II (de 1 a -1):

1. $W_0 = W_0 - inc$

2. Para i = 1 hasta n: $W_i = W_i - (X_i * inc)$

3. $net = W_0$

4. Para i = 1 hasta n: **net** = **net** + (X_i * W_i)

5. Si **net** >= 0, regresar al paso 1

6. **Salida** = -1

Una vez obtenida la nueva respuesta para la Adaline de prueba, se debe determinar la nueva salida de la Madaline. Para lo cual no es necesario propagar nuevamente el patrón desde la entrada de la Madaline, basta con propagar únicamente la salida de la unidad de prueba. Considerando como ***siguiente capa,*** a la capa subsecuente cuyas Adalines reciben directamente la salida de la unidad de prueba. Entonces es posible emplear un procedimiento que consiste en modificar directamente el valor de net de las unidades de la siguiente capa . Considerando la siguiente notación:

p -> número de la Adaline de prueba

A_p**.sb** -> nueva salida bipolar de la Adaline de prueba

g -> número de unidades de la siguiente capa

$U_i.W_k$ -> el *k*-ésimo peso de la *i*-ésima Adaline de la siguiente capa

U_i**.net** -> el valor de net de la *i*-ésima Adaline de la siguiente capa

U_i**.sb** -> la salida bipolar de la *i*-ésima Adaline de la siguiente capa

SC -> capa considerada como la siguiente capa

CA -> capa actual de trabajo

A_i**.sb** -> la salida bipolar de la *i*-ésima Adaline de la capa actual

h -> número de unidades de la capa actual

Entonces, es posible expresar la rutina de propagación de una sola unidad, de la siguiente manera:

1. Para i = 1 hasta **g** :

1.1. $U_i.net = U_i.net + (2 * A_p.sb * U_i.W_p)$

1.2. Si $U_i.net < 0$, $U_i.sb = -1$. En caso contrario, $U_i.sb = 1$

2. Si **SC** es igual a la capa de salida, terminar.

3. Hacer **CA** = **SC**

4. Hacer **SC** igual a la capa subsecuente de **CA**

5. Para i = 1 hasta **g** :

5.1. $U_i.net = U_i.W_0$

5.2. Para j = 1 hasta **h** : $U_i.net = U_i.net + A_j.sb * U_i.W_j$

5.3. Si $U_i.net >= 0$, $U_i.sb = 1$. En caso contrario, $U_i.sb = -1$

6. Regresar al paso 2

El paso 1 de este procedimiento únicamente modifica las unidades de la capa siguiente a la capa en que se encuentra la unidad de prueba. Cuando la capa siguiente es la de salida, el procedimiento logra la propagación completa. Cuando no es así, se hace necesario propagar el resto de las capas completas, a partir de la siguiente capa, hasta la salida. Esto último se realiza en los pasos 2 al 6. El propagar solamente la nueva salida de la unidad de prueba, en vez de el patrón completo desde la entrada, elimina el tiempo inútil empleado para calcular las salidas de todas las unidades que

pertenecen a capas anteriores a la capa en que se encuentra la Adaline de prueba. Además, se ahorra el tiempo requerido para calcular las salidas del resto de las unidades de la misma capa y también la necesidad de que dichas salidas se utilicen para calcular las nuevas respuestas de las Adalines de la capa subsecuente.

Volviendo nuevamente al algoritmo de pastoreo en el paso 8, una vez que se propaga la nueva respuesta de la Adaline de prueba y se obtiene la nueva salida de la Madaline, es necesario volver a calcular el error para el patrón aplicado, de acuerdo al procedimiento descrito previamente.

Si el nuevo error es mayor o igual que antes, se deben restaurar los pesos que tenía la Adaline de prueba antes de invertir su salida bipolar. Una vez que se ha realizado lo anterior, tanto el valor de net como la salida bipolar serán los originales. Como la salida invertida de la unidad de prueba ya fue propagada, es necesario volver a propagar la salida original de la misma, para ajustar nuevamente la red al estado que tenía antes de la prueba. Todo lo anterior se indica en el paso 9.

Algo que debe explicarse aquí es, ¿por qué perder tiempo en restaurar el estado original de la red cuando el nuevo error es igual al de antes? La justificación de lo anterior radica en el hecho de que, al modificar los pesos de la Adaline de prueba es probable que la red olvide algunos de los patrones previamente aprendidos; entonces es mejor hacer un doble esfuerzo para restaurar el estado original, a tener que hacer un esfuerzo mucho mayor para volver a aprenderse los patrones olvidados.

Si no fue posible reducir el error con la Adaline de prueba, es necesario seguir recorriendo la lista a partir de donde se encuentra el apuntador, para localizar la siguiente unidad disponible. En caso de que sí se haya reducido el error, tal como se indica en el paso 10, es necesario marcar la unidad de prueba como seleccionada previamente (ya no estará disponible) y se deberá considerar al error actual como el nuevo error a reducir. Si el nuevo error es mayor que 0, deberá seguirse recorriendo la lista para probar con la siguiente unidad disponible de la capa oculta con que se está trabajando. En caso de que el error se haya reducido hasta un valor de 0, es necesario seleccionar el siguiente patrón a aplicarse en la entrada de la red.

Si después de haber recorrido toda la lista, el error sigue siendo mayor que 0; entonces, si existen más capas ocultas, seleccionar la siguiente e intentar reducir el error con las Adalines que pertenecen a ella. Esto último se indica en el paso 11. En el caso de que ya no existan más capas ocultas y el error siga siendo mayor que 0, de acuerdo con lo indicado en el paso 12, se deberá invertir la salida (según el procedimiento descrito anteriormente) de todas aquellas Adalines de la capa de salida que no han sido seleccionadas previamente y cuya respuesta bipolar es diferente del bit correspondiente de la salida deseada para el patrón en la entrada. Cada vez que se invierta la salida de una Adaline, es necesario marcarla como seleccionada previamente y se decrementará el error.

Si no fue posible reducir el error original obtenido en el paso 2 cuando recién se aplicó el patrón en la entrada, porque no se encontró alguna Adaline disponible que lograra disminuir dicha diferencia; entonces se

deben **liberar** (desmarcar como seleccionadas previamente) todas las unidades de la Madaline, para que el entrenamiento pueda proseguir. Al dejar disponibles a todas las unidades, se deberá intentar reducir el error para el mismo patrón en la entrada partiendo nuevamente con la primer capa oculta, tal y como se indica en el paso 13. Dado que fue necesario liberar todas las unidades porque las que se encontraban disponibles no lograron reducir el error, entonces la unidad que logrará reducirlo debe ser alguna de las que no estaban disponibles y, por lo tanto, esto provocará que la red olvide algunos patrones previamente aprendidos al invertirse la salida bipolar de la Adaline seleccionada. Sin embargo, intuitivamente podemos suponer que el número de patrones olvidados es menor que si se invirtieran las salidas de dos o más unidades. Los patrones que resultan afectados por la modificación de una de las unidades que se habían utilizado en su aprendizaje, podrán ser aprendidos de nuevo mediante Adalines que habían quedado disponibles antes de desmarcarlas a todas. De esta forma es como los pesos de las unidades van convergiendo a la solución de la aplicación planteada.

Otro aspecto que contribuye a la convergencia de los pesos hacia la solución, es la manera en que se determina cual será el siguiente patrón a aplicarse en la Madaline. Cada vez que se intenta reducir el error para un determinado patrón, aún cuando no se haya eliminado completamente, es posible que la modificación de las unidades que se seleccionaron para ello, provoque que la Madaline olvide algunos de los patrones previamente aprendidos.

Para determinar cual será el próximo patrón a aplicar a la entrada, podemos considerar dos casos. El primero se presenta cuando no fue posible eliminar completamente el error para un patrón dado. Un criterio de selección y solución para esto, consiste en dejar el mismo patrón en la entrada, liberar todas la unidades y reintentar el ajuste de la red hasta que se arroje la salida deseada. El segundo caso es cuando se elimina completamente el error para el patrón en la entrada, aquí podemos aplicar el criterio de elegir el siguiente patrón en el conjunto de entrenamiento. Al utilizar dichos criterios de selección del próximo patrón a ser aplicado; cada vez que se elimina el error para el último patrón del conjunto de entrenamiento, se hace necesario revisar que la Madaline entregue la salida deseada para todos los patrones. Si existe algún patrón cuya salida obtenida sea diferente de su salida deseada, se deberá volver a presentar consecutivamente los patrones, a partir del cual se obtuvo el error.

Existe un tercer criterio de selección del próximo patrón, éste puede utilizarse tanto en el caso de reducción parcial del error como en el caso de la eliminación completa del mismo, y que también permite determinar cuando el entrenamiento ha concluido. Dicho criterio consiste en que cada vez que se reduce total o parcialmente el error de un determinado patrón, basándose en la premisa de la posibilidad de que la red olvide algunos patrones previamente aprendidos, se deben aplicar de nuevo todos los patrones del conjunto de entrenamiento, desde el primero de ellos. El primer patrón en el cual el error sea mayor que 0 deberá ser el próximo patrón a aplicarse. De esta forma los patrones que fueron olvidados serán aprendidos nuevamente, antes de proseguir con el patrón consecutivo al que se le redujo el error.

Esta forma de selección tiene algo de semejanza con la labor que realizan los pastores al arrear la ovejas, cuando una se queda rezagada tiene que volver por ella antes de seguir impulsando al resto del rebaño. Dado que en este algoritmo se utiliza dicho criterio de selección, se le dio el nombre de **algoritmo de pastoreo para el entrenamiento de la Madaline.**

La decisión de optar por el tercer criterio de selección se debe a la experimenteción. Se implementaron dos Madalines para resolver el mismo problema, la primera con los dos primeros criterios y la segunda con el tercer criterio. Después de entrenarlas varias veces, se observó que la segunda Madaline siempre terminaba primero. Posteriormente se hizo lo mismo para resolver otros problemas, y los resultados fueron iguales que los anteriores. Durante el entrenamiento de la primer Madaline, cuando la red aprende un nuevo patrón y olvida uno o más de los aprendidos previamente, los patrones que se olvidaron quedan pendientes de aprender hasta que el entrenamiento llegue al último patrón y comience de nuevo a recorrer el conjunto de entrenamiento.

Mientras que en el entrenamiento de la segunda Madaline, cuando se ajusta la red para reducir el error del patrón *i*, los que se olvidaron son reaprendidos antes de presentar el patrón $i + 1$ a la Madaline. En los pasos 14 al 18 se describe la manera secuencial en que se prueban, desde el primero, cada uno de los patrones del conjunto de entrenamiento. Cuando existe error para uno de ellos, dicho patrón se considera como el siguiente a ser aplicado y el algoritmo regresa al paso 1 para tratar de reducir el error obtenido. Si se recorre todo el conjunto de patrones y se obtienen las salidas deseadas para todos, el entrenamiento termina.

Capítulo 5

Entrenamiento de una Madaline para corrección de ruido

5.1 Descripción

La aplicación más importante de los modelos de redes neuronales es el **reconocimiento de patrones,** ya que sus estructuras les permiten que la información no se represente de manera localizada. Es decir, no es posible señalar a una determinada conexión o neurona como memoria de un dato determinado, sino que la información está distribuida a través de los pesos de las conexiones de toda la red, que son ajustados durante el proceso de entrenamiento. Este fenómeno es el que le dá a la red su característica de generalización, permitiendole que al aplicar un patrón que está fuera del conjunto de entrenamiento, la red pueda entregar una determinada salida que probablemente sea la misma que la de uno de los patrones del conjunto de entrenamiento.

Los patrones que son presentados a una Madaline deben estar codificados en forma bipolar (-1 ó 1). De esta manera, para introducir la letra A mediante el código ASCII de 8 bits, se debe aplicar el patrón -11-1-1-1-1-11. Entonces, la dimensión (número de neuronas) de la capa de entrada debe ser 8, y por ello, podemos presentar 2^8 patrones (caracteres) diferentes a la red. Considerando que solamante interesa almacenar las

letras de la A a la Z, si queremos entrenar a la red para que nos establesca una función **f(x)** *inyectiva* de la entrada con respecto a la salida, necesitaremos para mapear nuestro conjunto de entrenamiento, una salida compuesta por, al menos, 5 Adalines. Esto se puede visualizar en la Tabla 3.

Tabla 3. Descripción de una función inyectiva para el alfabeto

Cada letra tiene un patrón asociado mediante el código ASCII. En la salida de la Madaline, se obtiene un número binario correspondiente al orden de aparición de la letra dentro del alfabeto.

Caracter	Entrada	Salida=f(Entrada)
A	-1 1-1-1-1-1-1 1	-1-1-1-1-1
B	-1 1-1-1-1-1 1-1	-1-1-1-1 1
C	-1 1-1-1-1-1 1 1	-1-1-1 1-1
:	:	:
Y	-1 1-1 1 1-1-1 1	1 1-1-1 1
Z	-1 1-1 1 1-1 1-1	1 1-1 1-1

Hagamos **D** igual al *dominio* de **f**, que es exactamente nuestro conjunto de entrenamiento, y **R** igual al *rango* de **f**, que es el conjunto de las salidas deseadas. Supongamos que entrenamos la Madaline para establecer una nueva función **g(x)**, cuyo dominio incluye a todos los posibles patrones de 8 bits, pero su rango sigue siendo **R**. Entonces, **g** no puede ser inyectiva, tal como se muestra en la Tabla 4.

Tabla 4. Descripción de una función para el conjunto de patrones de 8 bits

Existen diferentes patrones en la entrada que tienen asociada la misma salida.

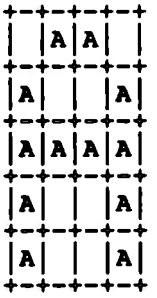
Caracter	Entrada	Salida=g(Entrada)
A	-1 1-1-1-1-1-1 1	-1-1-1-1-1
B	-1 1-1-1-1-1 1-1	-1-1-1-1 1
:	:	:
Z	-1 1-1 1 1-1 1-1	1 1-1 1-1
	1 1-1-1-1-1-1 1	-1-1-1-1-1
	1 1-1-1-1-1 1-1	-1-1-1-1 1
	:	:
	1 1-1 1 1-1 1-1	1 1-1 1-1
	:	:

De la Tabla 4, se puede apreciar que haciendo X_1 = -11-1-1-1-1-11 y X_2 = 11-1-1-1-1-11, $g(X_1) = g(X_2)$ = -11-1-1-1-1-11. Entonces, la red entrega la misma salida para dos patrones de entrada diferentes. Pero, ¿Ésto de qué puede servir?

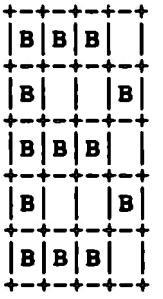
Lo anterior constituye la base para la corrección automática de ruido en la transmisión de datos. En el ejemplo anterior, X_2 puede ser el patrón recibido en una transmisión cuyo verdadero patrón enviado fue X_1. Sólo que durante el viaje, el ruido hizo que el primer bit se modificara a 1. Entonces, el entrenar la red para que reconozca los patrones originales y algunos derivados de posibles errores en los mismos, junto con la cualidad de asociar automáticamente una determinada salida a los patrones que están fuera del conjunto de entrenamiento; hacen de las redes neuronales una solución alternativa a las demás técnicas existentes para la corrección automática de ruido.

Dado que la función principal de esta aplicación es comprobar el funcionamiento del algoritmo de pastoreo para el entrenamiento de la Madaline, solamente nos interesa almacenar las letras del alfabeto. Sin embargo; con el propósito de dar más "potencia" a la señal que se va a transmitir y, con ello, mayor tolerancia al ruido, se utilizan 20 bits para codificar cada letra. La explicación de este incremento en el número de bits es similar a cuando estamos platicando y el ruido del ambiente no permite escuchar con claridad, entonces aumentamos el volumen de la voz para ser escuchados a pesar del ruido existente.

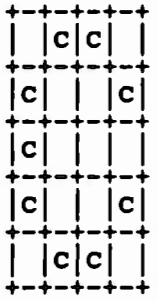
Para determinar la representación de cada letra, ésta será dibujada en una retícula de 5X4 puntos. Además, con el objeto de acelerar el entrenamiento, se utilizan 26 bits de salida para identificar cada letra. La forma en que se determina cual letra se obtuvo en la salida, consiste en localizar cual fue el bit de mayor orden que se encendió (tiene un valor de 1). Las representaciones de cada letra junto con su salida deseada se presentan a continuación:



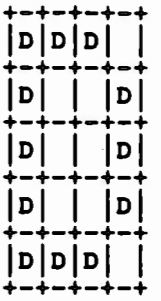
letra: A
patrón: -111-11-1-1111111-1-111-1-11
salida: 1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1



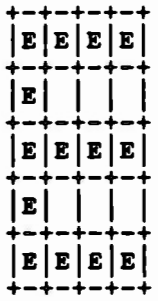
letra: B
patrón: 111-11-1-11111-11-1-11111-1
salida: /1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1



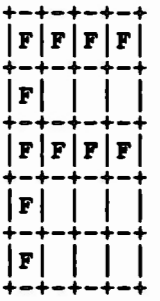
letra: C
patrón: -111-11-1-111-1-1-11-1-11-111-1
salida: //1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1



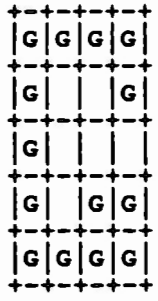
letra: D
patrón: 111-11-1-111-1-111-1-11111 -1
salida: ///1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1



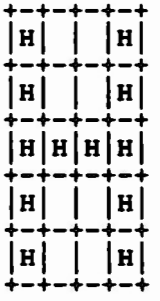
letra: E
patrón: 11111-1-1-111111-1-1-11111
salida: ////1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1



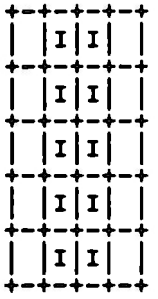
letra: F
patrón: 11111-1-1-111111-1-1-11-1-1-1
salida: /////1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1



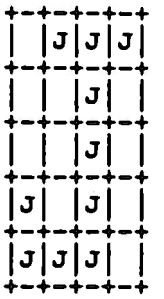
letra: G
patrón: 11111-1-111-1-1-11-1111111
salida: //////1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1



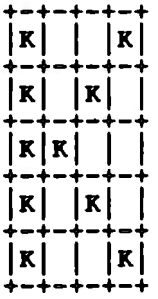
letra: H
patrón: 1-1-111-1-1111111-1-111-1-11
salida: ///////1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1



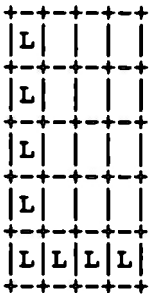
letra: I
patrón: -111-1-111-1-111-1-111-1-111-1
salida: ///////1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1



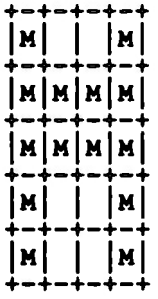
letra: J
patrón: -1111-1-11-1-1-11-11-11-1111-1
salida: ////////1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1



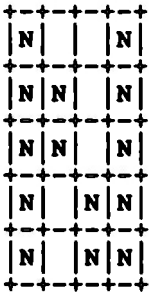
letra: K
patrón: 1-1-111-11-111-1-11-11-11-1-11
salida: ////////1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1



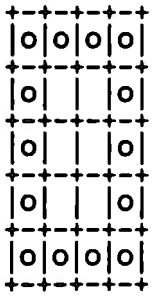
letra: L
patrón: 1-1-1-11-1-1-11-1-1-11-1-1-11111
salida: /////////1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1



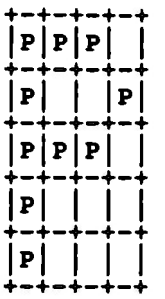
letra: M
patrón: 1-1-11111111111-1-111-1-11
salida: /////////1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1



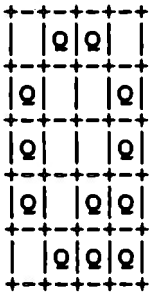
letra: N
patrón: 1-1-1111-1111-111-1111-111
salida: ////////1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1



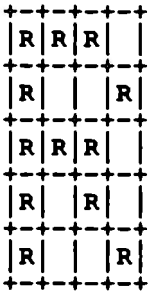
letra: O
patrón: 11111-1-111-1-111-1-111111
salida: ////////////1-1-1-1-1-1-1-1-1-1-1-1



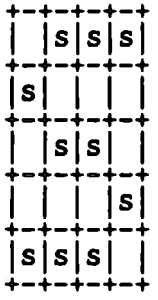
letra: P
patrón: 111-11-1-11111-11-1-1-11-1-1-1
salida: ////////////1-1-1-1-1-1-1-1-1-1-1



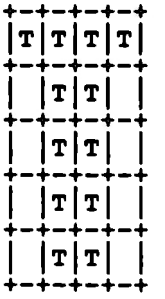
letra: Q
patrón: -111-11-1-111-1-111-111-1111
salida: ////////////1-1-1-1-1-1-1-1-1-1-1



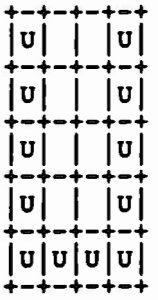
letra: R
patrón: 111-11-1-11111-11-11-11-1-11
salida: ////////////1-1-1-1-1-1-1-1-1



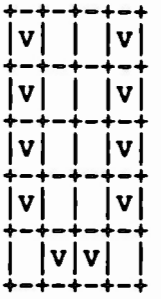
letra: S
patrón: -11111-1-1-1-111-1-1-1-11111-1
salida: ////////////////1-1-1-1-1-1-1-1



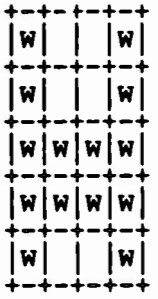
letra: T
patrón: 1111-111-1-111-1-111-1-111-1
salida: ////////////////1-1-1-1-1-1-1



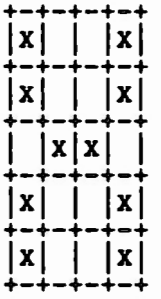
letra: U
patrón: 1-1-111-1-111-1-111-1-111111
salida: ////////////////1-1-1-1-1-1



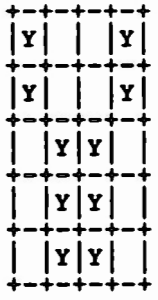
letra: V
patrón: 1-1-111-1-111-1-111-1-11-111-1
salida: ////////////////1-1-1-1-1



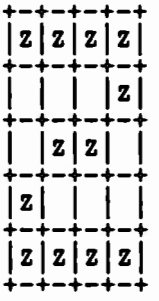
letra: W
patrón: 1-1-111-1-11111111111-1-11
salida: ////////////////1-1-1-1



letra: X
patrón: 1-1-111-1-11-111-11-1-111-1-11
salida: ////////////////1-1-1



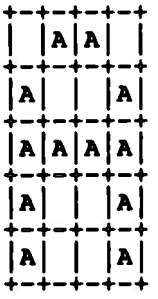
letra: Y
patrón: 1-1-111-1-11-111-1-111-1-111-1
salida: ////////////////1-1



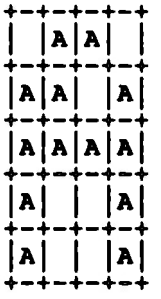
letra: Z
patrón: 1111-1-1-11-111-11-1-1-11111
salida: ////////////////1

La diagonal que se utiliza para representar los bits de las salidas deseadas de cada letra, representa que es indistinto el valor de ese bit. Lo anterior se debe a que, de acuerdo al criterio del bit de mayor orden, se recorren los bits de la salida obtenida comenzando desde el de mayor peso (26) hasta localizar uno que tenga un valor de 1. Entonces, el valor del resto de los bits de menor peso no interesa.

Con el propósito de aumentar el poder de generalización de la red, se agregaron al conjunto de entrenamiento, además de los patrones originales mostrados anteriormente, los patrones producidos por la alteración de 1 bit de cada patrón original. Por ejemplo, en la Figura 21 se muestran dos patrones cuya salida corresponde a la letra A.



letra: A
patrón original : -111-11-1-1111111-1-111-1-11
salida: 1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1



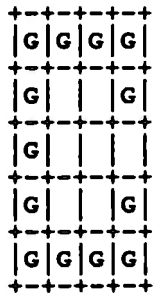
letra: A
patrón alterado: -111-111-1111111-1-111-1-11
salida: 1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1

Figura 21. Patrón original y patrón distorcionado para la letra A

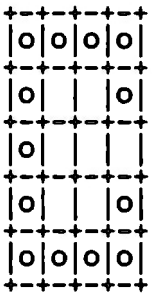
El patrón distorcionado corresponde a la alteración del sexto bit del patrón original para la letra A.

Entonces, para cada letra tenemos 21 patrones asociados, lo que genera un total de 546 patrones posibles. Sin embargo; dado que en algunos casos, al alterar un bit de un patrón original se genera un nuevo

patrón que coincide con la modificación de 1 bit de otro patrón original, tal como se muestra en la Figura 22; entonces el conjunto de entrenamiento de esta aplicación está formado por un total de 528 patrones de 20 bits. Al estar generando todas las combinaciones posibles de un bit alterado para cada patrón original (correspondiente a una determinada letra), cuando se producían patrones iguales partiendo de dos diferentes patrones originales, se utilizó el criterio de eliminar uno y asociar el otro a la letra que tiene la mayor frecuencia de uso



letra: G
patrón: 11111-1-111-1-1-11-1-111111
* se elimina



letra: O
patrón: 11111-1-111-1-1-11-1-111111
salida: ////////////1-1-1-1-1-1-1-1-1-1-1-1

Figura 22. Patrones iguales, generados por alteraciones en distintas letras

Se elimina el patrón generado por la letra G y se agrega el otro, generado por la letra O, al conjunto de entrenamiento.

5.2 Implementación

El software desarrrollado para esta aplicación, consiste de dos módulos. El primero se utiliza para entrenar la Madaline mediante el algoritmo de pastoreo, para que reconozca el conjunto de entrenamiento descrito anteriormente. El segundo se utiliza para comunicar dos

computadoras a través del puerto serial, introduciendo ruido de manera automática y/o manual sobre la letra que se transfiere. En la computadora receptora se implementa la Madaline previamente entrenada en el módulo 1, para que corrija la letra afectada por el ruido.

La Madaline que se utiliza, tiene 20 unidades en la capa de entrada, 600 en la capa oculta y 26 en la capa de salida. El módulo 1 consiste en el programa **ENTRENA.EXE**, el cual lee el archivo texto **PATRONES.TXT** que contiene los patrones del conjunto de entrenamiento, entrena la Madaline y graba los valores de los pesos de las Adalines de la capa oculta y de la capa de salida en los archivos **OCULTA.DAT** y **SALIDA.DAT** respectivamente. El módulo 2 está formado por dos programas, el primero es **ENVÍA.EXE** y el segundo es **RECIBE.EXE**.

El programa ENVÍA, tiene la función de mostrar una retícula de 5X4 en la cual se dibuja la representación de las letras introducidas por el teclado. El usuario puede modificar la figura trazada en la retícula. En cualquier momento se puede transmitir, por el puerto serial, el contenido de la retícula. Además, existe la opción de introducir ruido automáticamente.

El programa RECIBE, implementa la Madaline descrita anteriormente; para ello, carga los pesos de las Adalines de la capa oculta y de la capa de salida que fueron grabados previamente en los archivos OCULTA.DAT y SALIDA.DAT. También presenta una retícula, en la cual se dibuja el patrón recibido por el puerto serial. Este patrón es presentado a la Madaline, y ésta lo asocia a la letra correspondiente.

5.3 Operación

Durante la operación del programa ENTRENA, lo único que tiene que hacer el usuario es monitorear el avance del entrenamiento. Al ejecutarse el programa, automáticamente se carga el conjunto de entrenamiento del archivo PATRONES.TXT. Posteriormente, se le pregunta al usuario si desea que se utilicen los pesos grabados en el disco, o si quiere iniciar el entrenamiento con pesos arbitrarios. Periodicamente, se graban los pesos de las Adalines. Durante el entrenamiento, cada vez que la Madaline aumenta la cantidad de patrones que ha aprendido, se envía un mensaje indicando el número del patrón hasta el cual llegó. Cuando llega hasta el patrón 528, el entrenamiento concluye. Una característica que tiene este programa es que está desarrollado a prueba de fallas de energía eléctrica. Es decir, cuando la máquina se apaga por alguna circunstancia; al encenderse de nuevo, el programa puede reiniciar automáticamente el entrenamiento, ya que, si después de cierto tiempo el usuario no responde a la pregunta de cargar los pesos, el programa asume que debe hacerlo y continúa con el entrenamiento utilizando los pesos que se tenían hasta antes de la última grabación.

Al ejecutar el programa ENVIA aparece una retícula de 5X4, en la cual se puede trazar cualquier figura. Existe la posibilidad de que se dibuje automáticamente la representación, descrita anteriormente, de cada letra; para ello basta presionar la letra deseada. Inicialmente el porcentaje de ruido es 0, de tal forma que al enviarse la figura trazada en la retícula, ésta se mostrará tal cual en la retícula presentada en la computadora receptora. Entonces, para introducir ruido de forma manual; únicamente se necesita cargar la configuración de algún patrón original, presionando la letra

correspondiente, e invertir los bits deseados, moviendose con las flechas del teclado y presionando la barra espaciadora en la posición del bit que se desea alterar. Se agregó la opción de seleccinar el porcentaje de ruido automático. El usuario puede seleccionar 4 niveles de ruido 0%, 5%, 10% y 15%, lo cual equivale a la alteración aleatoria de 0, 1, 2 y 3 bits respectivamente. Para enviar un patrón, se envía una cadena de 20 caracteres; cada uno representa un bit. Así, cuando se introduce ruido automaticamente, se invierten aleatoriamente la cantidad estipulada de bits antes de ser transmitidos.

El programa RECIBE, consta de 2 opciones principales. Una de ellas, consiste en mostrar los pesos de todas las Adalines de la red. La otra, permite esclavizar a la máquina para que reciba cualquier cadena de 20 caracteres mediante el puerto serial. Una vez que se reciben los 20 bits, se dibuja en la retícula el patrón construido. Posteriormente, dicho patrón se aplica a la Madaline y ésta lo asocia a una de las letras.

En el apéndice A se presenta una sesión de transmisión utilizando los programas ENVIA y RECIBE. También se muestran los listados de los programas creados en Turbo-Pascal 5.0 de Borland Inc.

Conclusiones

Aunque en la actualidad existen diferentes esquemas de entrenamiento para conseguir que los modelos neuronales resuelvan ciertas tareas específicas, algunos de los conceptos presentados por Minsky y Papert aún no han sido refutados satisfactoriamente. Uno de dichos conceptos es el *escalamiento*, el cual consiste en el hecho de que muchos de los problemas tratados mediante redes neuronales caen dentro de la categoría de problemas de juguete. Esto significa que son abstracciones simplicadas del mundo real, y si se desea extenderlos a representaciones estrictas de la realidad, se presentan dificultades con la cantidad de recursos, las estructuras y tipos de datos, y el consumo de tiempo necesarios para su entrenamiento y operación. A pesar de esto, el problema del escalamiento puede ser resuelto; ya que la prueba de ello es el funcionamiento del propio cerebro humano. Probablemente la solución se pueda encontrar cuando se tenga conocimiento más profundo y con mayor certeza a cerca de la actividad cerebral real.

Al tratar de comprender la necesidad de utilizar redes neuronales cuyas unidades tengan salidas digitales, primeramente se pensó que era debido a que éstas tenían ciertas vantajas sobre las que poseen salida analógica al implementarse en hardware. Motivo por el cual, se intentó presentar un análisis comparativo entre las neuronas con salida analógica y las neuronas con salida digital con respecto a tamaño, precisión, velocidad y disipación de potencia; sin embargo, al consultar la bibliografía, se encontró que no existe gran diferencia ya que para ambas implementaciones se puede

utilizar la tecnología CMOS/VLSI (Complementary Metal Oxide Semiconductor, Very Large Scale Integration). Dado que existen ciertas aplicaciones de redes neuronales como la compresión y transmisión instantánea de imágenes, en las cuales las salidas de las unidades de una capa deben viajar largas distancias al ser enviadas hacia las entradas de las unidades de la siguiente capa. En esta aplicación se debe considerar que las señales transmitidas se ven afectadas durante el viaje por factores como atenuación, debido a la resistencia del propio medio de transmisión; y distorsión provocada por el ruido del medio ambiente. Entonces, si se emplea una red de neuronas con salida analógica; la alteración sufrida por la señal recibida puede provocar graves problemas de inconsistencia en el sistema, dado que el valor de la señal recibida en la entrada *i* de la neurona *j* no es igual al valor de salida de la neurona *i*. En cambio, con un modelo de neuronas con salida digital; es posible emplear dispositivos de reajuste para eliminar cualquier alteración en la señal recibida, de tal manera que el valor de la señal en la entrada *i* de la neurona *j* es el mismo valor de salida de la neurona *i*. Por lo tanto, la justificación del uso de redes cuyas neuronas tengan salidas digitales radica en las necesidades de la misma aplicación.

Dado que en algunas aplicaciones se requiere utilizar redes de neuronas con salida digital como el caso de la Madaline, entonces es necesario desarrollar algoritmos para su entrenamiento. En base a lo anterior, se desarrolló el algoritmo de pastoreo para el entrenamiento de la Madaline como objetivo principal de esta tesis. La prueba del funcionamiento de dicho algoritmo y, por consiguiente, el cumplimiento del

objetivo principal, consiste en que se consiguió entrenar una Madaline que se utiliza en una aplicación de corrección automática de ruido.

La aplicación de la Madaline para la corrección automática de ruido, puede considerarse como un problema de tamaño regular ya que se debe entrenar la red para que aprenda 528 patrones de 20 bits. En el primer intento de entrenamiento, se utilizó una configuración de 20, 1000 y 5 Adalines en la capa de entrada, oculta y de salida respectivamente. Con esta estructura fue necesario codificar en binario un número asignado a cada letra (A = -1-1-1-11, B = -1-1-110, .., Z = 11-11-1), para que se considerara como la salida deseada para todas las representaciones de misma. Sin embargo, dado que se utilizó una máquina PS/2 Modelo 35 SX de IBM a 16 MHz sin coprocesador matemático, el entrenamiento avanzaba muy lentamente; al grado de, que después de una semana, la Madaline solo había aprendido aproximadamente 200 patrones. Entonces se modificó la estructura de la red adicionando una segunda capa oculta; sin embargo, el aprendizaje se hizo aún más lento. Un tercer intento consistió en utilizar 26 Adalines en la capa de salida para que en la salida deseada de cada patrón se encendiera (tomara un valor de 1) el bit correspondiente a la letra que representaba (A el primero, B el segundo y así sucesivamente), mientras el resto permanecian apagados (tomaran un valor de -1). Con este esquema de representación se pusieron a trabajar dos Madalines, la primera con una estructura de 20-1000-26 y la segunda con 20-600-26 Adalines. Después de aproximadamente 22 días, la segunda configuración concluyó el entrenamiento. La primer configuración terminó 5 días después.

No conforme con los resultados obtenidos en el tercer intento, después de analizar la posiblidad; se implementó una cuarta y definitiva configuración. En ésta se utiliza una Madaline de 20-600-26 unidades, pero se considera el criterio del bit de mayor orden, descrito en el capítulo 5, con el propósito de reducir el tiempo requerido para el entrenamiento. Con esta nueva configuración, la Madaline quedó entrenada en poco menos de 9 días después de 1251 iteraciones. Considerando que el número de iteraciones es pequeño, podemos pensar que el tiempo de entrenamiento puede reducirse considerablemente en una máquina con características poderosas.

Un grave problema que ocurrió durante los primero intentos de entrenamiento, fueron las fallas en la energía eléctrica. Dado que todas las pruebas tardaron días en realizarce; cada vez que se interrumpía la alimentación, era necesario reiniciar el entrenamiento. Ante tal situación, primeramente se conectó una fuente de poder ininterrumpible (no-break); sin embargo, cuando las fallas ocurrían durante la noche esta solución no era suficiente. Entonces se corrigió el programa ENTRENA de tal forma que periodicamente estuviera grabando los pesos de las Adalines. Así; cada vez que se interrumpía la energía y posteriormente se reestablecía, la máquina automáticamente reiniciaba el entrenamiento, perdiendo solo una pequeña cantidad del trabajo realizado.

Al realizar pruebas del comportamiento de la Madaline ya entrenada mediante el uso de los programas ENVIA y RECIBE, se observó que el poder de generalización de la red era mayor del esperado. En el conjunto de entrenamiento solo se agregaron, para cada letra, el patrón original y las 20 posibles alteraciones de 1 bit del mismo. Sin embargo, al modificar 2 bits

de cada patrón original; la Madaline lo asocia a la letra correcta aproximadamente un 70% de las veces. En el caso de la alteración de 3 bits, la red responde en forma correcta aproximadamente el 50% de las ocasiones.

Por último, se espera que este trabajo pueda servir a personas neófitas en el área, para formarse un concepto general de las redes neuronales; ya que en él se resumen la historia y fundamentos de las mismas, y se presenta una aplicación práctica que puede ser estimulante para despertar el interés de nuevas investigaciones en este campo. En lo que respecta al algoritmo de pastoreo, se probó con otras aplicaciones además de la que se presenta en esta tesis; y en todos los casos se consiguio el entrenamiento de la red. Sin embargo; se desea que otras personas se interesen en su análisis y validación, con el propósito de enriquecer el conocimiento en esta rama de la Inteligencia Artificial.

Referencias Bibliográficas

1.- Carpenter, G. A. and Grossbery, S. The art of adaptative patten recognition by a selft-organizing neural network. Computer. March, 1988. pp 77-88.

2.- Duncan, Ray. Advanced MS-DOS. Microsoft Press. 1986.

3.- Freeman, James A. and Skapura, David M.Neural networks: algoritms, applications, and programming techniques. Addison-Wesley Publishing Company. 1991.

4.- Hebb, Donald O. The organization of behavior. Wiley, New York, 1949.

5.- Hecht-Nielsen, Robert. Neurocomputing. Addison-Wesley Publishing Company. 1990.

6.- Hertz, John; Krogh, Anders and Palmer, Richard G. Introduction to the theory of neural computation. Addison-Wesley Publishing Company. 1991.

7.- Hopfield, J. J. Neural networks an physical systems with emergent colective computational abilities. Proceedings of the National Academy of Sciencies. USA 79. 2554 - 2558. 1982.

8.- Kruglinski, David. Guía a las comunicaciones del IBM/PC. OSBORNE/McGraw-Hill. 1985.

9.- McClelland, James and Rumel Hart, David. Explorations in parallel distributed processing. MIT Press. Cambridge, MA. 1986.

10.- McCulloch, Warren S. and Pitts, Walter. A logical calculus of the ideas immanent in nervous activity. Bolletin of Mathematical Biophysics, 5:115-133, 1943.

11.- Minsky, Marvin and Papert, Seymour. Perceptrons. MIT Press, Cambridge. MA. 1969.

12.- Parker, D. B. Learning logic technical report TR-47. Center for computational researchig in economics and management science. MIT. Cambridge. MA. April, 1985.

13.- Rosenblatt, Frank. The perceptron: a probabilistic model for information storage and organization in the brain. Psychological Review, 65:386-408. 1958.

14.- Wasserman, P. D. Neural computing theory and practice. Van Nostrand R. 1989.

15.- Werbos, P. Beyond regression: new tools for prediction and analysis in the behavioral sciences. PhD thesis. Harvard, Cambridge. Ma, august, 1974.

16.- Widrow, B. Generalization and information storage in networks of ADALINE neurons. G. T. [Ed]. Self-Organizing Systems. Spartan Books. Washington, DC. 1962.

17.- Widrow, B. and Hoff, M. E. Adaptative swithching circuits. WESCON Convention Record. New York, 1960 IRE. pp 96-104.

18.- Winter, Rondney and Widrow, Bernard. MADALINE RULE II: a training algorithm for neural networks. In proceeding of the IEEE *Second* International Conference of Neural Networks. San Diego, CA. I: 401-408. July, 1988.

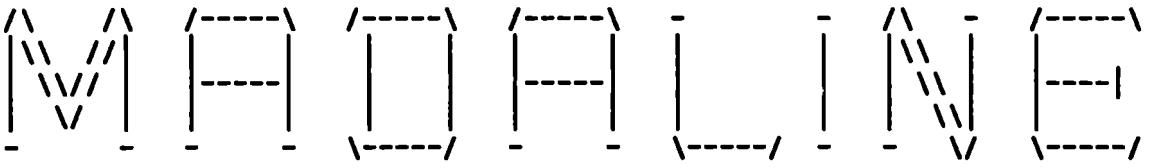
APÉNDICES

Apéndice A-i

Ejemplo de transmisión y corrección de caracteres

Con el propósito de mostrar los resultados obtenidos al entrenar la Madaline para la corrección automática de ruido, se presenta un ejemplo de ejecución de una sesión de transmisión entre los programas ENVIA y RECIBE:

Al ejecutar el programa ENVIA en la computadora transmisora, aparece la siguiente pantalla:

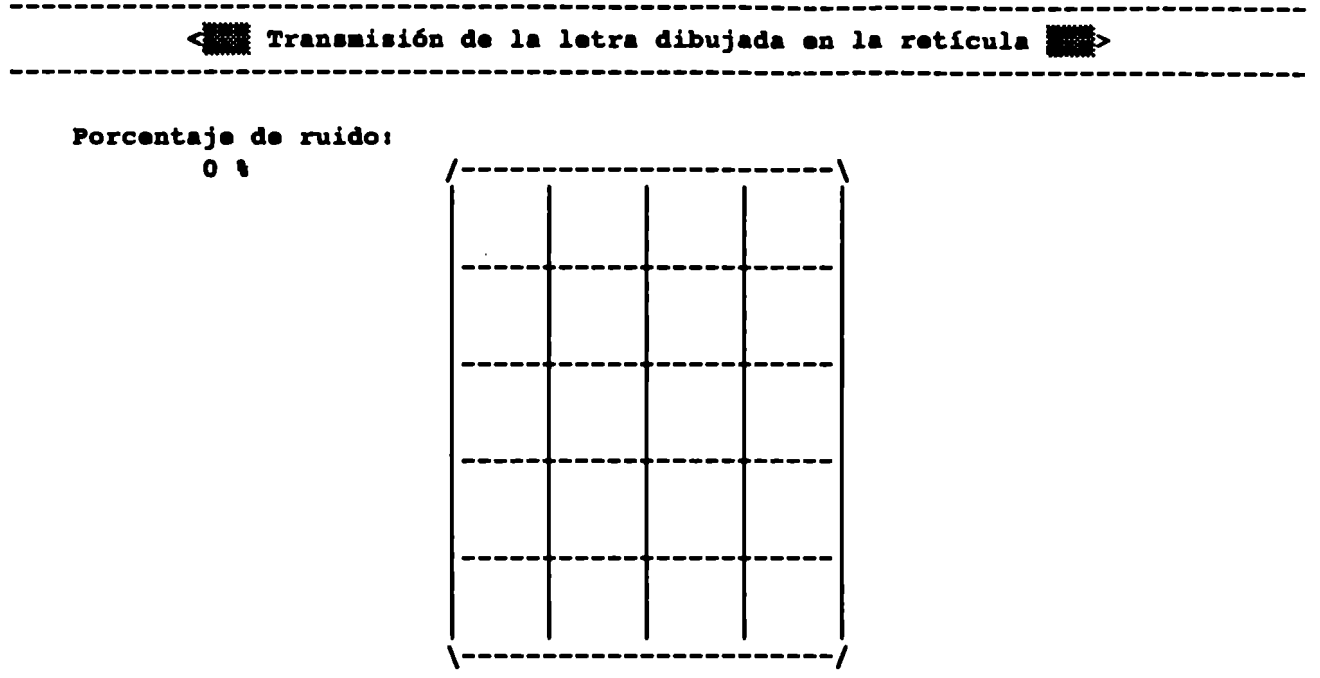


```
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
*                                                                               *
*          ▓▓▓▓ Genera diferentes representaciones de letras ▓▓▓▓                *
*                 y las transmite mediante el puerto serial                     *
*                                                                               *
*  Autor: Ing. Jesús Benjamín Rodríguez García   530946                         *
*  Maestría en Ciencias Computacionales                                         *
*  I.T.E.S.M.  Campus Estado de México                                          *
*                                                                               *
*                                                                               *
*                                                                               *
*                                                                               *
*              (Versión 1.0)              (c) Copyright Octubre de 1992.        *
*                                                                               *
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
```

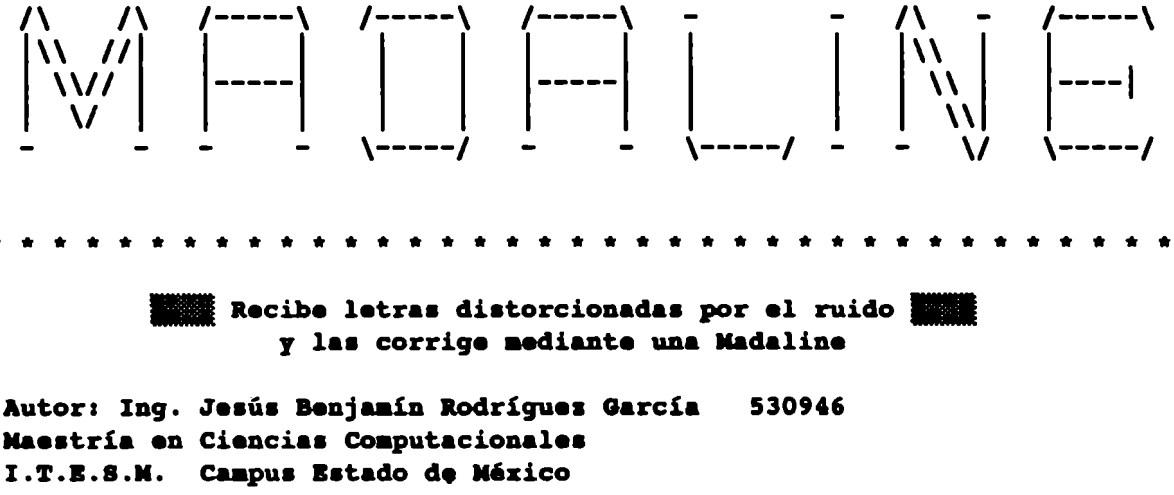
[Presione cualquier tecla para continuar]

Después de presionar una tecla aparecerá la siguiente pantalla de

trabajo:



Por otro lado, al ejecutarse el programa RECIBE en la computadora receptora, aparece la siguiente pantalla de presentación:



```
*                                                                       *
*                                                                       *
*                                                                       *
*          (Versión 1.0)            (c) Copyright Octubre de 1992.     *
*                                                                       *
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
```

[Presione cualquier tecla para continuar]

Posteriormente, se cargan los pesos de las Adalines de la capa oculta y de salida, que fueron grabados por el programa Entrena:

```
-> Cargando los pesos de las capas ocultas: ..................................
-> Número de Adalines en la capa oculta: 600

-> Cargando los pesos de las capas ocultas: ........................
-> Número de Adalines en la capa de salida: 26
```

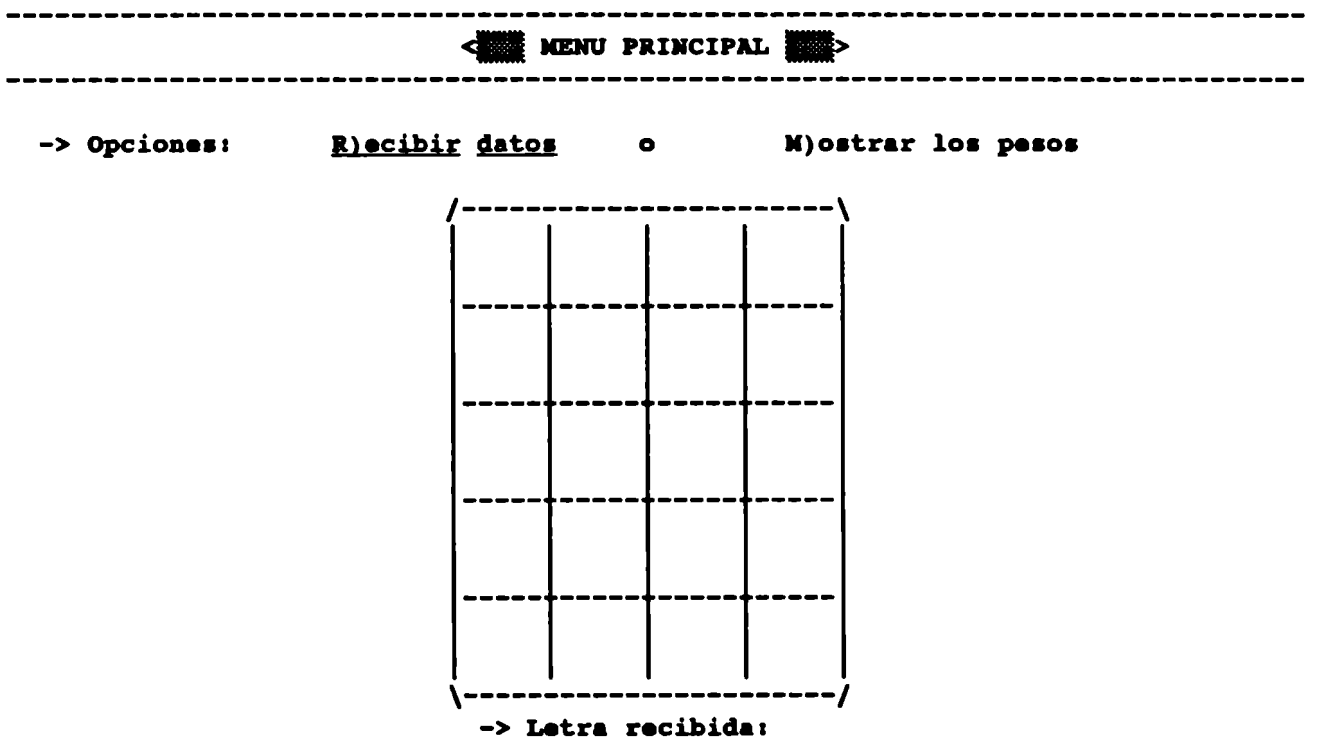
[Presione cualquier tecla para continuar]

Inmediatamente, después de presionar una tecla, aparece la pantalla principal:

```
------------------------------------------------------------------------------
                          <▓▓▓ MENU PRINCIPAL ▓▓▓>
------------------------------------------------------------------------------

  -> Opciones:      R)ecibir datos     o       M)ostrar los pesos




 [Esc] para terminar | Presione la letra de la opción deseada | I.T.E.S.M. CEM
```

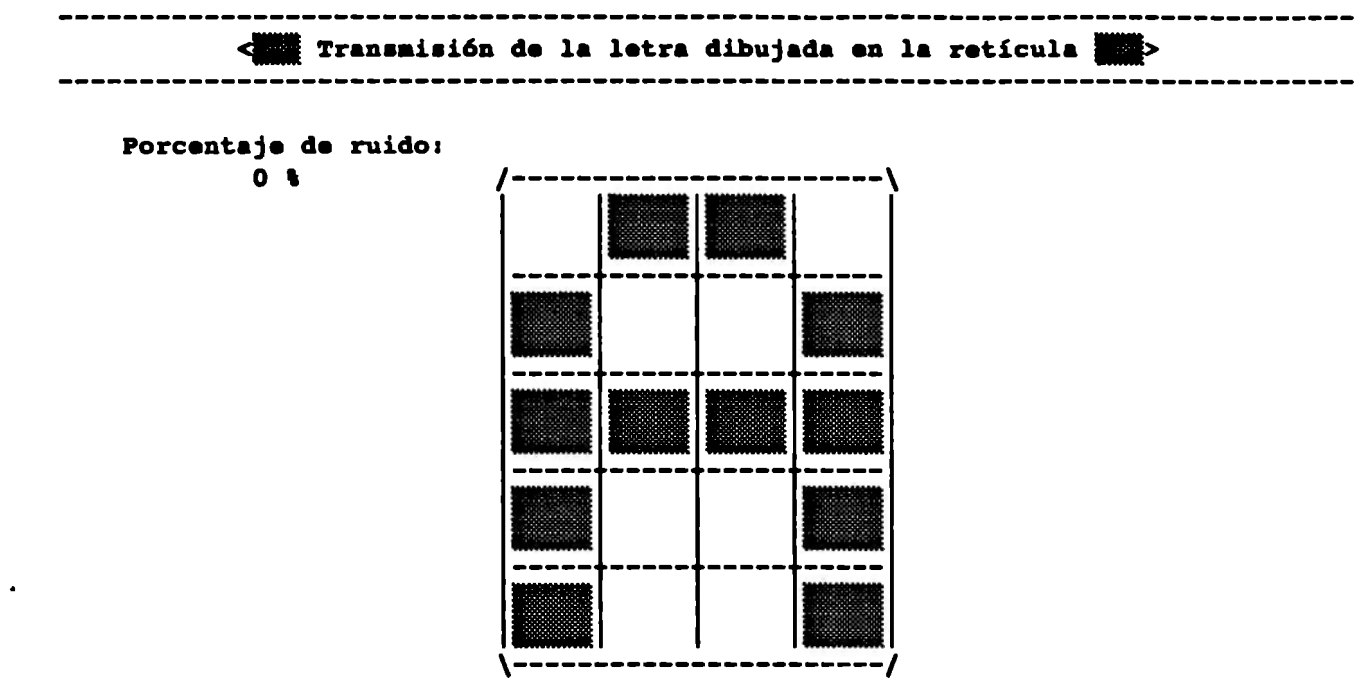
Si desea visualizar el valor de los pesos de la Madaline, presione la letra M. En esta opción se pregunta si desea el reporte por impresora, deberá contestar con una S para indicar que si, o una N para indicar que no. Cuando los pesos son mostrados por pantalla, cada vez que se llena una pantalla; se pide una tecla para continuar, si la tecla presionada es Esc se regresa al menú principal.

Al elegir recibir datos, presionando la letra R, se despliega la siguiente pantalla:

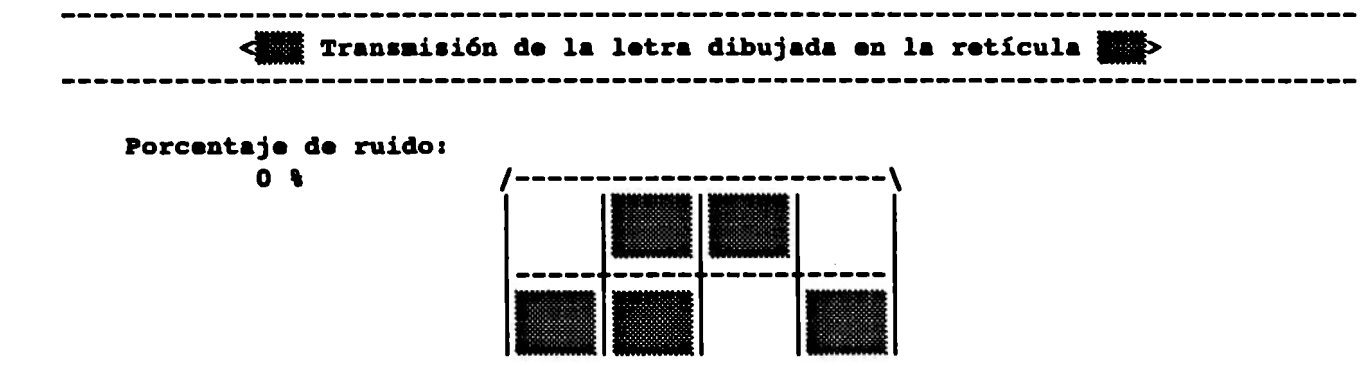


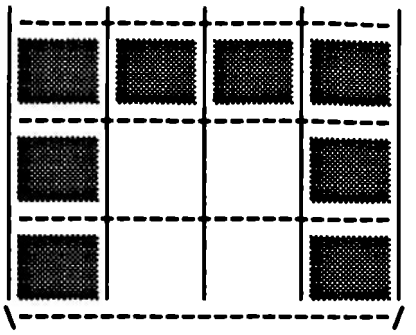
La máquina se quedará ciclada hasta que recibá, por el puerto serial, un patrón de 20 bits o se presione la tecla Esc en cuyo caso se regresará al menú principal.

Volviendo a la computadora transmisora, al presionar la letra A, aparecerá la representación original de dicha letra, tal como se muestra a continuación:



Enseguida, nos movemos, utilizando las flechas del teclado, hasta llegar al bit 6. Aquí presionamos la barra espaciadora y el bit se invierte, tal como se observa a continuación:



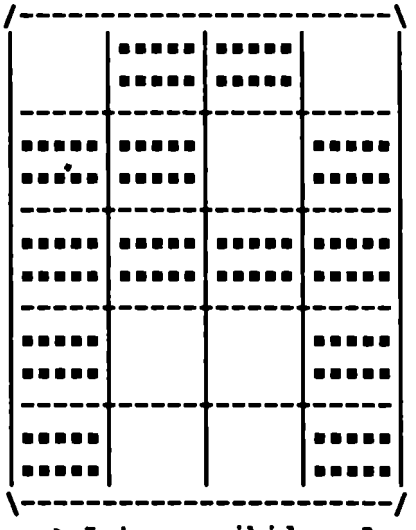


Moverse con: flechas [A..Z] Para seleccionar la representación de esa letra
[Esc] Termina [Espacio] Invierte [F8] Ruido [F9] Limpia [F10] Envía

Al presionar la tecla F10, la imagen dibujada en la retícula se transmite hacia la computadora receptora, la cual presenta el patrón recibido en la entrada de la Madaline y lo propaga hasta la salida. De esta forma, se asocia la imagen de la retícula a una letra. Ésto se puede apreciar enseguida:

<▓▓ MENU PRINCIPAL ▓▓>

-> Opciones: R)ecibir datos o M)ostrar los pesos



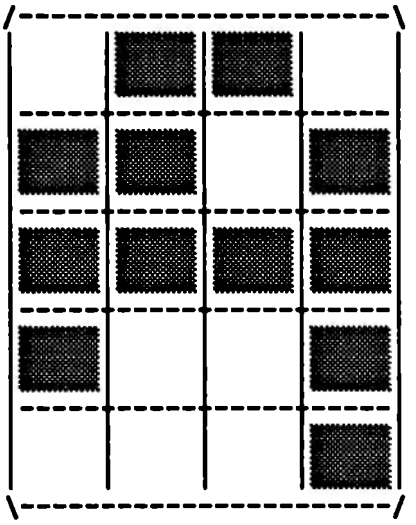
-> Letra recibida: A

[Esc] para terminar | Presione la letra de la opción deseada | I.T.E.S.M. CEM

Enseguida, nos movemos hasta el bit 17 y lo invertimos presionando la barra espaciadora, tal como se muestra a continuación:

<▓▓ Transmisión de la letra dibujada en la retícula ▓▓>

Porcentaje de ruido:
0 %

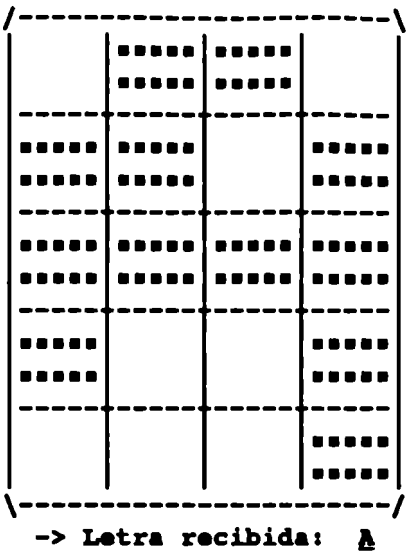


Moverse con: flechas [A..Z] Para seleccionar la representación de esa letra
[Esc] Termina [Espacio] Invierte [F8] Ruido [F9] Limpia [F10] Envía

Posteriormente, se envía con F10 y la Madaline la asocia de nuevo a la letra A.

<▓▓ MENU PRINCIPAL ▓▓>

-> Opciones: R)ecibir datos o M)ostrar los pesos



-> Letra recibida: A

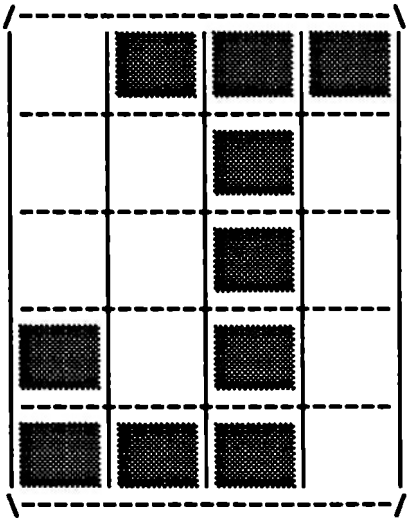
[Esc] para terminar | Presione la letra de la opción deseada | I.T.E.S.M. CEM

Como puede observarse, al lado izquierdo de la retícula se indica que el porcentaje de ruido es 0. Por ello, las imagenes de la retícula son recibidas integramente.

Ahora, presionamos la letra J y aparece su representación en la retícula:

<▓▓ Transmisión de la letra dibujada en la retícula ▓▓>

Porcentaje de ruido:
0 %

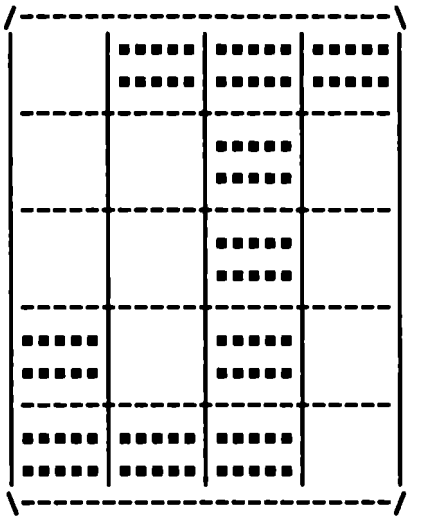


Moverse con: flechas [A..Z] Para seleccionar la representación de esa letra
[Esc] Termina [Espacio] Invierte [F8] Ruido [F9] Limpia [F10] Envía

Después de ser enviada, es reconocida perfectamente:

<▓▓ MENU PRINCIPAL ▓▓>

-> Opciones: R)ecibir datos o M)ostrar los pesos

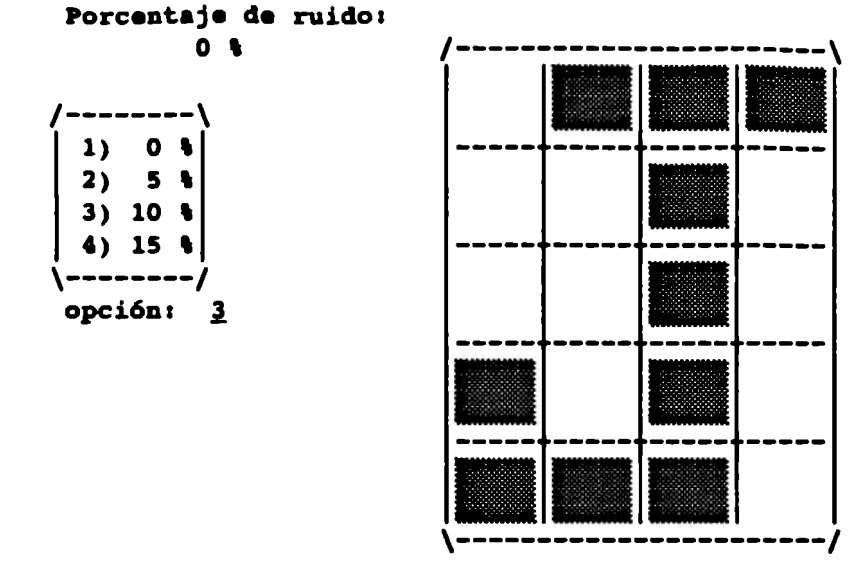


-> Letra recibida: J

[Esc] para terminar | Presione la letra de la opción deseada | I.T.E.S.M. CEM

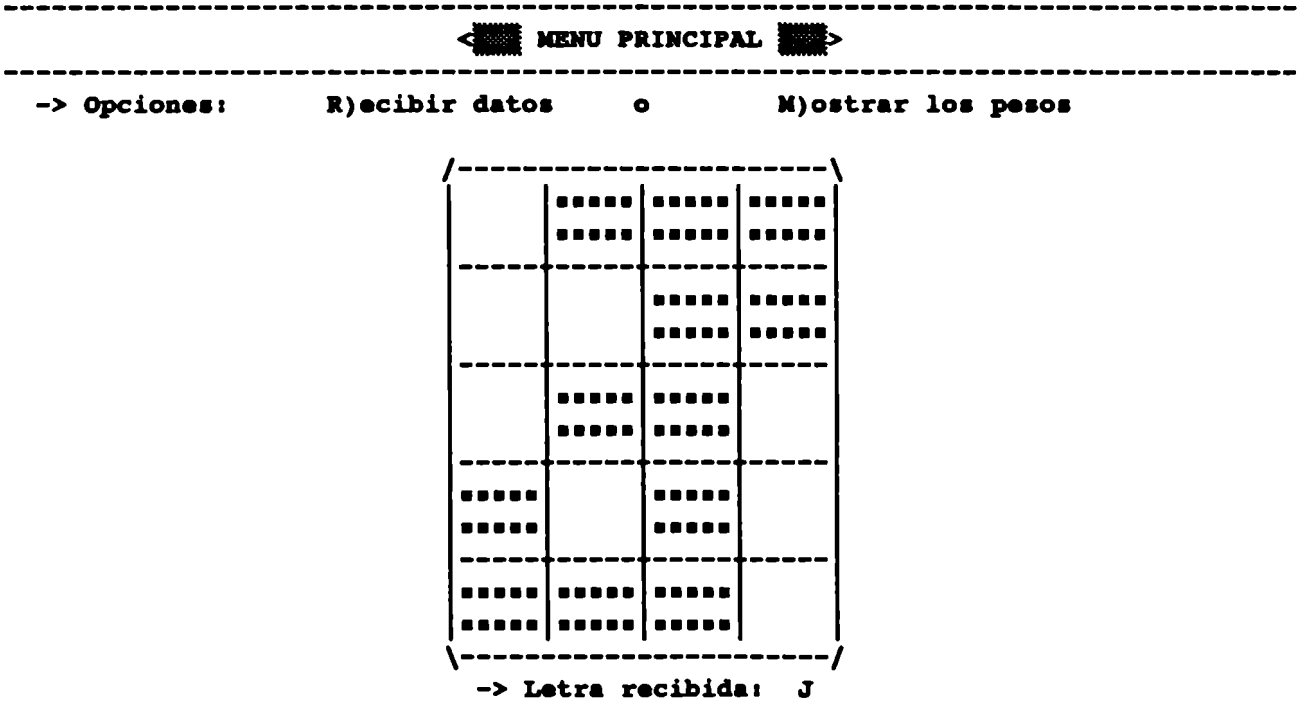
Posteriormente presionamos F8 para modificar el porcentaje de ruido, de tal forma que al transferirse la imagén, se distorsione al alterarse algunos bits:

<▓▓ Transmisión de la letra dibujada en la retícula ▓▓>



Moverse con: flechas [A..Z] Para seleccionar la representación de esa letra
[Esc] Termina [Espacio] Invierte [F8] Ruido [F9] Limpia [F10] Envía

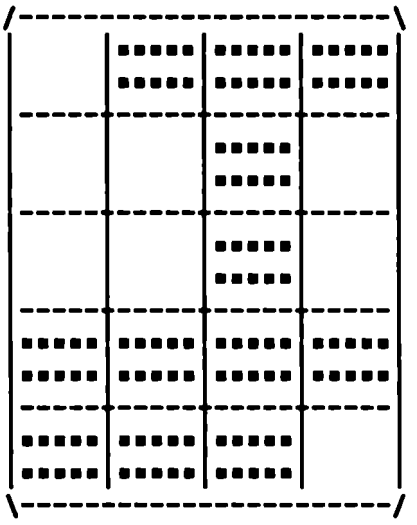
Dado que se seleccionó un 10% de ruido, cada vez que se transmita la imagen, se invertirán 2 bits. Ésto se puede apreciar en las dos siguientes recepciones:



[Esc] para terminar | Presione la letra de la opción deseada | I.T.E.S.M. CEM

<▓▓ MENU PRINCIPAL ▓▓>

-> Opciones: R)ecibir datos o M)ostrar los pesos



-> Letra recibida: J

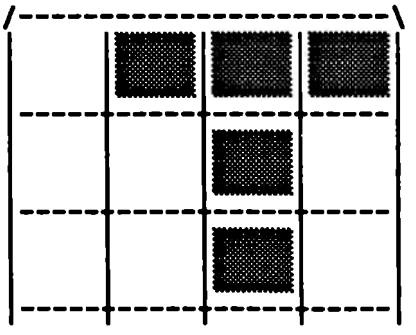
[Esc] para terminar | Presione la letra de la opción deseada | I.T.E.S.M. CEM

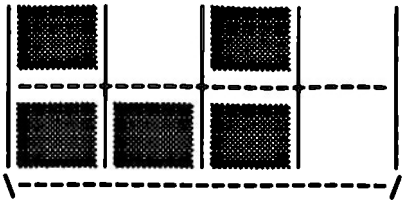
Enseguida, seleccionamos un 15% de ruido. Lo cual, significa que se alterarán 3 bits de cada patrón transmitido:

<▓▓ Transmisión de la letra dibujada en la retícula ▓▓>

Porcentaje de ruido:
10 %

```
/--------\
| 1)  0 %|
| 2)  5 %|
| 3) 10 %|
| 4) 15 %|
\--------/
 opción: 4
```





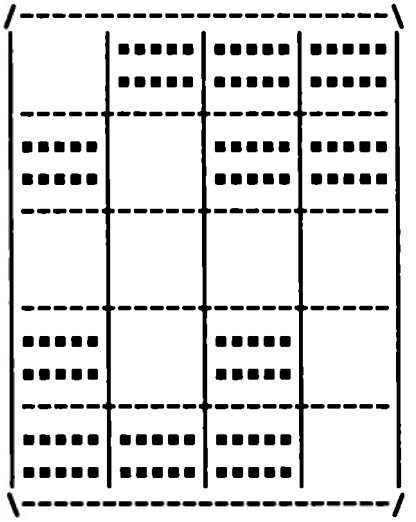
Moverse con: flechas [A..Z] Para seleccionar la representación de esa letra
[Esc] Termina [Espacio] Invierte [F8] Ruido [F9] Limpia [F10] Envía

Nótese, que en la pantalla aparece un 10% de ruido que es el que estaba establecido antes de aumentarlo al 15%.

A continuación se muestran tres casos de recepción. En los dos primeros, se asocia la letra correcta; mientras que en el tercero no es posible hacerlo, ya que la imagen recibida es más parecida a una letra Z.

<▓▓ MENU PRINCIPAL ▓▓>

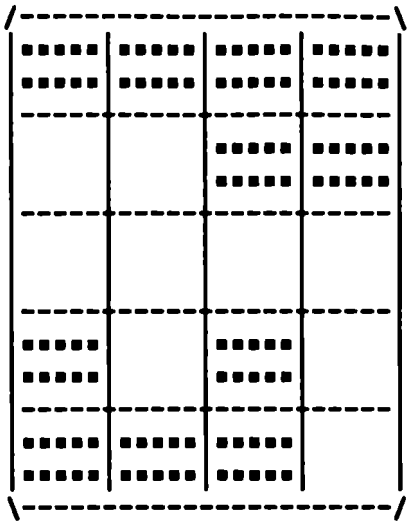
-> Opciones: R)ecibir datos o M)ostrar los pesos



-> Letra recibida: J

[Esc] para terminar | Presione la letra de la opción deseada | I.T.E.S.M. CEM

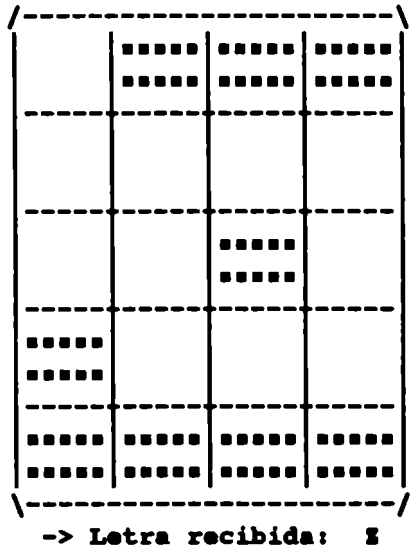
-> Opciones: R)ecibir datos o M)ostrar los pesos



-> Letra recibida: J

<▓▓ MENU PRINCIPAL ▓▓>

-> Opciones: R)ecibir datos o M)ostrar los pesos



-> Letra recibida: Z

Apéndice A-ii

Software de la implementación de la Madaline para corrección de ruido

```
(* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *)
(*                Implementación del algoritmo de pastoreo                     *)
(*                  para el entrenamiento de la Madaline                       *)
(*                                                                             *)
(*  Autor: Ing. Jesús Benjamín Rodríguez García   530946                       *)
(*  Maestría en Ciencias Computacionales                                       *)
(*  I.T.E.S.M.  Campus Estado de México                                        *)
(*                                                                             *)
(*                                                                             *)
(*  Archivo: ENTRENA.PAS              Lenguaje: Turbo Pascal 5.0 (Borland)     *)
(*                                                                             *)
(*            (Versión 1.0)             (c) Copyright Octubre de 1992.         *)
(* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *)

Program ENTRENA_MADALINE;

  uses
    crt,
    Libreria;

  const
    MaxTamConEnt = 900;
    MaxNumUni    = 80;

  type
    TipoAdaLiNe = record
                    Peso   : array [0..20] of real;
                    Salida : real;
                    Selec  : boolean;
                  end;

    TipoAdaLiNe2 = record
                     Peso   : array [0..600] of real;
                     Salida : real;
                     Selec  : boolean;
                   end;

    TipoOculta   = array [1..600] of ^TipoAdaLiNe;
    TipoSalida   = array [1..26] of ^TipoAdaLiNe2;

  var
    Entrenado     : boolean;
    TablaEntradas : array [1..MaxTamConEnt,1..20] of shortint;
    TablaSalidas  : array [1..MaxTamConEnt,1..26] of shortint;
    CapaOculta    : TipoOculta;
    CapaSalida    : TipoSalida;
    Entrada       : array [1..MaxNumUni] of shortint;
```

```
    TamConEnt     ,
    NumSalidas    ,
    NumOcultas    ,
    NumEntradas   : integer;
    op            : char;
    NumMaxIter    : longint;
    tope          : ^integer;
    nuevo         : boolean;
    archivo       : text;


(** Graba los pesos de las Adalines de la capa oculta y la capa de salida **)
  procedure Graba;
    var
      arch : file of TipoAdaLiNe;
      arch2 : file of TipoAdaLiNe2;
      i    : integer;
    begin
      assign(arch,'oculta.dat');
      rewrite(arch);
      assign(arch2,'salida.dat');
      rewrite(arch2);
      for i := 1 to NumOcultas do write(arch,CapaOculta[i]^);
      for i := 1 to NumSalidas do write(arch2,CapaSalida[i]^);
      close(arch);
      close(arch2);
    end;

(**** Propaga la entrada a través de toda la red hasta la capa de salida ****)
  procedure Propaga;
    var
      i, j : integer;
      s    : real;
    begin
      for i := 1 to NumOcultas do
        begin
          CapaOculta[i]^.Salida := CapaOculta[i]^.Peso[0];
          for j := 1 to NumEntradas do with CapaOculta[i]^ do
            Salida := Salida + Entrada[j] * Peso[j];
        end;
      for i := 1 to NumSalidas do
        begin
          CapaSalida[i]^.Salida := CapaSalida[i]^.Peso[0];
          for j := 1 to NumOcultas do with CapaSalida[i]^ do
            if CapaOculta[j]^.Salida >= 0 then Salida := Salida + Peso[j]
            else Salida := Salida - Peso[j];
        end;
    end;


(**** Propaga la capa oculta hasta la capa de salida ****)
  procedure PropOcul(uni : integer);
    var
      i, ns : integer;
```

```
    begin
      if CapaOculta[uni]^.salida >= 0 then ns := 1 else ns := -1;
      for i := 1 to NumSalidas do with CapaSalida[i]^ do
        Salida := Salida + (2 * ns * Peso[uni]);
    end;


(**** Entrena la red en base a un conjunto de entrenamiento previamente
     establecido ****)
  procedure EntrenaMAdaLiNe;
    var
      p, m, i, k, j, x, y, e, aux, sb, sb2, nsb, nsb2, ne, nc : integer;
      cambio, termina : boolean;
      inc, ns, sant : real;
      ni    : longint;
      orden : array [1..600] of integer;
      pesos : array [0..600] of real;
      l : char;
    begin
      if nuevo then
        begin
    {----- Asigna pesos arbitrarios -----}
          randomize;
          for i := 1 to NumOcultas do
            begin
              for j := 0 to NumEntradas do
                if random < 0.5 then CapaOculta[i]^.Peso[j] := - random
                else CapaOculta[i]^.Peso[j] := random;
              CapaOculta[i]^.Selec := false;
            end;
          for i := 1 to NumSalidas do
            begin
              for j := 0 to NumOcultas do
                if random < 0.5 then CapaSalida[i]^.Peso[j] := - random
                else CapaSalida[i]^.Peso[j] := random;
              CapaSalida[i]^.Selec := false;
            end;
        end;
      m  := 0;
      k  := 1;
      ni := 0;
      nc := 0;
      cambio := true;
      clrscr;
      window(1,2,80,25);
      repeat
        ni := ni + 1;
        write('-> Iteración: ',ni);
{----- Aplica un vector de entrenamiento -----}
        for i := 1 to NumEntradas do Entrada[i] := TablaEntradas[k,i];
{----- Propaga el vector de entrenamiento -----}
        Propaga;
{----- Calcula el número de errores en la salida -----}
        p := 0;
```

```
        j := NumSalidas;
        while (j >= 1) and (p = 0) do
          if TablaSalidas[k,j] = 1 then p := j else j := j - 1;
        e := 0;
        for j := p to NumSalidas do
          begin
            if (CapaSalida[j]^.Salida >= 0) and (TablaSalidas[k,j] <> 1) then e :=
            if (CapaSalida[j]^.Salida < 0) and (TablaSalidas[k,j] <> -1) then e :=
          end;
{----- Si existen errores se intentará un cambio de pesos -----}
        if e <> 0 then
          begin
            cambio := false;
{----- Ordena las neuronas de la capa oculta para modificar los pesos de
      acuerdo al criterio de "mínima disturbancia" -----}
            for j := 1 to NumOcultas do orden[j] := j;
            for i := 1 to NumOcultas - 1 do
              for j := i + 1 to NumOcultas do
                if abs(CapaOculta[orden[j]]^.Salida) < abs(CapaOculta[orden[i]]^.Sa
                  begin
                    aux := orden[i];
                    orden[i] := orden[j];
                    orden[j] := aux;
                  end;
{----- Modifica los pesos para cambiar las salidas de aquellas unidades de la
      capa oculta que aun no han sido modificadas ----}
            i := 1;
            while (e <> 0) and (i <= NumOcultas) do
              begin
                if not CapaOculta[orden[i]]^.Selec then
                  begin
                    if CapaOculta[orden[i]]^.Salida >= 0 then sb := 1 else sb := -1
                    inc := 0.05;
                    for j := 0 to NumEntradas do pesos[j] := CapaOculta[orden[i]]^.
                    repeat
                      if sb = 1 then
                        CapaOculta[orden[i]]^.Peso[0] := CapaOculta[orden[i]]^.Peso
                      else
                        CapaOculta[orden[i]]^.Peso[0] := CapaOculta[orden[i]]^.Peso
                      for j := 1 to NumEntradas do
                        if (Entrada[j] = 1) and (sb = 1) then
                          CapaOculta[orden[i]]^.Peso[j] := CapaOculta[orden[i]]^.Pe
                        else
                          if (Entrada[j] = 1) and (sb = -1) then
                            CapaOculta[orden[i]]^.Peso[j] := CapaOculta[orden[i]]^.
                          else
                            if (Entrada[j] = -1) and (sb = 1) then
                              CapaOculta[orden[i]]^.Peso[j] := CapaOculta[orden[i]]
                            else CapaOculta[orden[i]]^.Peso[j] := CapaOculta[orden[
                      ns := CapaOculta[orden[i]]^.Peso[0];
                      for j := 1 to NumEntradas do ns := ns + Entrada[j] * CapaOcul
                      if ns >= 0 then nsb := 1 else nsb := -1;
                    until nsb <> sb;
                    sant := CapaOculta[orden[i]]^.Salida;
```

```
                    CapaOculta[orden[i]]^.Salida := ns;
                    PropOcul(orden[i]);
                    ne := 0;
                    for j := p to NumSalidas do
                      begin
                        if (CapaSalida[j]^.Salida >= 0) and (TablaSalidas[k,j] <> 1
                        if (CapaSalida[j]^.Salida < 0) and (TablaSalidas[k,j] <> -1
                      end;
    {----- Si no se redujo el número de errores con el cambio de salida, entonces
          se restauran los pesos anteriores; en caso contrario se marca la unidad
          como previamente seleccionada y se actualiza el numero de errores -----}
                    if ne >= e then
                      begin
                        for j := 0 to NumEntradas do CapaOculta[orden[i]]^.Peso[j]
                        CapaOculta[orden[i]]^.Salida := sant;
                        PropOcul(orden[i]);
                      end
                    else
                      begin
                        e := ne;
                        CapaOculta[orden[i]]^.Selec := true;
                        cambio := true;
                      end;
                  end;
                i := i + 1;
              end;
{----- Modifica los pesos para cambiar las salidas de aquellas unidades de la
      capa de salida que aun no han sido modificadas y que son diferentes a la
      salida deseada ----}
            for i := p to NumSalidas do if not CapaSalida[i]^.Selec then
              begin
                if CapaSalida[i]^.Salida >= 0 then sb := 1 else sb := -1;
                if sb <> TablaSalidas[k,i] then
                  begin
                    inc := 0.05;
                    repeat
                      if sb = 1 then
                        CapaSalida[i]^.Peso[0] := CapaSalida[i]^.Peso[0] - inc
                      else
                        CapaSalida[i]^.Peso[0] := CapaSalida[i]^.Peso[0] + inc;
                      for j := 1 to NumOcultas do
                        begin
                          if (CapaOculta[j]^.Salida >= 0) and (sb = 1) then
                            CapaSalida[i]^.Peso[j] := CapaSalida[i]^.Peso[j] - inc
                          else
                            if (CapaOculta[j]^.Salida >= 0) and (sb = -1) then
                              CapaSalida[i]^.Peso[j] := CapaSalida[i]^.Peso[j] + in
                            else
                              if (CapaOculta[j]^.Salida < 0) and (sb = 1) then
                                CapaSalida[i]^.Peso[j] := CapaSalida[i]^.Peso[j] +
                              else CapaSalida[i]^.Peso[j] := CapaSalida[i]^.Peso[j]
                        end;
                      ns := CapaSalida[i]^.Peso[0];
                      for j := 1 to NumOcultas do
```

```
                    if CapaOculta[j]^.Salida >= 0 then ns := ns + CapaSalida[i]^
                    else ns := ns - CapaSalida[i]^.Peso[j];
                  if ns >= 0 then nsb := 1 else nsb := - 1;
                until nsb <> sb;
                CapaSalida[i]^.Salida := ns;
                CapaSalida[i]^.Selec := true;
                cambio := true;
              end;
          end;
      end;
{----- Si no se puedo efectuar algún cambio en las unidades que no han sido
      modificadas, de tal forma que se reduzca el número de errores, entonces
      es necesario desmarcar como modificadas todas las unidades -----}
    if not cambio then
      begin
        nc := nc + 1;
        if nc <= 300 then
          begin
            for i := 1 to NumOcultas do CapaOculta[i]^.Selec := false;
            for i := 1 to NumSalidas do CapaSalida[i]^.Selec := false;
          end
        else
{----- En caso de que hayan desmarcado más de 30 veces las unidades, es
      necesario nuevamente partir de valores aleatorios en los pesos -----}
          begin
            for i := 1 to NumOcultas do
              begin
                for j := 0 to NumEntradas do
                  if random < 0.5 then CapaOculta[i]^.Peso[j] := - random
                  else CapaOculta[i]^.Peso[j] := random;
                CapaOculta[i]^.Selec := false;
              end;
            for i := 1 to NumSalidas do
              begin
                for j := 0 to NumOcultas do
                  if random < 0.5 then CapaSalida[i]^.Peso[j] := - random
                  else CapaSalida[i]^.Peso[j] := random;
                CapaSalida[i]^.Selec := false;
              end;
            nc := 0;
          end;
      end;
{----- Se prueba con los pesos actuales todos los patrones del conjunto de
      entrenamiento, en caso de que uno falle, entoces se continua con las
      modificaciones -----}
    termina := true;
    i := 1;
    while (termina) and (i <= TamConEnt) do
      begin
        for j := 1 to NumEntradas do Entrada[j] := TablaEntradas[i,j];
        Propaga;
        p := 0;
        j := NumSalidas;
        while (j >= 1) and (p = 0) do
```

```
            if TablaSalidas[i,j] = 1 then p := j else j := j - 1;
          j := p;
          while (j <= NumSalidas) and (termina) do
            begin
              if (CapaSalida[j]^.Salida >= 0) and (TablaSalidas[i,j] <> 1) then t
              if (CapaSalida[j]^.Salida < 0) and (TablaSalidas[i,j] <> -1) then T
              if termina then j := j + 1;
            end;
          if termina then i := i + 1
          else
            begin
              if i > m then
                begin
                  window(1,1,80,25);
                  clrscr;
                  gotoxy(1,1);
                  write('Mayor patrón = ',i,' en la iteración ',ni);
                  m := i;
                  window(1,2,80,25);
                end;
              writeln(' k = ',k,' falló en el patrón ',i);
            end;
        end;
      if not termina then k := i;
      if keypressed then
        begin
          l := upcase(readkey);
          if l = 'G' then Graba;
        end;
      if (ni mod 20) = 0 then Graba;
    until (termina) or (ni = NumMaxIter);
    window(1,1,80,25);
    clrscr;
    Entrenado := termina;
  end;

(**** Inicializa los parámetros necesarios en el entrenamiento ****)
  procedure Inicializa;
    var
      i : integer;
    begin
      NumMaxIter  := 100000;
      Entrenado   := false;
      TamConEnt   := 0;
      NumEntradas := 20;
      NumSalidas  := 26;
      NumOcultas  := 600;
      mark(tope);
      for i := 1 to 600 do new(CapaOculta[i]);
      for i := 1 to 26 do new(CapaSalida[i]);
      nuevo := true;
      normal;
    end;
```

```
procedure Presenta;
  begin
    clrscr;
    gotoxy(01,25);
    writeln('     /\    /\   /-----\   /-----\   /-----\   -       -   /\   -
    writeln('     |\\  //|   |     |   |     |   |     |   |       |   |\\  |
    writeln('     | \\// |   |-----|   |     |   |-----|   |       |   | \\ |
    writeln('     |  \/  |   |     |   |     |   |     |   |       |   |  \\|
    writeln('     -      -   -     -   \-----/   -     -   \-----/   -   -   \/
    writeln('
    writeln;
    writeln(' * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
    writeln(' *
    writeln(' *           ▓▓▓▓ Implementación del algoritmo de pastoreo ▓▓▓▓
    writeln(' *                  para el entrenamiento de la Madaline
    writeln(' *
    writeln(' *  Autor: Ing. Jesús Benjamín Rodríguez García   530946
    writeln(' *  Maestría en Ciencias Computacionales
    writeln(' *  I.T.E.S.M.  Campus Estado de México
    writeln(' *
    writeln(' *
    writeln(' *
    writeln(' *             (Versión 1.0)             (c) Copyright Octubre de 199
    writeln(' *
    writeln(' * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
    writeln;
    writeln;
    pausa(19,25);
  end;

procedure CargaPesos;
  var
    arch  : file of TipoAdaLiNe;
    arch2 : file of TipoAdaLiNe2;
    a     : TipoAdaLiNe;
    a2    : TipoAdaLiNe2;
  begin
    clrscr;
    assign(arch,'oculta.dat');
    reset(arch);
    assign(arch2,'salida.dat');
    NumOcultas := 0;
    write('-> Cargando los pesos de las capas ocultas: ');
    while not eof(arch) do
      begin
        NumOcultas := NumOcultas + 1;
        read(arch,a);
        CapaOculta[NumOcultas]^ := a;
        if (NumOcultas mod 10) = 0 then write('.');
      end;
    writeln;
    writeln('-> Número de Adalines en la capa oculta: ',NumOcultas);
    writeln;
    close(arch);
```

```
    reset(arch2);
    NumSalidas := 0;
    write('-> Cargando los pesos de las capas ocultas: ');
    while not eof(arch2) do
      begin
        NumSalidas := NumSalidas + 1;
        read(arch2,a2);
        CapaSalida[NumSalidas]^ := a2;
        write('.');
      end;
    writeln;
    writeln('-> Número de Adalines en la capa de salida: ',NumSalidas);
    close(arch2);
    delay(5000);
  end;

procedure Carga;
  var
    s     : string[40];
    i, j  : integer;
    la, d : char;
    c     : longint;
  begin
    clrscr;
    writeln('-> Cargando el conjunto de entrenamiento:');
    window(1,2,80,25);
    assign(archivo,'patrones.txt');
    reset(archivo);
    la := chr(0);
    i  := 0;
    while not eof(archivo) do
      begin
        readln(archivo,s);
        if s[1] <> la then
          begin
            la := s[1];
            i := i + 1;
          end;
        TamConEnt := TamConEnt + 1;
        write(tamconent,'.- ',s[1],'    ');
        for j := 1 to NumEntradas do
          begin
            if s[j + 2] = '0' then TablaEntradas[TamConEnt,j] := -1
            else TablaEntradas[TamConEnt,j] := 1;
            write(TablaEntradas[TamConEnt,j]:2);
          end;
        writeln('    ',i);
        for j := 1 to i do TablaSalidas[TamConEnt,j] := 1;
        for j := i + 1 to NumSalidas do TablaSalidas[TamConEnt,j] := -1;
      end;
    close(archivo);
    window(1,1,80,25);
    clrscr;
    write('¿Cargar pesos? (S/N): ');
```

```
      c := 1;
      d := 'S';
      repeat
        if keypressed then
          begin
            d := upcase(readkey);
            c := 100000;
          end
        else c := c + 1;
      until c = 100000;
      writeln(d);
      if d = 'S' then
        begin
          CargaPesos;
          nuevo := false;
        end;
      EntrenaMAdaLiNe;
      if not Entrenado then
        writeln('   *** No fue posible entrenar el MAdaLiNe después de ',NumMaxIter,
      else
        begin
          writeln('            *** Entrenamiento concluido satisfactoriamente ***');
          graba;
        end;
      pausa(20,wherey);
    end;

  procedure Termina;
    var
      i : integer;
    begin
      release(tope);
      for i := 1 to 40 do
        begin
          marco3(i,1,81 - i,25);
          sound(i * 50);
          delay(2);
          nosound;
        end;
      clrscr;
      writeln('Hasta la próxima ...');
    end;


(**** Cuerpo principal del programa ****)
  begin
    Inicializa;
    Presenta;
    Carga;
    Termina;
  end.
```

```
(* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *)
(*      Transmite diferentes representaciones de letras en una retícula      *)
(*                                                                           *)
(*  Autor: Ing. Jesús Benjamín Rodríguez García   530946                     *)
(*  Maestría en Ciencias Computacionales                                     *)
(*  I.T.E.S.M.  Campus Estado de México                                      *)
(*                                                                           *)
(*                                                                           *)
(*  Archivo: ENVIA.PAS                Lenguaje: Turbo Pascal 5.0 (Borland)   *)
(*                                                                           *)
(*            (Versión 1.0)             (c) Copyright Octubre de 1992.       *)
(* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *)

Program ENVIA;

  uses
    Crt,
    Dos,
    Libreria;

  const
    PatOriginal : array [1..26,1..20] of shortint = (
      (-1, 1, 1,-1, 1,-1,-1, 1, 1, 1, 1, 1, 1,-1,-1, 1, 1,-1,-1, 1),
      ( 1, 1, 1,-1, 1,-1,-1, 1, 1, 1, 1,-1, 1,-1,-1, 1, 1, 1, 1,-1),
      (-1, 1, 1,-1, 1,-1,-1, 1, 1,-1,-1,-1, 1,-1,-1, 1,-1, 1, 1,-1),
      ( 1, 1, 1,-1, 1,-1,-1, 1, 1,-1,-1, 1, 1,-1,-1, 1, 1, 1, 1,-1),
      ( 1, 1, 1, 1, 1,-1,-1,-1, 1, 1, 1, 1, 1,-1,-1,-1, 1, 1, 1, 1),
      ( 1, 1, 1, 1, 1,-1,-1,-1, 1, 1, 1, 1, 1,-1,-1,-1, 1,-1,-1,-1),
      ( 1, 1, 1, 1, 1,-1,-1, 1, 1,-1,-1,-1, 1,-1, 1, 1, 1, 1, 1, 1),
      ( 1,-1,-1, 1, 1,-1,-1, 1, 1, 1, 1, 1, 1,-1,-1, 1, 1,-1,-1, 1),
      (-1, 1, 1,-1,-1, 1, 1,-1,-1, 1, 1,-1,-1, 1, 1,-1,-1, 1, 1,-1),
      (-1, 1, 1, 1,-1,-1, 1,-1,-1,-1, 1,-1, 1,-1, 1,-1, 1, 1, 1,-1),
      ( 1,-1,-1, 1, 1,-1, 1,-1, 1, 1,-1,-1, 1,-1, 1,-1, 1,-1,-1, 1),
      ( 1,-1,-1,-1, 1,-1,-1,-1, 1,-1,-1,-1, 1,-1,-1,-1, 1, 1, 1, 1),
      ( 1,-1,-1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,-1,-1, 1, 1,-1,-1, 1),
      ( 1,-1,-1, 1, 1, 1,-1, 1, 1, 1,-1, 1, 1,-1, 1, 1, 1,-1, 1, 1),
      ( 1, 1, 1, 1, 1,-1,-1, 1, 1,-1,-1, 1, 1,-1,-1, 1, 1, 1, 1, 1),
      ( 1, 1, 1,-1, 1,-1,-1, 1, 1, 1, 1,-1, 1,-1,-1,-1, 1,-1,-1,-1),
      (-1, 1, 1,-1, 1,-1,-1, 1, 1,-1,-1, 1, 1,-1, 1, 1,-1, 1, 1, 1),
      ( 1, 1, 1,-1, 1,-1,-1, 1, 1, 1, 1,-1, 1,-1,-1, 1,-1,-1, 1, 1),
      (-1, 1, 1, 1, 1,-1,-1,-1,-1, 1, 1,-1,-1,-1,-1, 1, 1, 1, 1,-1),
      ( 1, 1, 1, 1,-1, 1, 1,-1,-1, 1, 1,-1,-1, 1, 1,-1,-1, 1, 1,-1),
      ( 1,-1,-1, 1, 1,-1,-1, 1, 1,-1,-1, 1, 1,-1,-1, 1, 1, 1, 1, 1),
      ( 1,-1,-1, 1, 1,-1,-1, 1, 1,-1,-1, 1, 1,-1,-1, 1,-1, 1, 1,-1),
      ( 1,-1,-1, 1, 1,-1,-1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,-1,-1, 1),
      ( 1,-1,-1, 1, 1,-1,-1, 1,-1, 1, 1,-1, 1,-1,-1, 1, 1,-1,-1, 1),
      ( 1,-1,-1, 1, 1,-1,-1, 1,-1, 1, 1,-1,-1, 1, 1,-1,-1, 1, 1,-1),
      ( 1, 1, 1, 1,-1,-1,-1, 1,-1, 1, 1,-1, 1,-1,-1,-1, 1, 1, 1, 1));

  var
    Patron     : array [1..20] of shortint;
    np, nb, pr : integer;
    op         : char;
    params     : registers;
```

```
  procedure Despliega;
    var
      i, l , c : integer;
    begin
      escribe(28, 6,'/----------------------\');
      escribe(28, 7,'|      |      |      |      |');
      escribe(28, 8,'|      |      |      |      |');
      escribe(28, 9,'|------+------+------+------|');
      escribe(28,10,'|      |      |      |      |');
      escribe(28,11,'|      |      |      |      |');
      escribe(28,12,'|------+------+------+------|');
      escribe(28,13,'|      |      |      |      |');
      escribe(28,14,'|      |      |      |      |');
      escribe(28,15,'|------+------+------+------|');
      escribe(28,16,'|      |      |      |      |');
      escribe(28,17,'|      |      |      |      |');
      escribe(28,18,'|------+------+------+------|');
      escribe(28,19,'|      |      |      |      |');
      escribe(28,20,'|      |      |      |      |');
      escribe(28,21,'\----------------------/');
      textcolor(11);
      for i := 1 to 20 do if Patron[i] = 1 then
        begin
          l := 7 + trunc(3 * int((i - 1) / 4));
          c := 29 + trunc(6 * ((i - 1) mod 4));
          gotoxy(c,l);
          write('▓▓▓▓▓');
          gotoxy(c,l + 1);
          write('▓▓▓▓▓');
        end;
      normal;
    end;

(**** Inicializa los parámetros necesarios en el entrenamiento ****)
  procedure Inicializa;
    var
      i : integer;
    begin
      for i := 1 to 20 do Patron[i] := -1;
      np := 0;
      nb := 1;
      pr := 0;
      normal;
    end;

  procedure Presenta;
    begin
      clrscr;
      gotoxy(01,25);
      writeln('     /\    /\   /-----\   /-----\   /-----\   -         -   /\   -
      writeln('     |\\  //|   |     |   |     |   |     |   |         |   |\\  |
      writeln('     | \\// |   |-----|   |     |   |-----|   |         |   | \\ |
```

```
    writeln('     | \/ |  |     |   |    |  |     |  |      |  |  \\|
    writeln('     -     -  -     -   \-----/  -     -  \-----/  -  -   \/
    writeln('
    writeln;
    writeln(' * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
    writeln(' *
    writeln(' *          ▓▓ Genera diferentes representaciones de letras ▓▓
    writeln(' *               y las transmite mediante el puerto serial
    writeln(' *
    writeln(' *  Autor: Ing. Jesús Benjamín Rodríguez García   530946
    writeln(' *  Maestría en Ciencias Computacionales
    writeln(' *  I.T.E.S.M.  Campus Estado de México
    writeln(' *
    writeln(' *
    writeln(' *
    writeln(' *            (Versión 1.0)          (c) Copyright Octubre de 199
    writeln(' *
    writeln(' * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
    writeln;
    writeln;
    pausa(19,25);
    clrscr;
    escribe( 1, 1,unstr('-',80));
    escribe( 1, 2,'           <▓ Transmisión de la letra dibujada en la retícul
    escribe( 1, 3,unstr('-',80));
    escribe(28, 6,'/--------------------\');
    escribe(28, 7,'|    |    |    |    |');
    escribe(28, 8,'|    |    |    |    |');
    escribe(28, 9,'|----+----+----+----|');
    escribe(28,10,'|    |    |    |    |');
    escribe(28,11,'|    |    |    |    |');
    escribe(28,12,'|----+----+----+----|');
    escribe(28,13,'|    |    |    |    |');
    escribe(28,14,'|    |    |    |    |');
    escribe(28,15,'|----+----+----+----|');
    escribe(28,16,'|    |    |    |    |');
    escribe(28,17,'|    |    |    |    |');
    escribe(28,18,'|----+----+----+----|');
    escribe(28,19,'|    |    |    |    |');
    escribe(28,20,'|    |    |    |    |');
    escribe(28,21,'\--------------------/');
    escrinv( 1,24,' Moverse con:        [A..Z] Para seleccionar la representaci
    escrinv( 1,25,'  [Esc] Termina   [Espacio] Invierte   [F8] Ruido   [F9] Limpi
    escrinv(15,24,#27 + #24 + #25 + #26);
  end;

procedure Termina;
  var
    i : integer;
  begin
    for i := 1 to 40 do
      begin
        marco1(i,1,81 - i,25);
        sound(i * 50);
```

```
          delay(2);
          nosound;
        end;
      clrscr;
      writeln('Hasta la próxima ...');
    end;

  procedure Posiciona;
    var
      l , c : integer;
    begin
      l := 7 + trunc(3 * int((nb - 1) / 4));
      c := 29 + trunc(6 * ((nb - 1) mod 4));
      gotoxy(c,l);
    end;

  procedure Menu;
    begin
      escribe(5,5,'Porcentaje de riudo: ');
      case pr of
        0 : escribe(12,6,' 0 %');
        1 : escribe(12,6,' 5 %');
        2 : escribe(12,6,'10 %');
        3 : escribe(12,6,'15 %');
      end;
      Posiciona;
      repeat
        op := upcase(readkey);
      until (op in ['A'..'Z']) or (scan in [57,1,75,72,80,77,66,67,68]);
    end;

(*** Asigna la representación de una letra al patrón actual ***)
  procedure Representa;
    var
      i : integer;
    begin
      nb := 1;
      np := ord(op) - 64;
      for i := 1 to 20 do Patron[i] := PatOriginal[np,i];
      Despliega;
    end;

(*** Invierte el bit actual ***)
  procedure Invierte;
    var
      l , c : integer;
    begin
      l := 7 + trunc(3 * int((nb - 1) / 4));
      c := 29 + trunc(6 * ((nb - 1) mod 4));
      if Patron[nb] = 1 then
        begin
          Patron[nb] := -1;
          gotoxy(c,l);
          write('     ');
```

```
          gotoxy(c,l + 1);
          write('     ');
        end
      else
        begin
          textcolor(11);
          Patron[nb] := 1;
          gotoxy(c,l);
          write('▓▓▓▓▓');
          gotoxy(c,l + 1);
          write('▓▓▓▓▓');
          normal;
        end;
    end;

(*** Mueve el cursor a un nuevo bit ***)
  procedure Mueve;
    begin
      case scan of
        75 : if nb > 1  then nb := nb - 1;
        72 : if nb > 4  then nb := nb - 4;
        80 : if nb < 17 then nb := nb + 4;
        77 : if nb < 20 then nb := nb + 1;
      end;
    end;

(*** Captura el porcentaje de ruido ***)
  procedure Ruido;
    var
      c : char;
    begin
      window(4,8,19,15);
      marco2(1,1,10,6);
      escribe(3,2,'1)  0 %');
      escribe(3,3,'2)  5 %');
      escribe(3,4,'3) 10 %');
      escribe(3,5,'4) 15 %');
      escribe(2,7,'opción:');
      repeat
        c := readkey;
      until (scan = 1) or (c in ['1'..'4']);
      if scan <> 1 then
        begin
          write(c);
          pr := ord(c) - 49;
        end;
      clrscr;
      window(1,1,80,25);
    end;

(*** Inicializa el patrón actual ***)
  procedure Limpia;
    var
      i : integer;
```

```
    begin
      for i := 1 to 20 do Patron[i] := -1;
      np := 0;
      nb := 1;
      Despliega;
    end;

(*** Inicializa el puerto serial ***)
  procedure IniPuerto;
    begin
      params.ah := 0;
      params.al := 227;
      params.dx := 0;
      intr(20,params);
    end;

(*** Escribe un bit en el puerto serial ***)
  procedure EscPuerto(v : shortint);
    begin
      params.ah := 1;
      if v = -1 then v := 0;
      params.al := v;
      params.dx := 0;
      intr(20,params);
    end;

(*** Envía el patrón actual a través del puerto serial ***)
  procedure EnviaPat;
    var
      i, j : integer;
      r    : array [1..20] of boolean;
      v    : shortint;
    begin
      randomize;
      for i := 1 to 20 do r[i] := false;
      i := 0;
      while i < pr do
        begin
          j := random(20) + 1;
          if not r[j] then
            begin
              r[j] := true;
              i := i + 1;
            end;
        end;
      IniPuerto;
      for i := 1 to 20 do
        if not r[i] then EscPuerto(Patron[i])
        else if Patron[i] = 1 then EscPuerto(-1) else EscPuerto(1);
    end;

(**** Cuerpo principal del programa ****)
  begin
    Inicializa;
```

```
  Presenta;
  repeat
    Menu;
    if op in ['A'..'Z'] then Representa;
    case scan of
      57          : Invierte;
      75,72,80,77 : Mueve;
      66          : Ruido;
      67          : Limpia;
      68          : EnviaPat;
    end;
  until (scan = 1) and (op = #27);
  Termina;
end.
```

```
(* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *)
(*       Recibe letras distorcionadas por el ruido y las corrige             *)
(*      mediante una Madaline                                                *)
(*                                                                           *)
(*  Autor: Ing. Jesús Benjamín Rodríguez García   530946                     *)
(*  Maestría en Ciencias Computacionales                                     *)
(*  I.T.E.S.M.  Campus Estado de México                                      *)
(*                                                                           *)
(*                                                                           *)
(*  Archivo: RECIBE.PAS              Lenguaje: Turbo Pascal 5.0 (Borland)    *)
(*                                                                           *)
(*            (Versión 1.0)             (c) Copyright Octubre de 1992.       *)
(* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *)

Program RECIBE;

  uses
    Crt,
    Printer,
    Dos,
    Libreria;

  const
    MaxTamConEnt = 900;
    MaxNumUni    = 80;
    Letra        : string[26] = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ';

  type
    TipoAdaLiNe = record
                    Peso   : array [0..20] of real;
                    Salida : real;
                    Selec  : boolean;
                  end;

    TipoAdaLiNe2 = record
                     Peso   : array [0..600] of real;
                     Salida : real;
                     Selec  : boolean;
                   end;

    TipoOculta   = array [1..600] of ^TipoAdaLiNe;
    TipoSalida   = array [1..26] of ^TipoAdaLiNe2;

  var
    CapaOculta    : TipoOculta;
    CapaSalida    : TipoSalida;
    Entrada       : array [1..MaxNumUni] of shortint;
    NumSalidas    ,
    NumOcultas    ,
    NumEntradas   : integer;
    op            : char;
    tope          : ^integer;
    params        : registers;
```

```
(**** Propaga la entrada a través de toda la red hasta la capa de salida ****)
  procedure Propaga;
    var
      i, j : integer;
      s    : real;
    begin
      for i := 1 to NumOcultas do
        begin
          CapaOculta[i]^.Salida := CapaOculta[i]^.Peso[0];
          for j := 1 to NumEntradas do with CapaOculta[i]^ do
            Salida := Salida + Entrada[j] * Peso[j];
        end;
      for i := 1 to NumSalidas do
        begin
          CapaSalida[i]^.Salida := CapaSalida[i]^.Peso[0];
          for j := 1 to NumOcultas do with CapaSalida[i]^ do
            if CapaOculta[j]^.Salida >= 0 then Salida := Salida + Peso[j]
            else Salida := Salida - Peso[j];
        end;
    end;

(**** Inicializa los parámetros necesarios en el entrenamiento ****)
  procedure Inicializa;
    var
      i : integer;
    begin
      NumEntradas := 20;
      NumSalidas  := 26;
      NumOcultas  := 600;
      mark(tope);
      for i := 1 to 600 do new(CapaOculta[i]);
      for i := 1 to 26 do new(CapaSalida[i]);
      normal;
    end;

  procedure Presenta;
    begin
      clrscr;
      gotoxy(01,25);
      writeln('     /\    /\   /-----\   /-----\   /-----\   -        -   /\   -
      writeln('     |\\  //|   |     |    |    |   |     |   |        |   |\\  |
      writeln('     | \\// |   |-----|    |    |   |-----|   |        |   | \\ |
      writeln('     |  \/  |   |     |    |    |   |     |   |        |   |  \\|
      writeln('     -      -   -     -   \-----/   -     -   \-----/  -   -   \/
      writeln('
      writeln;
      writeln(' * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
      writeln(' *
      writeln(' *              ▓▓▓▓ Recibe letras distorcionadas por el ruido ▓▓▓▓
      writeln(' *                      y las corrige mediante una Madaline
      writeln(' *
      writeln(' *  Autor: Ing. Jesús Benjamín Rodríguez García   530946
      writeln(' *  Maestría en Ciencias Computacionales
```

```
      writeln(' *  I.T.E.S.M.  Campus Estado de México
      writeln(' *
      writeln(' *
      writeln(' *
      writeln(' *            (Versión 1.0)            (c) Copyright Octubre de 199
      writeln(' *
      writeln(' * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
      writeln;
      writeln;
      pausa(19,25);
    end;

  procedure CargaPesos;
    var
      arch  : file of TipoAdaLiNe;
      arch2 : file of TipoAdaLiNe2;
      a     : TipoAdaLiNe;
      a2    : TipoAdaLiNe2;
    begin
      clrscr;
      assign(arch,'oculta.dat');
      reset(arch);
      assign(arch2,'salida.dat');
      NumOcultas := 0;
      write('-> Cargando los pesos de las capas ocultas: ');
      while not eof(arch) do
        begin
          NumOcultas := NumOcultas + 1;
          read(arch,a);
          CapaOculta[NumOcultas]^ := a;
          if (NumOcultas mod 10) = 0 then write('.');
        end;
      writeln;
      writeln('-> Número de Adalines en la capa oculta: ',NumOcultas);
      writeln;
      close(arch);
      reset(arch2);
      NumSalidas := 0;
      write('-> Cargando los pesos de las capas ocultas: ');
      while not eof(arch2) do
        begin
          NumSalidas := NumSalidas + 1;
          read(arch2,a2);
          CapaSalida[NumSalidas]^ := a2;
          write('.');
        end;
      writeln;
      writeln('-> Número de Adalines en la capa de salida: ',NumSalidas);
      close(arch2);
      pausa(20,25);
    end;

(**** Muestra los pesos de la Madaline ****)
  procedure Mostrar;
```

```
var
  i, j       : integer;
  c          : char;
begin
  clrscr;
  write('¿Mostrar por impresora? (S/N): ');
  repeat c := upcase(readkey); until c in ['S','N'];
  writeln(c);
  writeln;
  writeln('-> Pesos de las unidades de la capa oculta:');
  if c = 'S' then writeln(lst,'-> Pesos de las unidades de la capa oculta:',#10
  window(1,4,80,25);
  for i := 1 to NumOcultas do
    for j := 0 to NumEntradas do
      begin
        write('    W',j:3,',',i:3,' = ',CapaOculta[i]^.Peso[j]:10:6);
        if c = 'S' then write(lst,'    W',j:3,',',i:3,' = ',CapaOculta[i]^.Peso
        if wherex > 60 then
          begin
            writeln;
            if c = 'S' then writeln(lst);
          end;
        if (wherey = 21) and (c = 'N') then
          begin
            pausa(20,21);
            if scan = 1 then
              begin
                window(1,1,80,25);
                exit;
              end;
            clrscr;
          end;
      end;
  window(1,1,80,25);
  clrscr;
  writeln('-> Pesos de las unidades de la capa de salida:');
  if c = 'S' then writeln('-> Pesos de las unidades de la capa de salida:',#10)
  window(1,4,80,25);
  for i := 1 to NumSalidas do
    for j := 0 to NumOcultas do
      begin
        write('    W',j:3,',',i:3,' = ',CapaSalida[i]^.Peso[j]:10:6);
        if c = 'S' then write(lst,'    W',j:3,',',i:3,' = ',CapaSalida[i]^.Peso
        if wherex > 60 then
          begin
            writeln;
            if c = 'S' then writeln(lst);
          end;
        if (wherey = 21) and (c = 'N') then
          begin
            pausa(20,21);
            if scan = 1 then
              begin
                window(1,1,80,25);
```

```
                exit;
              end;
            clrscr;
          end;
      end;
    writeln;
    pausa(20,wherey);
    window(1,1,80,25);
  end;

procedure Termina;
  var
    i : integer;
  begin
    release(tope);
    for i := 1 to 40 do
      begin
        marco1(i,1,81 - i,25);
        sound(i * 50);
        delay(2);
        nosound;
      end;
    clrscr;
    writeln('Hasta la próxima ...');
  end;

procedure Menu;
  begin
    normal;
    clrscr;
    escribe( 1, 1,unstr('-',80));
    escribe( 1, 2,'                                    <▓▓▓ MENU PRINCIPAL ▓▓▓>');
    escribe( 1, 3,unstr('-',80));
    escrinv( 1,25,' [Esc] para terminar | Presione la letra de la opción deseada
    escrinv( 1, 5,'  -> Opciones:      R)ecibir datos     o        M)ostrar los p
    textcolor(4);
    textbackground(7);
    gotoxy(21, 5);
    write('R');
    gotoxy(49, 5);
    write('M');
    normal;
    gotoxy(80,25);
    repeat
      op := upcase(readkey);
    until (op in ['R','M']) or (scan = 1);
  end;

procedure Despliega;
  var
    i, l , c : integer;
  begin
    escribe(28, 7,'/----------------------\');
    escribe(28, 8,'|      |      |      |      |');
```

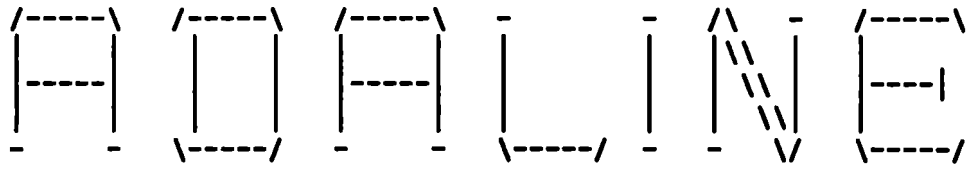
```
      escribe(28, 9,'|     |     |     |     |');
      escribe(28,10,'|-----+-----+-----+-----|');
      escribe(28,11,'|     |     |     |     |');
      escribe(28,12,'|     |     |     |     |');
      escribe(28,13,'|-----+-----+-----+-----|');
      escribe(28,14,'|     |     |     |     |');
      escribe(28,15,'|     |     |     |     |');
      escribe(28,16,'|-----+-----+-----+-----|');
      escribe(28,17,'|     |     |     |     |');
      escribe(28,18,'|     |     |     |     |');
      escribe(28,19,'|-----+-----+-----+-----|');
      escribe(28,20,'|     |     |     |     |');
      escribe(28,21,'|     |     |     |     |');
      escribe(28,22,'\---------------------/');
      textcolor(11);
      for i := 1 to NumEntradas do if Entrada[i] = 1 then
        begin
          l := 8 + trunc(3 * int((i - 1) / 4));
          c := 29 + trunc(6 * ((i - 1) mod 4));
          gotoxy(c,l);
          write('*****');
          gotoxy(c,l + 1);
          write('*****');
        end;
      normal;
    end;

(*** Inicializa el puerto serial ***)
  procedure IniPuerto;
    begin
      params.ah := 0;
      params.al := 227;
      params.dx := 0;
      intr(20,params);
    end;

(*** Lee un bit del puerto serial ***)
  function LeePuerto : shortint;
    var
      c : char;
    begin
      repeat
        params.ah := 3;
        params.dx := 0;
        intr(20,params);
        if keypressed then c := readkey;
        if scan = 1 then
          begin
            LeePuerto := -1;
            exit;
          end;
      until odd(params.ah);
      params.ah := 2;
      params.dx := 0;
```

```
      intr(20,params);
      if params.al = 0 then LeePuerto := -1 else LeePuerto := 1;
    end;

(*** Recibe una letra a través del puerto serial ***)
  procedure Recibir;
    var
      i, j : integer;
      c    : char;
    begin
      for i := 1 to NumEntradas do Entrada[i] := -1;
      Despliega;
      escribe(30,23,'-> Letra recibida: ');
      repeat
        IniPuerto;
        for i := 1 to NumEntradas do
          begin
            Entrada[i] := LeePuerto;
            if scan = 1 then exit;
          end;
        Despliega;
        escribe(50,23,' ');
        Propaga;
        i := NumSalidas;
        j := 0;
        while (i >= 1) and (j = 0) do
          if CapaSalida[i]^.salida >= 0 then j := i else i := i - 1;
        textcolor(15 + blink);
        gotoxy(50,23);
        write(Letra[j]);
        normal;
        if keypressed then
          begin
            c := readkey;
            if scan = 1 then exit;
          end;
      until false;
    end;

(**** Cuerpo principal del programa ****)
  begin
    Inicializa;
    Presenta;
    CargaPesos;
    repeat
      Menu;
      case op of
        'M' : Mostrar;
        'R' : Recibir;
      end;
    until (scan = 1) and (op = #27);
    Termina;
  end.
```

Apéndice B-i

Implementación de la Adaline

Se desarrolló un programa para entrenar una Adaline en base al algoritmo LMS. Al ejecutarse aparece la siguiente pantalla:



```
/----------------------------------------------------------------------------\
|                Implementación de un simulador de AdaLiNe                   |
|    Autor: Ing. Jesús Benjamín Rodríguez García                             |
|    Matricula: 530946                                                       |
|    Maestría en Ciencias Computacionales                                    |
|    ITESM    Campus Sinaloa - Campus Estado de México                       |
|                                                                            |
|                (Versión 1.0)              (c) Copyright Marzo de 1992.     |
\----------------------------------------------------------------------------/
```

[Presione cualquier tecla para continuar]

Después de presionar una tecla aparece la siguiente pantalla de opciones:

```
-> Estado actual:
    Entrenado = No       u = 0.050       Número máximo de iteraciones = 1000
Número de entradas = 2        Tamaño del conjunto de entrenamiento = 4
Valores permitidos en las entradas y la salida = -1 ó 1

-> Opciones:   E)ntrenar el AdaLiNe   o    R)econocer un patrón .... [ ]
-----------------------------------------------------------------------------


 » Simulador de AdaLiNe                                    [Esc] para terminar
```

Si se desea entrenar el AdaLiNe para que reconozca una determinada función, se presiona la letra E; enseguida se pide el número de entradas y el tamaño del conjunto de entrenamiento, posteriormente se pediran las entradas y la salida para cada uno de los patrones en el conjunto de entrenamiento. Estos valores pueden ser un 1 o un -1 por ello al presionar el símbolo - se asumirá automaticamente un -1. Después de introducir el conjunto de entrenamiento, comenzará el entrenamiento durante el cual se mostrará el número de interación que se realiza. Si el entrenamiento termina satisfactoriamente, se mostrarán los pesos obtenidos, en caso contrario se enviará un mensaje de que no fue posible el entrenamiento en el número máximo de iteraciones establecido.

A continuación se presentan el procedimiento de entrenamiento para la función OR con 2 entradas:

```
-> Estado actual:
    Entrenado = Si       u = 0.050       Número máximo de iteraciones = 1000
Número de entradas = 2       Tamaño del conjunto de entrenamiento = 4
Valores permitidos en las entradas y la salida = -1 ó 1

-> Opciones:   E)ntrenar el AdaLiNe   o    R)econocer un patrón .... [E]
--------------------------------------------------------------------------------
 Patrón de entrenamiento 1:
   Entrada 1 = -1
   Entrada 2 = -1
   Salida deseda = -1
 Patrón de entrenamiento 2:
   Entrada 1 = -1
   Entrada 2 = 1
   Salida deseda = 1
 Patrón de entrenamiento 3:
   Entrada 1 = 1
   Entrada 2 = -1
   Salida deseda = 1
```

```
 Patrón de entrenamiento 4:
   Entrada 1 = 1
   Entrada 2 = 1
   Salida deseda = 1

 -> Iteración: 5
             *** Entrenamiento concluido satisfactoriamente ***
   Peso 0 = 0.189698
   Peso 1 = 0.083733
   Peso 2 = 0.198944
                     [Presione cualquier tecla para continuar]
 » Simulador de AdaLiNe                                  [Esc] para regresar
```

Si posteriormente se desea reconocer algunos patrones se presiona la letra R y se presentará la siguiente pantalla:

```
-> Estado actual:
    Entrenado = Si       u = 0.050       Número máximo de iteraciones = 1000
Número de entradas = 2       Tamaño del conjunto de entrenamiento = 4
Valores permitidos en las entradas y la salida = -1 ó 1

-> Opciones:   E)ntrenar el AdaLiNe   o    R)econocer un patrón .... [R]
--------------------------------------------------------------------------------
 Patrón a reconocer:
   Entrada 1 = -1
   Entrada 2 = -1
 Salida obtenida = -1
                     [Presione cualquier tecla para continuar]

 Patrón a reconocer:
   Entrada 1 = -1
   Entrada 2 = 1
 Salida obtenida = 1
                     [Presione cualquier tecla para continuar]

 Patrón a reconocer:
   Entrada 1 = 1
   Entrada 2 = -1
 Salida obtenida = 1
                     [Presione cualquier tecla para continuar]

 Patrón a reconocer:
   Entrada 1 = 1
   Entrada 2 = 1
 Salida obtenida = 1
                     [Presione cualquier tecla para continuar]
```

A continuación se presenta el entrenamiento para la función AND con dos entradas:

```
-> Estado actual:
    Entrenado = Si       u = 0.050       Número máximo de iteraciones = 1000
Número de entradas = 2       Tamaño del conjunto de entrenamiento = 4
Valores permitidos en las entradas y la salida = -1 ó 1

-> Opciones:   E)ntrenar el AdaLiNe   o    R)econocer un patrón .... [E]
--------------------------------------------------------------------------------
 Patrón de entrenamiento 1:
   Entrada 1 = -1
   Entrada 2 = -1
   Salida deseda = -1
 Patrón de entrenamiento 2:
   Entrada 1 = -1
   Entrada 2 = 1
   Salida deseda = -1
 Patrón de entrenamiento 3:
   Entrada 1 = 1
   Entrada 2 = -1
   Salida deseda = -1
 Patrón de entrenamiento 4:
   Entrada 1 = 1
   Entrada 2 = 1
   Salida deseda = 1
 -> Iteración: 11
             *** Entrenamiento concluido satisfactoriamente ***

   Peso 0 = -0.337070
   Peso 1 = 0.258911
   Peso 2 = 0.083269
                    [Presione cualquier tecla para continuar]
 » Simulador de AdaLiNe                                    [Esc] para regresar

-> Estado actual:
    Entrenado = Si       u = 0.050       Número máximo de iteraciones = 1000
Número de entradas = 2       Tamaño del conjunto de entrenamiento = 4
Valores permitidos en las entradas y la salida = -1 ó 1

-> Opciones:   E)ntrenar el AdaLiNe   o    R)econocer un patrón .... [R]
--------------------------------------------------------------------------------
 Patrón a reconocer:
   Entrada 1 = -1
   Entrada 2 = -1
 Salida obtenida = -1
                    [Presione cualquier tecla para continuar]
```

```
Patrón a reconocer:
  Entrada 1 = -1
  Entrada 2 = 1
Salida obtenida = -1
                    [Presione cualquier tecla para continuar]

Patrón a reconocer:
  Entrada 1 = 1
  Entrada 2 = -1
Salida obtenida = -1
                    [Presione cualquier tecla para continuar]

Patrón a reconocer:
  Entrada 1 = 1
  Entrada 2 = 1
Salida obtenida = 1
                    [Presione cualquier tecla para continuar]
```

```
» Simulador de AdaLiNe                                    [Esc] para regresar
```

A continuación se presenta un intento fallido de entrenamiento para la función XOR con 2 entradas:

```
-> Estado actual:
    Entrenado = Si      u = 0.050      Número máximo de iteraciones = 1000
Número de entradas = 2       Tamaño del conjunto de entrenamiento = 4
Valores permitidos en las entradas y la salida = -1 ó 1

-> Opciones:   E)ntrenar el AdaLiNe   o    R)econocer un patrón .... [E]
--------------------------------------------------------------------------------
 Patrón de entrenamiento 1:
   Entrada 1 = -1
   Entrada 2 = -1
   Salida deseda = -1
 Patrón de entrenamiento 2:
   Entrada 1 = -1
   Entrada 2 = 1
   Salida deseda = 1
 Patrón de entrenamiento 3:
   Entrada 1 = 1
   Entrada 2 = -1
   Salida deseda = 1
```

```
 Patrón de entrenamiento 4:
   Entrada 1 = 1
   Entrada 2 = 1
   Salida deseda = -1
 -> Iteración: 1000
    *** No fue posible entrenar el AdaLiNe después de 1000 iteraciones ***
                      [Presione cualquier tecla para continuar]
 » Simulador de AdaLiNe                                    [Esc] para regresar

Hasta la próxima ...
```

Apéndice B-ii

Software de implementación de la Adaline

```
(* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *)
(*  Implementación de un simulador de AdaLiNe                                    *)
(*  Autor: Ing. Jesús Benjamín Rodríguez García                                  *)
(*  Matricula: 530946                                                            *)
(*  Maestría en Ciencias Computacionales                                         *)
(*  ITESM    Campus Sinaloa - Campus Estado de México                            *)
(*                                                                               *)
(*                                                                               *)
(*  Archivo: AdaLiNe.pas              Lenguaje: Turbo Pascal 5.0 (Borland)       *)
(*                                                                               *)
(*            (Versión 1.0)             (c) Copyright Marzo de 1992.             *)
(* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *)

Program SIMULA_ADALINE;

  uses
    crt,
    Libreria;

  const
    MaxTamConEnt = 100;
    MaxNumEnt    = 30;

  type
    TipoAdaLiNe = record
                    Entrada ,
                    Peso    : array [0..MaxNumEnt] of real;
                    Salida  : real;
                  end;

  var
    AdaLiNe       : TipoAdaLiNe;
    Entrenado     : boolean;
    TablaEntradas : array [1..MaxTamConEnt,1..MaxNumEnt] of integer;
    TablaSalidas  : array [1..MaxTamConEnt] of integer;
    TamConEnt     ,
    NumEntradas   : integer;
    op            : char;
    miu           : real;
    NumMaxIter    : longint;

(** Calcula el error de la salida obtenida con respecto a la salida deseada **)
  function CalculaError(SalidaDeseada : real) : real;
    var
      i  : integer;
      sa : real;
    begin
      with AdaLiNe do
```

```
        begin
          Salida := 0;
          for i := 0 to NumEntradas do Salida := Salida + Entrada[i] * Peso[i];
          sa := Salida;
          if Salida >= 0 then Salida := 1.0 else Salida := -1.0;
          if Salida = SalidaDeseada then CalculaError := 0
          else CalculaError := SalidaDeseada - sa;
        end;
    end;

(**** Entrena el AdaLiNe en base a un conjunto de entrenamiento previamente
     establecido ****)
  procedure EntrenaAdaLiNe;
    var
      i, k, j, x, y : integer;
      termina       : boolean;
      e, sa         : real;
      ni            : longint;
    begin
{----- Asigna peso arbitrarios inicialmente -----}
      randomize;
      AdaLiNe.Entrada[0] := 1;
      for i := 0 to NumEntradas do
        if random < 0.5 then AdaLiNe.Peso[i] := - random
        else AdaLiNe.Peso[i] := random;
      k  := 0;
      ni := 0;
      write('-> Iteración: ');
      x := wherex;
      y := wherey;
      repeat
        ni := ni + 1;
        gotoxy(x,y);
        write(ni);
        k := (k mod TamConEnt) + 1;
{----- Aplica un vector de entrenamiento -----}
        for i := 1 to NumEntradas do AdaLiNe.Entrada[i] := TablaEntradas[k,i];
{----- Obtiene el error con respecto a la salida deseada para el vector
      aplicado -----}
        e := CalculaError(TablaSalidas[k]);
{----- Si existe error recalcula los pesos en base a dicho error -----}
        if e <> 0 then for i := 0 to NumEntradas do
          AdaLiNe.Peso[i] := AdaLiNe.Peso[i] + 2 * miu * e * AdaLiNe.Entrada[i];
        termina := true;
        i := 1;
{----- Prueba todo el conjunto de entrenamiento con los nuevos pesos, si un
      patrón falla, entonces se deben recalcular los pesos -----}
        while (i <= TamConEnt) and (termina) do
          begin
            sa := AdaLiNe.Peso[0];
            for j := 1 to NumEntradas do
              sa := sa + TablaEntradas[i,j] * AdaLiNe.Peso[j];
            if sa >= 0 then sa := 1 else sa := -1;
            if sa <> TablaSalidas[i] then termina := false
```

```
          else i := i + 1;
        end;
    until (termina) or (ni = NumMaxIter);
    Entrenado := termina;
  end;

(**** Inicializa los parámetros necesarios para entrenar el AdaLiNe ****)
 procedure Inicializa;
  begin
    miu        := 0.05;
    NumMaxIter := 1000;
    Entrenado  := false;
    TamConEnt := 4;
    NumEntradas := 2;
  end;

 procedure Presenta;
  begin
    normal;
    clrscr;
    gotoxy(01,25);
    writeln('          /-----\   /-----\   /-----\   -       -   /\   -   /-----
    writeln('          |     |   |     |   |     |   |       |   |\\  |   |
    writeln('          |-----|   |     |   |-----|   |       |   | \\ |   |----|
    writeln('          |     |   |     |   |     |   |       |   |  \\|   |
    writeln('          -     -   \-----/   -     -   \-----/ -   -   \/   \-----
    writeln('
    writeln;
    writeln('  /-------------------------------------------------------------
    writeln('  |              Implementación de un simulador de AdaLiNe
    writeln('  |    Autor: Ing. Jesús Benjamín Rodríguez García
    writeln('  |    Matricula: 530946
    writeln('  |    Maestría en Ciencias Computacionales
    writeln('  |    ITESM    Campus Sinaloa - Campus Estado de México
    writeln('  |
    writeln('  |
    writeln('  |              (Versión 1.0)             (c) Copyright Marzo de 199
    writeln('  \-------------------------------------------------------------
    writeln;
    writeln;
    pausa(19,25);
  end;

 procedure Menu;
  begin
    clrscr;
    escrinv(01,25,' » Simulador de AdaLiNe                                    [E
    escribe(01,01,'-> Estado actual:');
    escribe(05,02,'Entrenado = ');
    inverso;
    if Entrenado then write('Si') else write('No');
    normal;
    escribe(25,02,'u = ');
    inverso;
```

```
      write(miu:5:3);
      normal;
      escribe(40,02,'Número máximo de iteraciones = ');
      inverso;
      write(NumMaxIter);
      normal;
      escribe(01,03,'Número de entradas = ');
      inverso;
      write(NumEntradas);
      normal;
      escribe(30,03,'Tamaño del conjunto de entrenamiento = ');
      inverso;
      write(TamConEnt);
      normal;
      escribe(01,04,'Valores permitidos en las entradas y la salida = ');
      inverso;
      write('-1 ó 1');
      normal;
      escribe(01,06,'-> Opciones:   E)ntrenar el AdaLiNe   o    R)econocer un patró
      escribe(01,07,unstr(#205,80));
      escrinv(16,06,'E');
      escrinv(44,06,'R');
      gotoxy(71,06);
      repeat
        op := upcase(readkey);
        if scan = 1 then exit;
        if not (op in ['E','R']) then write(#7);
      until op in ['E','R'];
      inverso;
      write(op);
      normal;
    end;

(**** Captura una entrada cuyo valor es -1 ó 1 ****)
    function CapEntrada : integer;
      var
        c : char;
      begin
        repeat
          c := readkey;
          if scan = 1 then exit;
          if not (c in ['-','1']) then write(#7);
        until c in ['-','1'];
        if c = '-' then
          begin
            writeln('-1');
            CapEntrada := -1;
          end
        else
          begin
            writeln('1');
            CapEntrada := 1;
          end;
      end;
```

```
(**** Caputura el conjunto de entrenamiento e intenta entrenar el AdaLiNe ****)
  procedure Entrena;
    var
      i, j : integer;
      c    : char;
    begin
      escrinv(72,25,'regresar');
      capent(22,03,NumEntradas,01,MaxNumEnt,01,16);
      if scan = 1 then exit;
      capent(69,03,TamConEnt,01,MaxTamConEnt,01,16);
      if scan = 1 then exit;
      window(02,08,79,24);
      clrscr;
      for i := 1 to TamConEnt do
        begin
          writeln('Patrón de entrenamiento ',i,':');
          for j := 1 to NumEntradas do
            begin
              write('  Entrada ',j,' = ');
              TablaEntradas[i,j] := CapEntrada;
              if scan = 1 then exit;
            end;
          write('  Salida deseda = ');
          TablaSalidas[i] := CapEntrada;
          if scan = 1 then exit;
        end;
      EntrenaAdaLiNe;
      writeln;
      if not Entrenado then
        writeln('   *** No fue posible entrenar el AdaLiNe después de ',NumMaxIter,'
      else
        begin
          writeln('            *** Entrenamiento concluido satisfactoriamente ***');
          for j := 0 to NumEntradas do writeln('  Peso ',j,' = ',AdaLiNe.Peso[j]:8:(
        end;
      pausa(20,wherey);
    end;

(**** Captura un patrón y obtiene la salida para él ****)
  procedure reconoce;
    var
      e, j, so : integer;
      s        : real;
    begin
      escrinv(72,25,'regresar');
      window(02,08,79,24);
      repeat
      clrscr;
        if not Entrenado then
          begin
            writeln(#7,'                *** Error, primero debe entrenar el AdaLiNe
            pausa(20,wherey);
            window(01,01,80,25);
```

```
            exit;
          end;
        writeln('Patrón a reconocer:');
        s := AdaLiNe.Peso[0];
        for j := 1 to NumEntradas do
          begin
            write('  Entrada ',j,' = ');
            e := CapEntrada;
            if scan = 1 then exit;
            s := s + e * Adaline.Peso[j];
          end;
        if s >= 0 then so := 1 else so := -1;
        write('Salida obtenida = ');
        inverso;
        writeln(so);
        normal;
        pausa(20,wherey);
      until scan = 1;
    end;

  procedure Termina;
    var
      i : integer;
    begin
      for i := 1 to 40 do
        begin
          marco3(i,1,81 - i,25);
          sound(i * 50);
          delay(2);
          nosound;
        end;
      clrscr;
      writeln('Hasta la próxima ...');
    end;


(**** Cuerpo principal del programa ****)
  begin
    Inicializa;
    Presenta;
    repeat
      Menu;
      case op of
        'E' : Entrena;
        'R' : Reconoce;
      end;
      window(01,01,80,25);
    until (scan = 1) and (op = #27);
    Termina;
  end.
```