15
2e1.

UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE INGENIERÍA

CONTROL DE UN MÓVIL MECÁNICO MEDIANTE REDES NEURONALES EMPLEANDO EL ALGORITMO DE RETROPROPAGACIÓN DEL ERROR

TESIS

QUE PARA OBTENER EL TÍTULO DE:

INGENIERO MECÁNICO ELECTRICISTA

PRESENTAN:

AGUSTÍN DE JESÚS ASTORGA DE RIQUER

JOSÉ LUIS CANO GARCÍA

DIRECTOR DE TESIS:

ING. JUAN MANUEL GÓMEZ GONZÁLEZ

MÉXICO. D.F. 1997

TESIS CON
FALLA DE ORIGEN

Universidad Nacional Autónoma de México
UNAM
Biblioteca Central
Dirección General de Bibliotecas de la UNAM

UNAM – Dirección General de Bibliotecas
Tesis Digitales
Restricciones de uso

DERECHOS RESERVADOS ©
PROHIBIDA SU REPRODUCCIÓN TOTAL O PARCIAL

Todo el material contenido en esta tesis esta protegido por la Ley Federal del Derecho de Autor (LFDA) de los Estados Unidos Mexicanos (México).

El uso de imágenes, fragmentos de videos, y demás material que sea objeto de protección de los derechos de autor, será exclusivamente para fines educativos e informativos y deberá citar la fuente donde la obtuvo mencionando el autor o autores. Cualquier uso distinto como el lucro, reproducción, edición o modificación, será perseguido y sancionado por el respectivo titular de los Derechos de Autor.

Gracias a Dios por permitirme estar aquí, en esta vida.

Gracias a mis Padres, por haberme dado la vida, por haber sacrificado la suya procurando que nunca nos faltara nada a mis hermanos y a mi, gracias por haberme tenido el amor, la libertad, la confianza y la paciencia para lograr esta meta, aunque solo sea una pequeña muestra de lo mucho que los amo.

Gracias a todos y cada uno de mis hermanos, por todo su apoyo y sus enseñanzas, por ser mi ejemplo.

Gracias a Gina, por estar conmigo en todos los momentos de mi vida, por el amor y la paciencia que me has brindado, así como por la esperanza que hiciste nacer en mi, gracias por todo lo que me has enseñado.

Gracias a Juan Manuel, por no solo haber sido mi maestro y director durante la carrera, si no por brindarme su amistad.

Gracias a José Luis, por tenerme la confianza y la paciencia de ser mi compañero y amigo.

Gracias a Raúl, por toda su ayuda y amistad no solo durante la carrera, si no en mi vida.

Gracias a Pedro, por la paciencia que tuvo para conmigo y por su gran ayuda en la realización de la presente tesis.

Gracias a todos los que se atrevieron a brindarme su amistad.

Gracias a todas aquellas personas que sin saberlo, contribuyeron para lograr esta meta.

Agustín

Doy gracias infinitas a las siguientes personas, ya que sin la ayuda de ellas no hubiera podido salir adelante y forjar mi propio camino para conseguir esta, que es mi primer meta tan anhelada, ser Ingeniero.

A mi abuelita Ruth H. Cisneros que aunque ya no está presente, fue la primer persona que empezó a enseñarme cosas nuevas en la vida.

A mi madrina Carmen Alvarado porque junto con mi mamá siempre me han ayudado tanto económica como moralmente para mi realización como profesionista.

A mi prima Teresa Villarreal porque ella fue la primer persona que me sugirió estudiar esta carrera, Ingeniería Electrónica, viendo el futuro que esta tendría al pasar de los años.

A María Luisa García Guzmán, mi madre, por haberme dado la vida y haberme soportado siempre en mis buenos y malos momentos, por darme su amor. Por haber salido adelante para darme una buena educación.

Sobre todo, gracias a todos ellos por haberme demostrado siempre su cariño y por darme su apoyo en todo momento por difícil que este haya sido.

José Luis

A nuestro director de tesis Juan Manuel, por apoyarnos en la realización de la presente tesis, por confiar en nosotros y brindarnos además de sus conocimientos, su amistad.

A Carmen Alvarado y Teresa Villarreal por haber aportado ideas para la estructura de esta tesis, así como para la corrección de frases, puntuación y mejor expresión de nuestras ideas.

Gracias a Pedro, por toda su ayuda y por su amistad.

Gracias a esta Facultad y a esta Universidad por habernos dado las herramientas necesarias para enfrentarnos al mundo y poder hacer algo por nuestro país: "México".

Agustín

y

José Luis

NO DESISTAS

Cuando vayan mal las cosas,
como a veces suelen ir;
Cuando ofrezca tu camino,
sólo cuestas que subir:
Cuando tengas poco haber,
pero mucho que pagar;
y precise sonreir,
aun teniendo que llorar;
Cuando el dolor te agobie
y no puedas ya sufrir;
Descansar acaso debes;
pero Nunca desistir.

Tras las sombras de la duda,
ya plateadas, ya sombrías,
puede bien surgir el triunfo,
no el fracaso que temías.

Y no es dable a tu ignorancia
figurarse cuán cercano
puede estar el bien que anhelas
y que juzgas tan lejano.

Lucha, pues, por más que en la
tarea tengas que sufrir.
Cuando todo esté perdido,
más debemos de insistir.

RUDYARD KIPLING

Índice

Índice

C-1 Introducción

Introducción

En estos tiempos, en los que hay que optimizar todo, desde nuestras vidas hasta las herramientas de las que nos valemos para facilitar nuestras actividades, es en donde nos situamos para analizar la importancia que implica el empleo de las Redes Neuronales.

Es importante mencionar, que a pesar del amplio campo de aplicación que promete el empleo de esta tecnología, su uso y desarrollo se ha limitado, en su mayor parte, a centros educativos y de investigación, esto se debe principalmente a dos aspectos: Por un lado se debe al alto costo que implica cambiar toda una infraestructura existente en las empresas, la cual data en algunos casos de decenas de años, dicho costo no puede ser absorbido por la empresa prefiriendo seguir trabajando con tecnologías viejas y en algunos casos obsoletas, aunque no por ello dejan de ser funcionales.

Por el otro lado se encuentra el tabú que todavía existe en ciertas personas por falta de una educación tecnológica que todavía lo entienden como tema de ciencia ficción, de sistemas inteligentes e independientes al 100% de la intervención del hombre, entendiendo como sistema ya sea una computadora o un mecanismo "híbrido", es decir que involucre parte mecánica con parte computacional.

Es cierto, que el deseo que ha impulsado a través de los años a cientos de investigadores ha sido el crear una máquina a su imagen y semejanza, pero es muy pronto para llegar a una conclusión de esa índole, falta mucho camino por recorrer, lo único que se ha logrado con los avances que se han tenido hasta nuestros días, es que el sistema sea "menos tonto". Recordemos que estos sistemas no son capaces de hacer nada si nosotros no le indicamos detalladamente la acción a realizar y que si no se llega a contemplar una acción, el sistema podía llegar a fallar.

Ahora bien, con el empleo de las Redes Neuronales, sigue siendo necesario indicar los pasos a seguir, pero con la diferencia de que no es necesario contemplar todas las opciones posibles, basta con tomar, por así decirlo, las muestras o patrones más significativos para que el sistema funcione y en caso de que se presente alguna opción no contemplada, dicho sistema pueda "generalizar" y "reconocer" la opción dada. Dicho en otras palabras, el

sistema es capaz de aprender sobre la experiencia, haciendo que el sistema se vuelva de alguna manera, más "independiente" .

En relación a la educación de la gente, es necesario dar a conocer las bondades que presta esta tecnología, pues a pesar de dichas bondades su uso no ha sido lo suficientemente difundido.

Actualmente se pueden encontrar un gran número de aplicaciones, en las que el empleo de la tecnología de las Redes Neuronales es transparente para el usuario. A continuación mostramos algunas de las aplicaciones que existen hasta la fecha[1].

- Negocios
  - Mercadeo
- Procesamiento de formas y documentos
  - Reconocimiento de caracteres impresos en máquina
  - Reconocimiento de caracteres impresos a mano
  - Reconocimiento de gráficos
  - Reconocimiento de caracteres cursivos escritos a mano
- Industria alimenticia
  - Análisis de aromas
  - Desarrollo de productos
- Industria financiera
  - Detección de fraudes
  - Manejo de créditos
- Industria de la energía
  - Operación de centrales hidroeléctricas
  - Obtención de gas natural
- Manufactura
  - Control de Procesos
  - Control de calidad
- Industria médica y del cuidado de la salud
  - Análisis de imágenes
  - Elaboración y detección de drogas

[1] Ver el Apéndice D para una lista de empresas y productos empleando Redes Neuronales.

- Ciencia e Ingeniería
  - Ingeniería química
  - Ingeniería eléctrica
- Industria del transporte y de la comunicación

A pesar de estas aplicaciones tomadas como ejemplo, y de muchas otras que se encuentran en desarrollo, sigue sin tenerse de manera clara el concepto de las Redes Neuronales así como de su uso, el cual no necesariamente tiene que ser en sistemas complejos como los mencionados anteriormente, también puede aplicarse en cosas sencillas.

Es en este punto donde podemos comenzar, respondiendo a algunas preguntas, tales como:

¿Qué es una Red Neuronal?

Se han dado diversos nombres a las Redes Neuronales Artificiales (RNA) dependiendo del area de aplicación en el que se vean inmersas, dichos nombres pueden ser: Sistemas Neuronales Artificiales (SNA), Neurocomputadoras, Procesadores Paralelos Distribuidos (PPD). Estos son sólo algunos de los nombres con que se conoce a los sistemas que tratan de simular, aunque de una manera parcial, burda y exageradamente simple, a la estructura y funcionamiento del cerebro y del Sistema Nervioso Central de los seres vivientes pues actualmente, aún con todas las investigaciones realizadas y con la tecnología existente, sigue sin entenderse el funcionamiento del cerebro, el cual es altamente complejo, por lo cual, sería muy aventurado y pretencioso decir que una red neuronal artificial es un modelo representativo del cerebro. Se trata más bien de ir entendiendo y tratando de reproducir el funcionamiento de sus elementos fundamentales: las neuronas.

Ahora bien, también se ha tratado de dar definiciones de lo que es una Red Neuronal, tales como:

1) "Una Red Neuronal consiste en una serie de neuronas individuales (que son los elementos procesadores). Cada neurona actúa como un elemento procesador independiente. las entradas y los pesos de interconexión son procesados por una función suma (tipicamente una

sumatoria de pesos), el resultado de esta sumatoria es mapeado por una función de transferencia de característica no lineal (generalmente una función sigmoide). La salida de esta función no lineal es la salida de la neurona".

2) "Una Red Neuronal puede consistir en múltiples capas de neuronas interconectadas con otras neuronas en la misma o en diferentes capas. Una topología de conexión de la neurona con otras neuronas puede variar desde una completa interconexión de todas las neuronas de una capa con la posterior, hasta una conexión parcial. Las capas están referidas como capa de entrada, capa(s) intermedia(s) o capa de salida".

3) Las Redes Neuronales deben "aprender" cómo procesar la información de entrada antes de que ésta pueda ser utilizada en una aplicación. El proceso de entrenamiento de una Red Neuronal involucra el ajuste de los pesos de entrada en cada neurona hasta que la salida de la red se aproxima a la salida deseada. Este procedimiento involucra la creación de un archivo de entrenamiento, el cual está formado por los datos de cada nodo de entrada y la respuesta deseada para cada nodo de salida de la red. Una vez que la red está entrenada, sólo los datos de entrada son provistos a la red, la cual "recuerda" la respuesta que "aprendió" durante el entrenamiento.

4) La definición más simple de una Red Neuronal, que más hace referencia a una Red Neuronal Artificial, es provista por el inventor de una de las primeras Neurocomputadoras, el Dr. Robert Hecht-Nielsen. Él define una Red Neuronal como:

" *...un sistema computacional hecho por un alto número de simples elementos procesadores, pero altamente interconectados, los cuales procesan la información por la respuesta en estado dinámico de entradas externas* " *[1]*

Dos conceptos mencionados en las definiciones anteriores y que serán utilizados a lo largo de este trabajo son:

Pesos:

Es una magnitud que indica la fuerza que tiene la conexión entre dos neuronas o elementos procesadores, dicha magnitud funciona como un factor determinante para definir la excitación o inhibición de la neurona.

Conexiones:

Son las uniones entre una o más neuronas, las cuales indican la relación existente entre ambas y tiene que ver con la definición de los pesos.

Decimos que es la unión entre una o más neuronas porque una neurona puede recibir entradas provenientes de ella misma a manera de realimentación.

Las Redes Neuronales Artificiales, son dispositivos procesadores (algoritmos o hardware), que tratan de modelar la estructura de la corteza cerebral animal, pero en mucho menor escala. Una Red Neuronal Artificial grande puede tener cientos o miles de unidades procesadoras, mientras que el cerebro animal, tiene billones de neuronas con su respectivo incremento en magnitud debido a la interacción y al comportamiento.

¿Cuándo debemos considerar utilizar a las Redes Neuronales?

Las Redes Neuronales deben ser aplicadas en situaciones en las que las técnicas tradicionales han fallado en dar resultados satisfactorios, o cuando una mejora en el modelado puede significar una diferencia en la eficiencia de la operación de un sistema, lo que lleva como consecuencia una mejora en la relación costo-beneficio. El sistema trabajará de mejor manera cuando presente una alta tolerancia al error, pues las Redes Neuronales son aproximadores universales.

Resumiendo, debemos considerar el uso de las Redes Neuronales, cuando el número de variables o la diversidad de los datos sean muy grandes, cuando las relaciones entre las variables sean vagamente entendibles, cuando la relación entre estas variables sean difíciles de describir adecuadamente

mediante los métodos convencionales o cuando las variables o capturas presenten semejanzas dentro de un conjunto de patrones, tal como sucede en aplicaciones de procesamiento de señales, de control, de reconocimiento de patrones, producción y reconocimiento del habla, en los negocios, en la medicina, etc.

El punto de partida que se tomó para el desarrollo de la presente Tesis es el de explicar de una manera sencilla los conceptos básicos sobre las Redes Neuronales tanto biológicas como artificiales, así como el presentar una aplicación de esta tecnología, un pequeño "robot" el cual será controlado por medio de símbolos escritos a mano, los cuales serán procesados por una red neuronal bajo el algoritmo de "Retropropagación del Error".

Bibliografía

[1] Caudill, Maureen. *"Neural Network Primer: Part I"* . AI Expert, Feb. 1989.

Keller, Paul. *"Products That Use Neural Networks"*. Pacific Northwest National Laboratory. 1996. http://www.emsl.pnl.gov:2080/docs/cie/neural/products/

C-2 Neurona Biológica

La Neurona Biológica

Antes de comenzar esta sección, cabe mencionar que el estudio teórico del funcionamiento del sistema nervioso se ha limitado al estudio de varias de las especies de seres vivientes, en las que es notoria la especialización dependiendo del hábitat en el que se encuentra y a las actividades que realiza.

Ante esto cabe mencionar que no todos los miembros del Reino Animal poseen cerebro. En el caso de los invertebrados las células nerviosas tienden a agruparse formando ganglios, éstos ganglios están organizados de una manera típica para la mayoría de los invertebrados: los cuerpos de las células nerviosas están dispuestos alrededor de la superficie externa, mientras que las ramas y las conexiones sinápticas constituyen un neurópilo dentro del núcleo ganglionar. También las neuronas tienen una forma característica en los invertebrados: cada célula tiene una sola fibra fuerte que da ramas hacia el neurópilo y luego entra en los conectivos o en las comisuras para conectar con otros ganglios o con las raíces nerviosas para inervar la periferia. En algunos invertebrados se pueden encontrar algunas células grandes y fácilmente reconocibles, las cuales han sido de gran ayuda para los estudios electrofisiológicos.

Estos ganglios tienen una actividad asociada, por ejemplo, en la parte sensorial hay varios tipos de receptores. Algunos de éstos están especializados para el tacto, el dolor y la presión y constituyen un primitivo sistema somatosensorial para detectar la estimulación de la superficie corporal o de la pared corporal. Hay también receptores para detectar el equilibrio, sustacias químicas y la luz.

Pasando a los vertebrados, es notorio que la evolución tuvo gran influencia para que el cerebro se volviera cada vez más complejo y especializado en las especies más avanzadas pero, al mismo tiempo, conservará todas sus antiguas partes. Esta falta de cambio radical no debería sorprendernos, puesto que el medio ambiente, que es la base del reino animal, es el mismo para todos. Lo que hace que el sistema nervioso de una especie sea diferente al de otra es el sitio particular que las especies ocupan en la ecología del mundo y el tamaño y complejidad del cuerpo que el sistema nervioso debe controlar.

A manera de ejemplo, se ve que el tiburón, un pez primitivo cuyos receptores principales son de tipo olfatorio, tiene una enorme proporción de su cerebro dedicada a este proceso sensorial. Por ello no necesita que sus hemisferios cerebrales estén completos, sino sólo un mesencéfalo bien desarrollado para conducir impulsos entre su aparato olfatorio y los centros motores. Si ahora se pasa de los peces a los reptiles, se observa que han tenido lugar varios cambios notables en la evolución del cerebro. El cocodrilo tiene zonas especializadas en el sistema olfatorio menos desarrolladas que el tiburón, probablemente porque realiza menos actividades guiadas por el olfato y más por sus otros sentidos. En los reptiles, los hemisferios cerebrales están poco desarrollados. En las aves los hemisferios cerebrales son mucho más grandes que en los reptiles, los lóbulos ópticos están bien desarrollados al igual que el cerebro, cuya función es coordinar y controlar las actividades motoras.

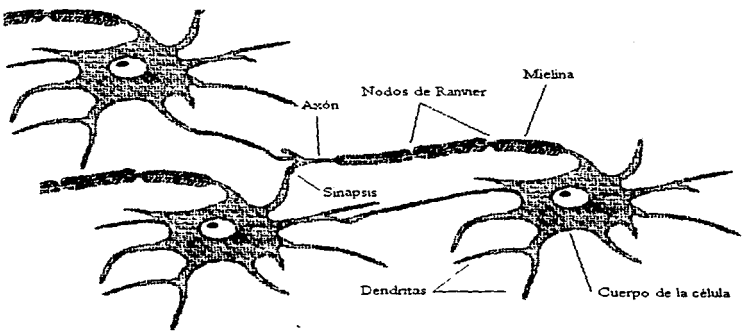
Se podría seguir mencionando ejemplos, pero no viene al caso de esta tesis, sólo cabe hacer notar que la cúspide, hablando desde el punto de vista de la complejidad, no puede ser limitada a un tipo de especie, pues esta depende de la actividad desarrollada por el organismo, así como tambien hay que considerar que el grado de "inteligencia" o capacidad de aprendizaje alcanzada por un organismo dado, tiene mucho que ver con la alta capacidad de procesamiento de la información, la cual va de la mano con la alta densidad de neuronas y sinápsis.

Antecedentes

El Sistema Nervioso Central ha sido estudiado por la medicina desde hace siglos, pero su compleja estructura ha sido revelada apenas hace un siglo. En la segunda mitad del siglo XIX, prevalecían dos escuelas de investigadores de esta rama, la de los *reticularistas* los cuales argumentaban que el sistema nervioso se encontraba formado por una continua e ininterrumpida red de fibras nerviosas; la otra escuela, la de los *neuronistas* o *neuristas* defendían que el sistema nervioso estaba formado por un basto número de simples unidades celulares interconectadas entre sí, las *neuronas*. Con el continuo desarrollo de la ciencia, la lucha entre ambas escuelas fue definida por la venida de una nueva técnica, inventada por Camillo Golgi alrededor de 1880, por la coloración de las fibras nerviosas bajo una reacción de dicromato de plata. Esta técnica fue ingeniosamente

aplicada por el doctor español Santiago Ramón y Cajal en 1888, eliminándose la doctrina del reticularismo al descubrirse las pequeñas separaciones o brechas entre las neuronas individuales.

La investigación detallada de la estructura interna de las células nerviosas, especialmente después de la invención del microscopio electrónico hace ya 50 años, ha revelado que todas las neuronas están constituidas por las mismas partes básicas, independientemente del tamaño y forma de la neurona.



Esquema de un grupo de neuronas
Figura 2.1

La parte central de la célula es llamada cuerpo de la célula o "soma", de la que se proyectan numerosas extensiones en forma de raíz, las "dendritas". La neurona también está formada por una simple fibra tubular, el "axón", el cual se divide en numerosas ramificaciones. El tamaño típico del soma es de unos 10 a 80 micrómetros, las dendritas y el axón tienen un diámetro de apenas unos cuantos micrómetros.

El propósito de las dendritas es servir como elementos receptores para las señales provenientes de neuronas adyacentes.

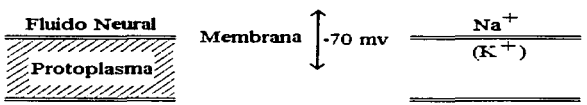
Las dendritas y el soma reciben mensajes de otras células. La información es procesada y el mensaje resultante, si lo hay, pasa a lo largo del axón a otro conjunto de neuronas, por lo que el propósito del axón es la transmisión de la actividad neuronal generada a otra célula nerviosa o fibra muscular. Para el caso de las células nerviosas se utiliza el término interneuronal, mientras que para las fibras musculares se emplea el nombre de neurona motora. Existe un tercer tipo de neuronas, las que reciben la información desde los músculos o de los órganos sensoriales, las cuales son llamadas neuronas receptoras.

La unión entre el final de una rama del axón, la cuál tiene forma de plato, y otra neurona o músculo es llamada "sinapsis". Cada sinapsis entre dos células está separada por una pequeña brecha de aproximadamente 200 nanómetros de ancho, dicha brecha es conocida como "brecha sináptica" o "cleft", apenas visible por la técnica de Ramón y Cajal, pero fácilmente revelada por las técnicas modernas. Existen dos estructuras relacionadas con la sinapsis denominadas "presinapsis" y "postsinapsis" en relación a si se encuentran antes o después de la unión sináptica. Las sinapsis pueden estar localizadas en cualquier parte del cuerpo de la célula o en las dendritas. La fuerza de la sinapsis generalmente disminuye con el incremento de la distancia, el cual se mide desde el cuerpo de la célula hasta la unión sináptica.

La señal nerviosa es transmitida por medio de una reacción electro-química. La transmisión eléctrica prevalece en el interior de la neurona, mientras que el mecanismo químico opera entre neuronas, por ejemplo en las sinapsis.

La transmisión eléctrica está basada en una descarga eléctrica que se genera mediante el flujo de iones de sodio (Na^+) y potasio (K^+), dicha transmisión inicia en el cuerpo de la célula y viaja a lo largo del axón hasta las diversas uniones sinápticas. En un estado de inactividad en el interior de la neurona, el protoplasma, está cargado negativamente en comparación con el líquido neuronal circundante. Esta diferencia de potencial, la cual es aproximadamente de -70 milivolts, es soportada por la acción de la membrana celular, la cual separa el fluido intracelulàr del fluido intersticial

que se encuentra envolviendo a la célula, tal y como se muestra en la figura de la estructura de un axón (figura 2.2). La membrana es permeable para ciertos iones, con lo cual mantiene la diferencia de potencial entre el fluido intracelular y el fluido extracelular. Este efecto se consigue primordialmente mediante la acción de una bomba de sodio-potasio. También están presentes iones de cloro así como iones orgánicos negativos.



Estructura de un axón
Figura 2.2

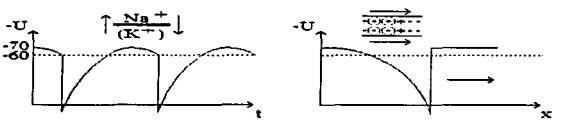
Todos los iones se pueden difundir a través de la membrana, con la excepción de los iones orgánicos, que son demasiado grandes. Dado que los iones orgánicos no pueden salir de la célula por difusión, su carga negativa neta dificulta la entrada a la célula de iones de cloro por difusión; por lo tanto, habrá una concentración más alta de iones de cloro fuera de la célula. La bomba de sodio-potasio determina una concentración más alta de potasio dentro de la célula y una concentración más alta de sodio fuera de ella.

La membrana celular es selectivamente más permeable para los iones de potasio que para los iones de sodio. El gradiente químico del potasio tiende a hacer que los iones de potasio salgan de la célula por difusión, pero la fuerte atracción de los iones orgánicos negativos tiende a mantener adentro el potasio. El resultado de estas fuerzas opuestas es que se alcanza un equilibrio en el cual hay más iones de sodio y cloro fuera de la célula, y más iones orgánicos y iones de potasio dentro de ella.

Las entradas excitatorias, que llegan a la célula por medio de las dendritas, reducen la diferencia de potencial que existe entre los dos lados de la membrana celular. La despolarización resultante en el montículo del axón altera la permeabilidad de la membrana celular por efecto de los iones de sodio. Como resultado, hay un fuerte flujo entrante de iones de sodio positivos, que penetran en la célula, contribuyendo aún más a la

despolarización, la cual, al sobrepasar un cierto umbral, tiene como consecuencia generar el potencial de acción.

Entonces la membrana recupera gradualmente sus propiedades originales y regenera su diferencia de potencial en un período de algunos milisegundos. Durante este período de recuperación, la neurona es incapáz de atender a una nueva excitación, una vez recuperada la neurona, está lista para "disparar" de nuevo.

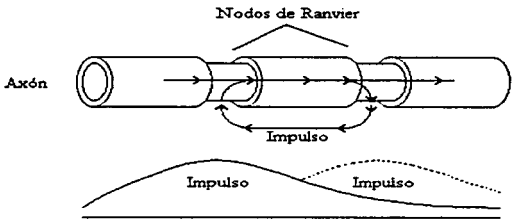


Forma de acción de la neurona
Figura 2.3

La velocidad de propagación de la señal de descarga a lo largo de la fibra nerviosa varía enormemente. En el interior de la célula nerviosa del cerebro humano, la señal viaja a velocidades de aproximadamente 0.5 a 2 m/s, mientras que 2 células cerebrales se comunican entre sí de 20 a 40 m/s, lo cual es aceptable tratándose de una zona local, pero si consideramos la comunicación entre el cerebro y algún miembro periférico, esto toma largos períodos de reacción.

Para incrementar la velocidad de propagación, los axones de las neuronas están compuestos de segmentos individuales, los cuales están cubiertos por una capa de mielina, que es una capa aislante, discontinua en varios puntos, los que son llamados Nodos de Ranvier.

En estos nodos es donde se regenera el impulso. Aquí, los canales de sodio están agrupados entre sí con una densidad de aproximadamente 12,000 canales/μm^2 . En contraste con esto, la membrana mediadora bajo la mielina tiene pocos canales de conductancia dependientes del voltaje, por lo tanto, la propagación a través de los segmentos internodulares es puramente eléctrica tal y como se puede ver en la figura 2.4.



Mecanismo de propagación del impulso a través del Axón
Figura 2.4

Esta disposición de la membrana es muy efectiva porque la sección envuelta de mielina tiene una alta resistencia y una baja capacitancia que hace que la corriente tienda a fluir a lo largo de la fibra hasta el siguiente nodo, más que a escapar al exterior a través de la membrana. En realidad, el impulso salta de nodo en nodo, y por lo tanto, esta forma de propagación se denomina conducción saltatoria.

Este es un mecanismo eficiente que consigue una velocidad máxima de conducción (velocidades de transmisión de hasta 100 m/s), con un mínimo de membrana activa, de maquinaria metabólica y de tamaño de la fibra.

También hay que notar que, siendo el impulso de naturaleza eléctrica y que el axón presenta una resistencia y capacitancia asociada de la cual podríamos obtener una analogía a los conductores eléctricos (sin ser real), la señal (impulso), va decreciendo en amplitud a través del tiempo. Es aquí donde los Nodos de Ranvier, desempeñan un papel de repetidores de la señal, amplificándola en amplitud y retransmitiéndola al siguiente nodo, tal y como se mostró en la figura 2.4.

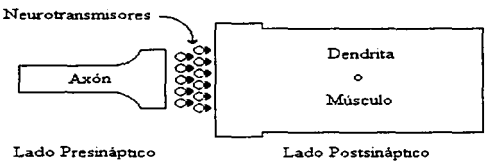
La señal de descarga viaja a lo largo del axón hasta la sinapsis, porque no existe un medio de conducción a la siguiente neurona o fibra muscular. La transmisión de la señal a través de la brecha sináptica es debida más bien a

mecanismos químicos y la transmisión eléctrica ocurre sólo en raras ocasiones.

En la transmisión química, cuando la espiga de la señal llega a la terminal nerviosa presináptica, unas sustancias especiales, llamadas neurotransmisores, son liberadas en pequeñas cantidades desde vesículas contenidas en la cara de la terminal presináptica.

El transmisor es liberado por la influencia de iones de calcio (Ca^{++}) en el axón presináptico durante la despolarización causada por la acción de los iones de sodio (Na^{+}).

Las moléculas neurotransmisoras viajan a través de la brecha sináptica hasta la neurona postsináptica en aproximadamente 0.5 ms. Al llegar los neurotransmisores a unos receptores especiales modifican la conductividad de la membrana postsináptica para ciertos iones tales como sodio, potasio, cloro, etc., los cuales fluyen hacia adentro o hacia afuera de la neurona causando una polarización o despolarización del potencial postsináptico. Después de esta acción, las moléculas transmisoras son rápidamente rotas por enzimas, las cuales son menos potentes en el cambio de la conductividad iónica de la membrana.



Transmisión de la señal en la brecha sináptica por transmisión química
Figura 2.5

Si el potencial de polarización inducido δU es positivo, la sinapsis es excitatoria porque por influencia de la sinapsis se tiende a activar la neurona postsináptica. Si δU es negativo, la sinapsis es inhibitoria, contrarrestando

la excitación de la neurona. Las sinapsis inhibitorias terminan en la unión presináptica del axón siguiente, inhibiendo su habilidad de mandar neurotransmisores a través de la brecha sináptica. En este caso estamos hablando de una inhibición presináptica.

Ante todo esto, surge una pregunta: ¿Bajo qué condiciones es estimulada la neurona postsináptica para activarse o inhibirse?. En principio, una sinapsis sencilla puede inspirar a una neurona a "disparar" o a "no disparar", esto en realidad es muy raro, especialmente si la sinapsis está localizada en el extremo terminal de una dendrita. En realidad, cada axón manda sinapsis a las dendritas y a los cuerpos de numerosas neuronas y cada una de estas neuronas están conectadas con otras, de las cuales también reciben sus señales, entonces, el cuerpo de una neurona actúa como un tipo de dispositivo sumador, el cual suma los efectos polarizantes y/o despolarizantes de sus diversas señales de entrada.

Estos efectos decaen con un tiempo característico de 5 a 10 ms, pero varias señales llegan a la misma sinapsis en un mismo período de tiempo, acumulándose estos efectos excitatorios. Un alto porcentaje de las repeticiones de disparo de una neurona, indica que la señal(es) de entrada fue intensa.

Cuando la magnitud total del potencial de despolarización en el cuerpo de la célula excede el umbral crítico (el cuál es de aproximadamente 10 mv), la neurona dispara.

Entonces, podemos decir que la influencia que tiene una sinapsis depende de varios aspectos, tales como: La fuerza del efecto despolarizante o peso de la conexión, la localización de la sinapsis con respecto al cuerpo de la célula y el porcentaje de repetición en que llegan las señales a la neurona.

Lo anterior es un tratado que pone en evidencia que la fuerza inherente de una sinapsis no está dada por una sola sinapsis, sino por todas, tal y como lo postuló Donald Hebb:

"Cuando un axón de la célula A está suficientemente próximo para excitar a una célula B o toma parte en su disparo de forma persistente, tiene lugar algún proceso de crecimiento o algún cambio metabólico en una de las

células, o en las dos, de tal modo que la eficiencia de A, como una de las células que desencadena el disparo de B, se ve incrementada" [1].

Una sinapsis activa que dispara repetidamente la activación de una neurona postsináptica, puede crecer en fuerza, mientras que otras pueden debilitarse.

Este mecanismo de "plasticidad" sináptica en la estructura de la conectividad neuronal, es conocido como "Regla de Hebb", la cual juega un importante papel en el complejo proceso de aprendizaje.

Bibliografía

[1] Hebb, Donald. *"The Organization of Behavior"* . Wiley. New York, 1949.

Sheperd, Gordon. *"Neurobiología"*. Ed. Labor. Barcelona, 1985.

Chaplin, James. Damers, Aline. *"Introducción a la Neurología y Neurofisiología"*. Ed. Noriega Limusa. México, 1990.

Carlson, Neil. *"Biblioteca de Psicología Experimental. Tomo I"*. Ed. C.E.C.S.A. México, 1987.

C-3 Neurona Artificial

La Neurona Artificial

Antes de comenzar este apartado, cabe decir que los seres vivos, sobre todo los invertebrados y vertebrados inferiores, son organismos que, como se mencionó en el apartado anterior, al no poseer un cerebro en forma, sus células nerviosas agrupadas en ganglios proporcionan un alto grado de especialización. Ahora bien, en el campo ingenieril y específicamente tratándose de las redes neuronales artificiales, se pretende "crear" sistemas flexibles para la solución de determinados problemas a semejanza de los seres vivos, retomando el grado de especialización y la "sencillez" que presenta su sistema nervioso, lo cual, desde el punto de vista tecnológico implica un alto grado de dificultad logrando apenas pequeños resultados.

Una red neuronal artificial, "intenta ser" la representación matemática de una red neuronal biológica. Decimos que intenta ser, porque dada la complejidad todavía no resuelta del funcionamiento del cerebro, todos los trabajos hasta ahora desarrollados son muy burdos en comparación de lo que realmente es, esto es en gran parte debido a las limitantes tecnológicas actuales.

Modelar significa abstraer y simplificar a partir de características generales reales. La elaboración de modelos de neuronas y circuitos de neuronas que intentan reproducir de una manera plausible la arquitectura y funcionamiento del sistema nervioso central, ha sufrido profundas transformaciones en su enfoque a lo largo de la última década, sobre todo por los continuos avances de la neurobiología experimental.

En el modelo matemático, las neuronas están representadas como elementos procesadores (EP), las rutas de entrada están definidas como las interconexiones, los procesos combinan las señales y la salida generada es tratada por una función de transferencia de naturaleza no lineal. La fuerza sináptica de cada conexión está representada como un peso y el cambio en la fuerza sináptica lo definimos como el aprendizaje de la red.

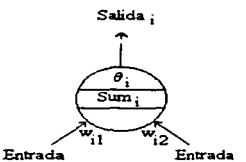
La arquitectura empleada para modelar una red neuronal, define la configuración para cada uno de los elementos básicos indicados anteriormente, en el que el paradigma de las redes neuronales artificiales

está caracterizado en cómo los algoritmos de aprendizaje determinan el cambio de los pesos.

Elementos Procesadores

Un elemento procesador (EP), es un dispositivo simple, cuyo funcionamiento trata de asemejar de una manera muy burda al funcionamiento de una neurona biológica. Aunque los detalles biológicos no pueden ser representados por un modelo de computadora o electrónico (por lo menos hasta nuestros días) la estructura de los EP la podemos tomar para simular o tratar de entender la manera en que trabaja una neurona biológica.

En los modelos de redes neuronales artificiales los EP realizan diversas funciones tales como: la evaluación de las señales de entrada, la suma de las mismas, así como la comparación de dicha suma con una valor de umbral determinado que fija el valor de la salida.



Esquema de un elemento procesador
Figura 3.1

La implementación de una neurona puede ser en un microprocesador de 1 bit, en un procesador con punto flotante o por medio de un amplificador operacional. Dicha implementación depende del modelo de red seleccionado y de la aplicación en cuestión.

Entradas y Salidas

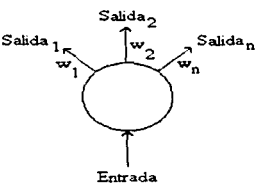
Cada EP puede recibir varias señales de entrada simultáneamente, pero sólo presenta una señal de salida, la cual puede ser positiva o negativa, 0 ó 1 o en

general entre un pequeño rango, dicha salida depende de las señales de entrada de los pesos y del umbral para cada EP. Algunos modelos de red empleados tienen una entrada extra, la cual es denominada "bias" cuyo único objetivo es lograr una convergencia más rápida de la red, aunque el empleo de este término dependerá de la aplicación.

Pueden ser identificados tres tipos de EP dependiendo de sus entradas y salidas, a saber:

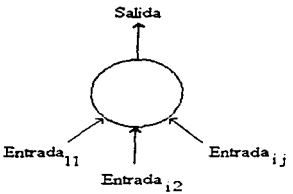
- EP de la capa de entrada
- EP de la capa de salida
- EP de la capa intermedia u oculta

Los EP de la capa de entrada sólo reciben las señales de entrada desde fuentes externas al sistema y sin procesarlas transmiten la señal a las capas siguientes, presentando el esquema mostrado en la figura 3.2.



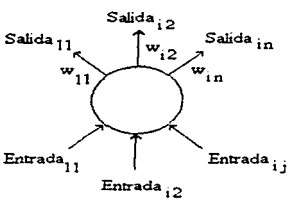
Esquema de un Elemento Procesador de la Capa de Entrada
Figura 3.2

Los EP de salida mandan las señales que reciben hacia afuera del sistema como se muestra en la figura 3.3.



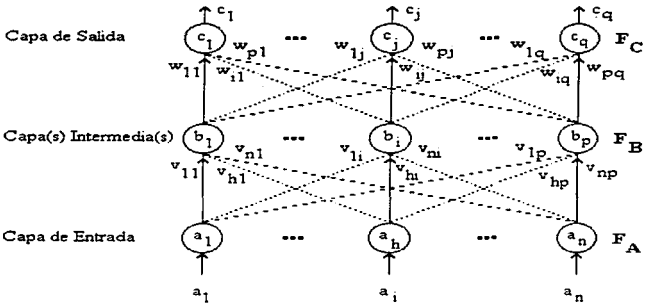
Esquema de un Elemento Procesador de la Capa de Salida
Figura 3.3

Los EP de la capa intermedia o capas intermedias, tienen sus entradas y salidas dentro del sistema como se muestra en la figura 3.4.



Esquema de un Elemento Procesador de la Capa Intermedia
Figura 3.4

En conjunto, todos los EP presentan el esquema mostrado en la figura 3.5.



Cuya nomenclatura es la siguiente:

F_A, F_B, F_C	Capa de Entrada, Intermedia y de Salida respectivamente
$a_1 \ldots a_n$	Señales de entrada
$c_1 \ldots c_q$	Señales de salida
$a_1 \ldots a_n$	Elementos procesadores de la capa de entrada
$b_1 \ldots b_p$	Elementos procesadores de la capa intermedia
$c_1 \ldots c_q$	Elementos procesadores de la capa de salida
v_{np}	Indica los pesos asociados entre las neuronas n, p correspondientes a las capas de entrada e intermedia
w_{pq}	Indica los pesos asociados entre las neuronas p, q correspondientes a las capas de entrada e intermedia

Topología básica de una Red Neuronal Artificial
Figura 3.5

El mecanismo para transformar la señal de entrada en la señal de salida en cada EP es el siguiente.

Función de Transferencia

El primer paso es crear el valor de entrada a la red para cada EP. Algunas entradas pueden ser más importantes que otras, esto se debe al peso correspondiente a cada entrada en el EP. Estos pesos son la fuerza de interconexión entre los EP y normalmente son representados en términos de vectores del tipo:

$$peso_i = (peso_{i1}, \ldots, peso_{in}) \tag{3.1}$$

Después, cada EP recibe todas las señales de entrada, éste calcula el total de entradas dadas desde la ruta de entrada con su respectivo peso. El método mas comúnmente usado es la función sumatoria

$$entrada\_total_p = \sum_{i=1}^{n} peso_{np} entrada_n \tag{3.2}$$

Donde $entrada\_total_p$ es la entrada afectada por el peso correspondiente que recibe cada EP, $peso_{np}$ es el valor del peso de la interconexión desde EP_i a EP_j y $entrada_n$ es la señal de entrada desde EP_j.

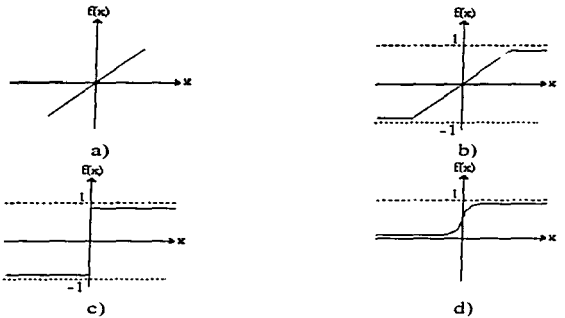
El segundo paso de la transformación es la creación del valor de activación para cada EP, dicho valor se logra por medio de la función de activación o función de transferencia.

Las funciones de activación mapean, escalan o limitan el dominio de los EP en su entrada a un rango específico a la salida del mismo.

Las funciones de activación más comúnmente usadas son:

a) La función lineal.
b) La función rampa.
c) La función escalón.
d) La función sigmoide.

Las cuales son mostradas en la figura 3.6.



Gráficas de las funciones de activación
Figura 3.6

La función Lineal presenta la siguiente ecuación:

$$f(x) = \alpha x \tag{3.3}$$

donde α es una constante en el dominio de los números reales la cual indica la ganancia en la actividad del EP "x".

Cuando la función lineal es limitada al rango $[-\gamma, +\gamma]$, la ecuación anterior se convierte en la función rampa, descrita por la siguiente ecuación:

$$f(x) = \begin{cases} +\gamma & x \geq \gamma \\ x & |x| < \gamma \\ -\gamma & x \leq -\gamma \end{cases} \tag{3.4}$$

donde γ ($-\gamma$) es el valor máximo y mínimo respectivamente, que puede tomar el EP, esos valores son los que se refieren al nivel de saturación de la señal.

Si la función de activación, solamente responde a la señal de entrada, entregando + γ si la entrada es positiva y entregando - γ si es negativa, entonces la función de activación es llamada función escalón, donde δ y γ son escalares positivos.

La función escalón presenta la siguiente ecuación:

$$f(x) = \begin{cases} +\gamma & x \geq 0 \\ -\delta & x \leq \text{cualquier otro} \end{cases} \tag{3.5}$$

Ahora bien, la función de activación más comúnmente usada, es la función sigmoide, esta función con forma de S, es acotada y no decreciente, la cual provee una respuesta no lineal, la ecuación que la representa es la siguiente:

$$f(x) = (1 + e^{-x})^{-1} \tag{3.6}$$

La elección de la función de activación depende de la forma en que se desee sean representados los datos de salida, por ejemplo, si se desea que las unidades de salida sean binarias, se utiliza la función de activación sigmoidal en la que su primer derivada es continua.

$$f(x) = \frac{1}{1 + e^{-x}} \tag{3.7.a}$$

entonces

$$\frac{df(x)}{dx} = \frac{1}{(1 + e^{-x})}\left[1 - \frac{1}{(1 + e^{-x})}\right] \tag{3.7.b}$$

es decir

$$f'(x) = f(x)(1 - f(x)) \tag{3.7.c}$$

La función de transferencia es usada para convertir el valor de activación de cada EP en el valor de salida.

La mayoría de las redes usan el valor de umbral en la determinación de la salida del EP, esto es debido a que el valor de umbral determina cuando el EP se vuelve activo o inactivo, es decir, excita o inhibe.

$$\text{Salida}_i = f(\text{sum}_i) \text{ si } \text{sum}_i > \theta_i \qquad (3.8)$$

$$\text{Salida}_i = 0 \text{ para cualquier otro valor}$$

Donde θ_i es el valor de umbral correspondiente a cada EP. En otras palabras, el EP puede generar la señal de salida si la entrada afectada por el peso es mayor que el umbral, de otro modo, el EP no genera ninguna señal o genera una señal inhibitoria.

Interconexiones

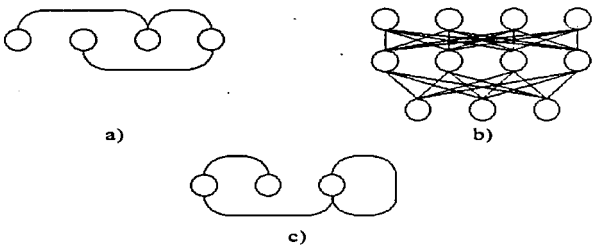
El esquema de interconexión es lo que define a la arquitectura de una red neuronal artificial, es el que indica como se propagará la señal desde un EP a otro o hacia sí mismo. Dichas interconexiones son unidireccionales y tienen un peso asociado para cada conexión, estos pesos forman la memoria de la red.

Las conexiones excitatorias, usualmente señales positivas o cercanas a 1, aumentan el valor de entrada al EP, mientras que las conexiones inhibitorias decrementan el valor.

A continuación mostramos tres diferentes esquemas de interconexión entre los EP en una red neuronal

a) Conexiones entre EP de la misma capa.
b) Conexiones entre EP de diferente capa.
c) Conexiones recurrentes que conectan a un EP consigo mismo.

Mostradas en la figura 3.7.



Esquemas de interconexión entre EP´s
Figura 3.7

Si la información fluye en una dirección, las conexiones son llamadas de propagación hacia adelante. La realimentación permite que la información fluya entre EP en ambas direcciones y/o recursivamente. En la figura 3.5 la realimentación entre capas está dada por una arquitectura de red neuronal de Retropropagación.

Reglas de Aprendizaje

Las reglas de aprendizaje son el último atributo utilizado para definir a una red neuronal artificial. El aprendizaje es considerado como un porcentaje en el cambio de los pesos o memoria de la red.

Existen dos categorías en base al tipo de aprendizaje en el que se basa una arquitectura de red determinada a saber:

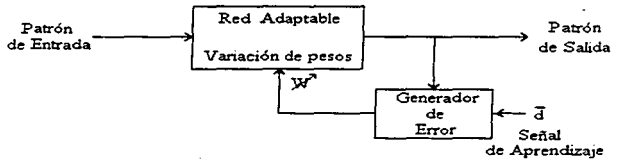
- Aprendizaje supervisado.
- Aprendizaje no supervisado.

El tipo de aprendizaje se debe a que nosotros, dependiendo de la arquitectura de red empleada, podemos considerar un proceso que force a una red a entregar una respuesta dada ante una entrada específica. Una respuesta particular puede o no ser especificada.

El aprendizaje supervisado, es generalmente empleado en redes de propagación hacia adelante, en las que los patrones de entrenamiento están compuestos de dos partes fundamentales, un vector de entrada y un vector de salida, asociando la entrada con su correspondiente salida en cada elemento procesador al cual denominaremos nodo.

La manera en que trabaja este tipo de aprendizaje de manera general es la siguiente:

Un vector de entrada es presentado en la capa de entrada, junto con un conjunto de respuestas deseadas para ese vector, el cual denominaremos **d** para efectos del esquema mostrado abajo, el cual es presentado en la capa de salida. Al realizar una iteración, se genera el error o discrepancia entre la respuesta deseada y la obtenida para cada nodo en la capa de salida, dicho valor de error es utilizado para actualizar el vector de pesos y, como se puede ver, se realimenta para producir una nueva salida. Como se mencionó, mediante este tipo de aprendizaje, los pares de vectores de entrenamiento se van modificando hasta que el porcentaje de cambio de los pesos esté cercano a 0 indicando en realidad que el error total está próximo a cero. Cabe mencionar que no es necesario que lleguemos hasta un valor de cero en el error, nosotros, dependiendo de la aplicación, podemos fijar que el error pueda estar dentro de un rango dado. Esto se hace, porque si hacemos que el error total sea igual a cero, estamos haciendo que la red sea completamente selectiva para el problema para el que fué entrenada, en cambio, manteniendo el error dentro de un cierto rango, hacemos que la red pueda generalizar. Este tipo de aprendizaje es utilizado en arquitecturas de redes neuronales artificiales que necesitan ser entrenadas fuera de línea, es decir, que nosotros necesitamos entrenarlas antes de poder aplicarlas. En la figura 3.8 mostramos este esquema.

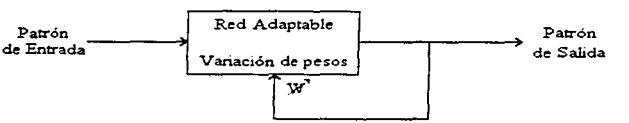


Esquema indicativo del aprendizaje supervisado
Figura 3.8

El aprendizaje no supervisado no incorpora un entrenador externo, trabaja en base a información local y a un control interno. La red se auto organiza usando reglas internas, es decir, la red va modificando sus pesos y demás parámetros de manera dinámica y en línea.

Los pasos que sigue este tipo de aprendizaje son los siguientes: Un vector es aplicado a la capa de entrada, la red se encarga de llegar a un equilibrio, es decir, se auto organiza. El cambio en el valor de los pesos se basa en la información previamente establecida y en el control interno antes mencionado.

En la figura 3.9 mostramos el esquema representativo de este tipo de entrenamiento.



Esquema indicativo del aprendizaje no supervisado
Figura 3.9

Bibliografía

Müller, B. Reinhardt, J. *"Neural Networks An Introduction"*. Springer-Verlag. Germany, 1991.

Dagli, Cihan. *"Artificial Neural Networks for Intelligent Manufacturing"*. Chapman & Hall. Great Britain, 1994.

Zurada, Jacek M. *"Introduction to Artificial Neural Systems"*. West Publishing Company. USA, 1992.

Freeman, James A. & Skapura, David M. *"Redes Neuronales. Algoritmos, aplicaciones y técnicas de programación"*. Addison-Wesley / Diaz de Santos. Wilmington, Delaware. 1993.

Tebo, Albert. *"Artificial neural networks: a developing science"*. OE Reports, 1994.
http://www.spie.org/web/oer/september/neural_net.html

"A Basic Introduction To Neural Networks".
http://ice.gis.uiuc.edu/Neural/neural.html

Fausett. *"Introduction to neural networks. Lecture 2"*. 1995.
e-mail: lpratt@mines.colorado.edu

Phillips, Dwayne. *"The Backpropagation Neural Network"*. C/C++ Users Journal. Volume 14 Number 1. January, 1996.

C-4 Retropropagación

Retropropagación

Antecedentes Históricos

La neurona formal

El primer desarrollo es presentado por McCulloch y Pitts en 1943 en su trabajo *"A logical calculus of the ideas immanent in nervous activity"[1].* En este trabajo, las neuronas fueron presentadas como modelos de las neuronas biológicas y como componentes conceptuales de los circuitos que pueden desarrollar eventos computacionales.

Aprendizaje de Hebb

Hebb señaló en 1949 en su trabajo *"The Organization of Behavior" [2]* que si dos neuronas que están interconectadas entre sí, se activan al mismo tiempo esto indica que existe un incremento en la fuerza sináptica. Así mismo, la forma de corrección que emplea esta regla, es incrementar la magnitud de los pesos si ambas neuronas están inactivas al mismo tiempo.

El Perceptrón

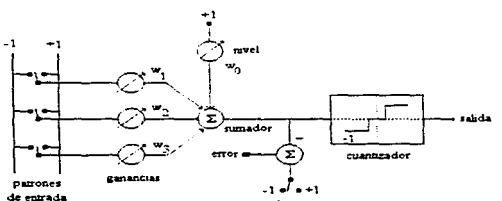
El Perceptrón fué propuesto por Rosenblatt en 1959 en su obra *"Principles of Neurodynamics"[3].* Los Perceptrones son redes de propagación hacia adelante basados en unidades binarias. En una forma sencilla, el Perceptrón consta de una capa de entrada de n elementos, dichas entradas, se propagarán a una capa de m unidades actuadoras y de estas a una sola unidad de salida. El objetivo de esta operación es aprender a dar una transformación dada usando muestras de aprendizaje, con entrada x y su correspondiente salida y . En la definición original la actividad de las unidades actuadoras puede ser cualquier función ϕ de la capa de entrada, pero el procedimiento de aprendizaje sólo ajusta las conexiones de la unidad de salida. La razón para esto es que no hay una fórmula para encontrar el ajuste de conexiones entre la capa de entrada y la función ϕ . La unidad de salida de un Perceptrón es un elemento lineal o elemento de umbral, el cual se adecua a la siguiente regla:

$$salida = F\left(\sum_{i=1}^{n} w_i \phi_i + \theta\right)$$

Rosemblatt probó un teorema sobre el aprendizaje del perceptrón y dado esto, en los 60´s los Perceptrones crearon un gran interés en la comunidad científica sobre las Redes Neuronales. La euforia inicial se convirtió en desilusión cuando Minsky y Paper publicaron su libro *"Perceptrons: An Introduction to Computational Geometry"[4]* en 1969, en el cual ellos mostraban las deficiencias de los modelos del Perceptrón, con lo cual frenaron el desarrollo de las Redes Neuronales. Por un tiempo sólo algunos investigadores continuaron trabajando, los más notables fueron Teuvo Kohonen, Stephen Grossberg, James Anderson y Kunihiko Fukushima.

Adaline

El *"Elemento Lineal Adaptable"* , también llamado Adaline (primeramente conocido como Neurona Lineal Adaptable), fue sugerido por Widrow y Hoff en su obra *"Adaptive switching circuits"[5]*. En una simple implementación física, la cual es mostrada en la figura 4.1, muestra un dispositivo que comprende a un conjunto de potenciómetros conectados a un circuito en una configuración de sumador. Usualmente el bloque central, el sumador, es sólo seguido por un elemento que cuantifica las salidas entre +1 y -1, dependiendo del signo de la suma.



Esquema representativo del Adaline
Figura 4.1

En la figura 4.1 se muestra un sistema con una sola salida, ahora bien, es claro ver que para implementar un sistema con varias salidas en paralelo, basta con montar varios de estos dispositivos, siendo este el caso, recibe el nombre de Madaline *(Multiples Adalines)*.

El problema principal de este tipo de circuito es la determinación de los coeficientes w_i (que son los pesos sinápticos), para que el circuito entregue una salida correcta para un gran conjunto de señales de entrada. Si un mapeo exacto no es posible, la media del error debe de ser minimizada, una operación adaptable debe tener lugar para ir adaptando estos coeficientes iterativamente hasta lograr el valor correcto, para esto, Widrow introdujo la regla Delta la cual realiza los siguientes pasos:

1. *Se aplica un vector de entrada a la red y se calculan los correspondientes vectores de salida.*
2. *Se comparan las salidas obtenidas con las salidas correctas y se determina una medida del error.*
3. *Se determina en qué dirección debe cambiar cada peso con objeto de reducir el error.*
4. *Se determina la cantidad en que es preciso cambiar cada peso.*
5. *Se aplican las conexiones a los pesos.*
6. *Se repiten los pasos del 1 al 5 con todos los vectores de entrenamiento hasta que el error para todos los vectores del conjunto de entrenamiento quede reducido a un valor aceptable. [6]*

Después del documento escrito por Minsky y Paper, el interés por las redes neuronales renació sólo después de que se dieron importantes resultados teóricos en la década de los 80's, lo cual marca formalmente el renacimiento de las Redes Neuronales, siendo el más notable, el desarrollo del algoritmo de Retropropagación del error. Para esto fue de gran ayuda el desarrollo de nuevo hardware, lo cual incrementó las capacidades de procesamiento. Este renovado interés se reflejó en un gran número de científicos dedicados a esta área, en un gran número de conferencias sobre el tema, publicaciones relacionadas con las Redes Neuronales, etc. El interés creció a tal grado que actualmente hay muchas Universidades que tienen un grupo de desarrollo de Redes Neuronales.

Retropropagación

Las redes neuronales trabajando bajo el algoritmo de retropropagación del error, han sido independientemente desarrolladas por un basto número de investigadores de muy diversas disciplinas. La primera aproximación del gradiente descendente en el entrenamiento de una red neuronal artificial multicapa fue desarrollado por un matemático llamado Amari en 1967 con su obra *"A theory of adaptive pattern classifiers"[7]*, quien introdujo una capa intermedia de elementos procesadores para realizar una clasificación no lineal. Las aproximaciones de Amari, iban por el camino correcto, pero no acabó la descripción de cómo formar un mapeo multicapa.

Bryson y Ho *[8]*, en 1969 desarrollaron un algoritmo muy similar al de retropropagación del error para un control no lineal adaptable.

Werbos *[9]*, en 1974, desarrolló de manera independiente el algoritmo de retropropagación del error y varias variantes, llamándolo algoritmo de propagación dinámica hacia adelante, todo esto mientras desarrollaba su tesis doctoral en estática.

Parker *[10]*, en 1982, rediseñó el algoritmo de retropropagación del error, llamándolo algoritmo de aprendizaje lógico y dió los primeros pasos para patentar su trabajo mientras se graduaba en la Universidad de Stanford.

En 1986 Rumelhart, Hinton y Williams *[11]* explotaron el potencial del algoritmo de retropropagación del error, despertando el interés de la comunidad científica.

Son muchas las aplicaciones que resultan difíciles de realizar porque hay muchos problemas cuya solución no es adecuada mediante procesos secuenciales. Por ejemplo, las aplicaciones que deben realizar complejas traducciones de datos que no poseen una función de correspondencia predefinida que describa el proceso de traducción o aquellas que deben proporcionar una mejor aproximación como salida cuando se les presentan unos datos de entrada ruidosos. Las Redes Neuronales se emplean debido a su capacidad para adaptarse o aprender, generalizar u organizar datos cuyas operaciones están basadas en procesos paralelos.

Un algoritmo que ha resultado útil para la solución de estos y muchos otros problemas que requieren el reconocimiento de tramas complejas y la realización de funciones de correspondencia no triviales es el algoritmo de retropropagación del error, descrito formalmente por Werbos y posteriormente por Parker, Rumelhart y McClelland.

Una red trabajando bajo el algoritmo de Retropropagación del error, está diseñada para que funcione como una red multicapa con propagación hacia adelante, empleando un aprendizaje supervisado.

Una red de Retropropagación es entrenada con una serie de entradas y sus correpondientes salidas, motivo por el cual, se dijo que el algoritmo de retropropagación del error trabaja bajo un modo de aprendizaje supervisado.

Una red de Retropropagación funciona de la manera siguiente: Durante la sesión de entrenamiento, los pesos de la red convergen en un punto en el espacio de pesos de la red, en el que el problema es conocido por la red; cuando la red ha alcanzado este punto, puede dar la salida correcta para cada entrada dada. Sin embargo, debido a la naturaleza de la red, no siempre se puede aprender el problema exactamente. La salida producida por la red cuando se presenta una entrada, puede ser una aproximación de la salida correcta, motivo por el cual es difícil decir cuando la red ha aprendido el problema en cuestión.

Las redes de retropropagación son en realidad redes de propagación hacia adelante, cuyo algoritmo de aprendizaje es el algoritmo de Retropropagación del error.

En general, una red neuronal consiste en una serie de elementos procesadores con conexiones directas entre ellos. Es posible asociar un peso a cada una de estas conexiones. Para una red de Retropropagación, estos pesos pueden ser actualizados durante el entrenamiento con la ayuda de una regla de aprendizaje.

En una red de propagación hacia adelante, estos elementos procesadores están agrupados en un número de "n" capas con conexiones directas entre cada uno de los elementos procesadores desde la capa 1 hasta la capa n-1.

Una red de este tipo, consiste en una red de mínimo 3 capas (capa de entrada, intermedia u oculta y de salida), como la mostrada en la figura 3.5.

En el aprendizaje supervisado, el conocimiento de la salida deseada por cada EP es suficiente para calcular el error total producido por la red.

A continuación describiremos el funcionamiento de la red de retropropagación, para lo cual utilizaremos como base la figura 3.5 y su notación.

Las señales de entrada formadas por los patrones a_1 ,... a_n , son presentadas a cada uno de los EP de la capa de entrada, es decir, a los elementos a_1 ... a_n .. Estas señales son transmitidas íntegramente a cada uno de los elementos procesadores de la capa siguiente (capa intermedia u oculta). En cada uno de los elementos procesadores de la capa intermedia (b_1 ... b_n) se efectúa la multiplicación de cada una de las entradas por el peso v_{np} correspondiente a la interconexión de la unidad a_n a la unidad b_p para posteriormente efectuar la sumatoria de esos productos tal y como se muestra en la ecuación 4.1.

$$b_p = \sum v_{np} a_n \tag{4.1}$$

Una vez realizada la sumatoria, el resultado es valuado por la función de transferencia que para el presente caso es la función sigmoide mostrada en la ecuación 3.7.a.

$$S_{bp} = f(b_p) \tag{4.2}$$

El valor obtenido será ahora la entrada aplicada a cada uno de los elementos de la capa de salida, es decir los elementos (c_1 ... c_q), en el que nuevamente se vuelve a efectuar la sumatoria del producto de estas entradas por el peso w_{pq} tal y como se muestra en la ecuación 4.3.

$$c_q = \sum w_{pq} s_{bp} \tag{4.3}$$

El resultado anterior vuelve a ser valuado por la función de transferencia como se muestra en la ecuación 4.4.

$$S_{cq} = f(c_q) \tag{4.4}$$

En donde s_{cq} es la salida de la red para el EP_c en la capa q.

Dicha salida es comparada contra el valor deseado para producir un error en cada elemento procesador, dicho error es calculado de la manera mostrada en la ecuación 4.5.

$$error = salida\_deseada - s_{cq} \tag{4.5}$$

Este error es propagado hacia la capa anterior (capa intermedia) en una cantidad proporcional a la influencia que haya tenido esa neurona para dar la salida deseada.

Para determinar qué fracción del error total es la que corresponde a cada EP de la capa intermedia, se emplea la dispersión de la ley normal de errores (desviación estándar), la cual es mostrada en su forma general en la ecuación 4.6.

$$\sigma^2 = \frac{\sum_{i=1}^{n} \varepsilon_i^2}{n-1} \tag{4.6}$$

Donde:

σ = Desviación estándar
ε = Error
n = Número máximo de iteraciones

Por lo tanto, el error correspondiente es calculado mediante la ecuación 4.7.

$$error\_prop = \frac{1}{2}\sum(salida\_deseada - s_{cq})^2 \tag{4.7}$$

Como lo que se tiene que hacer es minimizar el error, entonces se emplea el método del gradiente descendente, pues los pesos forman una superficie *n*

dimensional, en la que los pesos deben de acercarce al mínimo total, para lo cual lo que se tiene que hacer es derivar el error, es decir:

$$\frac{\delta error\_prop}{\delta w_{pq}} = \frac{\delta}{\delta w_{pq}}\left[\frac{1}{2}\sum (salida\_deseada - s_{cq})^2\right] \tag{4.8}$$

A continuación se muestra el desarrollo matemático para obtener la fórmula de corrección de pesos de la capa de salida.

$$\frac{\delta error\_prop}{\delta w_{pq}} = \frac{1}{2}(2)\left[salida\_deseada - s_{cq}\right]\left(\frac{\delta(salida\_deseada)}{\delta w_{pq}} - a\right)$$

Donde a es la derivada parcial de S_{cq} con respecto a w_{pq}

$$a = \left[\frac{\delta(g(\sum w_{pq} s_{bp}))}{\delta(\sum w_{pq} s_{bp})} * \frac{\delta(\sum w_{pq} s_{bp})}{\delta(w_{pq})}\right]$$

como

salida_deseada es constante entonces su derivada es 0

entonces

$$\frac{\delta error\_prop}{\delta w_{pq}} = \left[salida\_deseada - s_{cq}\right]\left[-\left[\frac{\delta(g(\sum w_{pq} s_{bp}))}{\delta(\sum w_{pq} s_{bp})}\right] * \left[\frac{\delta(\sum w_{pq} s_{bp})}{\delta(w_{pq})}\right]\right]$$

por lo tanto

$$\frac{\delta error\_prop}{\delta w_{pq}} = -\left[\left[salida\_deseada - s_{cq}\right]\left[\frac{\delta(g(\sum w_{pq} s_{bp}))}{\delta(\sum w_{pq} s_{bp})}\right]\left[\frac{\delta(\sum w_{pq} s_{bp})}{\delta(w_{pq})}\right]\right]$$

por lo que la fórmula para la corrección de los pesos w_{pqc} de la capa de salida queda:

$$w_{pqc} = w_{pq} + \eta\left[\frac{\delta error\_prop}{\delta w_{pq}}\right]\left[s_{hp}\right] \tag{4.9}$$

En la que el término η indica la velocidad de aprendizaje, que es un factor que controla la cantidad total de cambios que suceden en el peso correspondiente a una conexión durante el entrenamiento. Un valor pequeño del parámetro de velocidad de aprendizaje conduce a obtener un valor del cálculo de la corrección de error muy rápido, pero no por ello correcto. Ahora bien, un valor grande disminuirá el error haciendo que la red tienda a la convergencia más lento. En general, el valor del factor de velocidad de aprendizaje es menor a la unidad.

Para determinar la ecuación de corrección de los pesos de la capa intermedia se podría suponer que el cálculo sería similar que el empleado para las capas de salida, pero es claro ver, que para el cálculo de la corrección del error en la capa de salida, se utiliza el valor de salida deseado (la ecuación 4.9 involucra ese término). Ahora bien, para el cálculo en la capa intermedia surge un problema y es el que no se conoce cual es la salida correcta que deben de tener estos elementos procesadores, pero se puede efectuar el cálculo con la ayuda de las ecuaciones 4.1, 4.2, 4.3, 4.4 y 4.7 como a continuación se describe.

Partiendo de la ecuación 4.7 se tiene:

$$error\_prop = \frac{1}{2}\sum(salida\_deseada - s_{cq})^2$$

en la que efectuando las sustituciones necesarias con las ecuaciones 4.1, 4.2 y 4.4 se tiene:

$$error\_prop = \frac{1}{2}\sum\left[salida\_deseada - g(\sum w_{pq} f(\sum v_{np} a_n))\right]^2 \qquad (4.10)$$

Minimizando este error se tiene:

$$\frac{\delta error\_prop}{\delta v_{np}} = \frac{1}{2}(2)\left[salida\_deseada - g(\sum w_{pq} f(\sum v_{np} a_n))\right]\left[\frac{\delta(salida\_deseada)}{\delta v_{np}} - b\right]$$

$$b = \left[\frac{\delta g(\sum w_{pq} f(\sum v_{np} a_n))}{\delta(\sum w_{pq} f(\sum v_{np} a_n))}\right]\left[\frac{\delta(\sum w_{pq} f(\sum v_{np} a_n))}{\delta f(\sum v_{np} a_n)}\right]\left[\frac{\delta f(\sum v_{np} a_n)}{\delta(\sum v_{np} a_n)}\right]\left[\frac{\delta(\sum v_{np} a_n)}{\delta v_{np}}\right]$$

en la que, al igual que en el desarrollo para obtener la ecuación de corrección de pesos para la capa de salida.

salida_deseada es constante entonces su derivada es 0

Por lo que la ecuación anterior queda de la siguiente manera:

$$\frac{\delta error\_prop}{\delta v_{np}} = \left[salida\_deseada - g\left(\sum\left(w_{pq} f\left(\sum v_{np} a_n\right)\right)\right)\right][-b] \qquad (4.11)$$

Por lo tanto, la ecuación para calcular la corrección del error en la capa intermedia es:

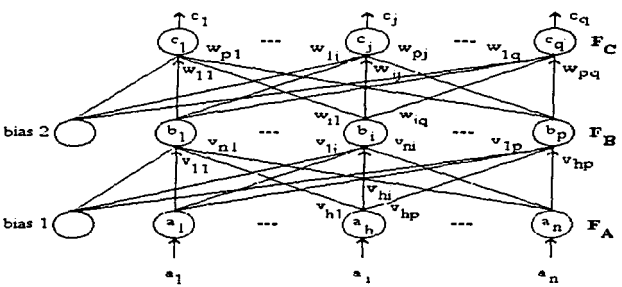
$$v_{npx} = v_{np} + \eta\left(\frac{\delta error\_prop}{\delta v_{np}}\right)(a_n) \qquad (4.12)$$

Al tener los valores de los pesos corregidos se efectúa otra iteración y así sucesivamente hasta que el error calculado por la ecuación 4.5 se encuentre dentro de un rango determinado previamente. Una vez logrado esto, se puede decir que la red ha sido entrenada.

Cabe mencionar que antes del entrenamiento los pesos de las conexiones para cada neurona tienen un valor aleatorio, ninguna vía o ruta es favorecida. Como ya se mencionó, durante el entrenamiento las salidas de la red son comparadas contra un valor deseado generando un error, el cual es usado para corregir a la red, el algoritmo de entrenamiento es el responsable de efectuar estas correcciones por ajuste de los pesos de las conexiones en cada neurona, después de varias iteraciones. Una vez que ya se considera la red entrenada, algunas de las rutas de conexiones tienen mayor importancia que otras, lo que nos permite llegar a obtener una respuesta correcta.

Ahora bien, para agilizar el entrenamiento y lograr la convergencia de la red más rápido, lo que se hace es agregar un término denominado *bias*, el cual se implementa de la manera siguiente:

Tomando como base la topología de red neuronal mostrada en la figura 3.5, se le agrega un elemento procesador a cada capa excepto en la capa de salida, este EP, tendrá conexión directa con cada uno de los EP's de la capa siguiente excepto con el EP de bías, tal y como se muestra en la figura 4.2.



Vista de la topología de la Red Neuronal considerando el término de Bias
Figura 4.2

De la figura 4.2 se puede observar que el EP de bias no recibe entrada alguna, esto es porque por definición, este EP toma un valor constante de 1.

El elemento procesador del bias tiene un peso asociado a sus conexiones con los EP's de las capas siguientes. Con esto se ve que la red puede ser entrenada y que el bias actúa como un factor en el resultado final lo cual hace que los pasos hacia el mínimo total a alcanzar sean mayores reduciendo el tiempo necesario para alcanzar este punto, es decir, lograr la convergencia de la red más rápido.

La consideración de este término es importante pues se usó en el desarrollo de la aplicación, el cual se verá en el capítulo siguiente.

En relación al análisis matemático realizado anteriormente, las fórmulas generales para actualizar los pesos de la capa de salida y de la capa intermedia se mantienen tal y como se muestra en las ecuaciones 4.9 y 4.12, lo que cambia es lo que involucran los términos $\frac{\delta error\_prop}{\delta w_{pq}}$ y $\frac{\delta error\_prop}{\delta v_{np}}$, pues son los que involucran el término de bias. A continuación se presenta su desarrollo:

Partiendo de que en la ecuación 4.1 hay que agregar el término de bias, se tiene lo siguiente para los elementos de la capa intermedia:

$$b_p = (bias1)(v_{np}) + \sum v_{np} a_n \qquad (4.13)$$

Realizando el análisis seguido para obtener la ecuación 4.9 se tiene:

$$S_{bp} = f(b_p) \qquad (4.14)$$

Ahora para los elementos de la capa de salida.

$$c_q = (bias2)(w_{pq}) + \sum w_{pq} s_{bp} \qquad (4.15)$$

$$S_{cq} = f(c_q) \qquad (4.16)$$

Recordando que el error en cada elemento procesador es:

$$error = salida\_deseada - s_{cq}$$

Partiendo de la ecuación 4.6

$$error\_prop = \frac{1}{2}\sum \left(salida\_deseada - s_{cq}\right)^2$$

Donde para minimizar el error se tiene que derivar.

$$\frac{\delta error\_prop}{\delta w_{pq}} = \frac{\delta}{\delta w_{pq}}\left[\frac{1}{2}\sum \left(salida\_deseada - s_{cq}\right)^2\right]$$

desarrollando:

$$\frac{\delta error\_prop}{\delta w_{pq}} = \frac{1}{2}(2)\left[salida\_deseada - s_{cq}\right]\left(\frac{\delta salida\_deseada}{\delta w_{pq}} - a\right)$$

donde *salida_deseada* es una constante por lo que su derivada vale 0 y a es la derivada parcial de S_{cq} con respecto a w_{pq}.

$$a = \frac{\delta g\left((bias2)(w_{pq}) + \sum w_{pq}s_{hp}\right)}{\delta\left((bias2)(w_{pq}) + \sum w_{pq}s_{hp}\right)} * \frac{\delta\left((bias2)(w_{pq}) + \sum w_{pq}s_{hp}\right)}{\delta w_{pq}}$$

por lo tanto:

$$\frac{\delta error\_prop}{\delta w_{pq}} = -\left(salida\_deseada - s_{cq}\right)[a] \qquad (4.17)$$

Siendo este el nuevo valor para calcular $\frac{\delta error\_prop}{\delta w_{pq}}$ para la actualización de los pesos de la capa de salida (ver ecuación 4.9), por eso se mencionó que la fórmula general no varía.

Realizando de manera análoga para la capa intermedia:

$$error\_prop = \frac{1}{2}\sum\left(salida\_deseada - s_{cq}\right)^2$$

sustituyendo la ecuación 4.14 y 4.16 en la anterior:

$$error\_prop = \frac{1}{2}\sum\left(salida\_deseada - g\left((bias2)(w_{pq}) + \sum w_{pq} f\left((bias1)(v_{np}) + \sum v_{np} a_n\right)\right)\right)^2$$

Minimizando:

$$\frac{\delta error\_prop}{\delta v_{np}} = \frac{1}{2}(2)[b] \qquad \text{Ecuación 4.18}$$

Donde:

$$b = \left[salida\_deseada - g\left((bias2)(w_{pq}) + \sum w_{pq} f\left((bias1)(v_{np}) + \sum v_{np} a_n\right)\right)\right](c)$$

$$c = \frac{\delta(salida\_deseada)}{\delta v_{np}} - d$$

$$d = e * h * i * j * k * l$$

donde cada uno de estos elementos son:

$$e = \frac{\delta g\left((bias2)(w_{pq}) + \sum w_{pq} f\left((bias1)(v_{np}) + \sum v_{np} a_n\right)\right)}{\delta\left((bias2)(w_{pq}) + \sum w_{pq} f\left((bias1)(v_{np}) + \sum v_{np} a_n\right)\right)}$$

$$h = \frac{\delta\left((bias2)(w_{pq}) + \sum w_{pq} f\left((bias1)(v_{np}) + \sum v_{np} a_n\right)\right)}{\delta w_{pq} f\left((bias1)(v_{np}) + \sum v_{np} a_n\right)}$$

$$i = \frac{\delta w_{pq} f\left((bias1)(v_{np}) + \sum v_{np} a_n\right)}{\delta f\left((bias1)(v_{np}) + \sum v_{np} a_n\right)}$$

$$j = \frac{\delta f\left((bias1)(v_{np}) + \sum v_{np} a_n\right)}{\delta\left((bias1)(v_{np}) + \sum v_{np} a_n\right)}$$

$$k = \frac{\delta\left((bias1)(v_{np}) + \sum v_{np} a_n\right)}{\delta\left(\sum v_{np} a_n\right)}$$

$$l = \frac{\delta\left(\sum v_{np} a_n\right)}{\delta\left(v_{np}\right)}$$

Siendo $\frac{\delta error\_prop}{\delta w_{pq}}$ el nuevo valor para la actualización de los pesos de la capa intermedia (ver ecuación 4.12).

Los valores obtenidos mediante las ecuaciones 4.17 y 4.18 son los nuevos valores de los pesos con los que trabajará la red, ejecutándose este ciclo tantas veces como sea necesario para minimizar el error hasta un valor determinado.

La explicación que se ha dado en este capítulo sobre el algoritmo de retropropagación del error es la base matemática para el desarrollo de la aplicación.

Bibliografía

[1] Warren S. McCulloch & Walter Pitts. *"A logical calculus of the ideas immanent in nervous activity"*. Bulletin of Mathematical Biophysics, 1943.

[2] Donald O. Hebb. *"The Organization of Behavior"*. Wiley. New York, 1949.

[3] Rosenblatt, F. *"Principles of Neurodynamics"*. Washington, 1962.

[4] Marvin Minsky & Seymour Papert. *"Perceptrons"*. MIT Press. Cambridge, MA, 1969.

[5] Widrow, B. & Hoff, M. *"Adaptive switching circuits"*. WESCON Convention Record, 1960.

[6] Freeman, James A. & Skapura, David M. *"Redes Neuronales. Algoritmos, aplicaciones y técnicas de programación"*. Addison-Wesley / Diaz de Santos. Wilmington, Delaware. 1993.

[7] Amari, S-I. *"A theory of adaptive pattern classifiers"*. IEEE Transactions on Electronic Computers. 1967.

[8] Bryson, A. & Ho, Y. *"Applied Optimal Control"*. Blaisdell. New York. 1969.

[9] Werbos, P. *"Beyond regression: New tools for prediction and analysis in the behavioral sciences"*. Ph. D. Dissertation, Harvard University. 1974.

[10] Parker, D. *"Learning logic"*. Invention report. Office of Technology Licensing. Standford University. 1982.

[11] Rumelhart, D. Hinton, G. & Williams, R. *"Learning representations by backpropagation errors"*. Neture. 1986.

Simpson, Patrick K. *"Artificial Neural Systems. Fountations, Paradigms. Applications, and Implementations"*. Pergamon Press. San Diego, 1990.

Wulms, Alex. " *Project Study Neural Networks. On the dynamic behaviour of backpropagation networks"*. St. Nr. 8949670. October 1994.

Kröse, Ben & van der Smagt, Patrick. *"An introduction to Neural Networks"*. University of Amsterdam. Netherland. 1993.

Phillips, Dwayne. *"The Backpropagation Neural Network"*. C/C++ Users Journal. Volume 14 Number 1. January, 1996.

Espinosa Espinosa, Ismael. *"Las redes neuronales artificiales"*. ICYT Informacion Científica y Tecnológica. Vol. 12, Núm. 163. Abril de 1990.

Leonetti del Negro, Giovanni. *"Simulador de Redes Neuronales Artificiales"*. Tesis de Licenciatura. Universidad Nacional Autónoma de México. México D.F. 1992.

Fausett. *"Introduction to neural networks. Lecture 2"*. 1995. e-mail: lpratt@mines.colorado.edu

Hamagami, Fimiaki. *"Major History of Neural Networks"*. http://kiptron.psyc.virginia.edu/

Carpenter, Gail. *"A Brief History of Neural Networks"*. Neural Computing week at the Chinese University of Hong Kong. 1995. http://nucleus.cs.cuhk.hk:1050/seminar/95032001.html

Bronshtein, I. And Semendiaev, K. *"Manual de Matemáticas para ingenieros y estudiantes"*. *Editorial MIR. URSS. 1988.*

Gieck, Kurt. *"Manual de fórmulas técnicas"*. Representaciones y Servicios de Ingeniería, S.A. México. 1981.

Spiegel, M. "*Cálculo Superior*". McGraw-Hill. Colombia, 1969.

C-5 Desarrollo de la Aplicacion

Desarrollo de la aplicación

En el campo de las Redes Neuronales, existen varios algoritmos con los cuales se puede trabajar, la elección del mismo depende de la aplicación hacia la que esté enfocado su uso. Uno de los algoritmos más usados es el de Retropropagación del error en el que se encuentran un gran número de aplicaciones posibles[1], dichas aplicaciones pueden ser:

- Procesamiento de imágenes.
- Procesamiento de voz.
- Procesamiento de textos.
- Procesamiento de señales en general.
- Predicción de errores.
- Optimización de procesos.
- Diagnóstico de sistemas.
- Procedimientos de Control.
- Reconocimiento de caracteres.
- Procesos generales de reconocimiento de patrones.

En relación a su implementación[2], esta puede ser en:

- Circuitos Integrados (Neurochips)
- Coprocesadores
- Vía software

Aunque existen diversas aplicaciones empleando Redes Neuronales, la mayoría de ellas son costosas por el hardware que involucra implementarlas. El objetivo de esta tesis es explicar de una manera sencilla los principios de las Redes Neuronales por medio de una aplicación un tanto didáctica. También se quiere enseñar la posibilidad del uso de las redes neuronales en actividades no tan "científicas", como lo puede ser en "juguetes". Considerando a estos "juguetes" como ejemplos de aplicaciones prácticas pero a pequeña escala.

[1] Ver el Apéndice D para una lista de empresas y productos empleando Redes Neuronales.
[2] Ver el Apéndice D para una lista de empresas y productos empleando Redes Neuronales.

Objetivo

Mostrar un ejemplo didáctico del empleo de la tecnología de las Redes Neuronales Artificiales.

Para el presente trabajo se seleccionó el algoritmo de Retropropagación del error por las facilidades que presenta el mismo y por ser de los más usados en el reconocimiento de patrones.

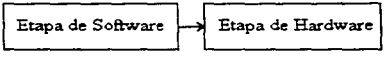
El ejemplo seleccionado fue el control de un móvil mecánico vía software. El móvil mecánico que utilizamos fue un pequeño robot.

El control que se llevará a cabo se basa en la interpretación de símbolos por parte de la Red Neuronal.

El control del móvil mecánico se llevará de la siguiente manera:

Una serie de patrones que indican un movimiento determinado del móvil, son presentados a la red, los mismos son introducidos a la computadora, dibujándolos en la pantalla, con la ayuda del mouse o de un tablero apuntador electrónico. Una vez realizada la captura de los patrones, un programa (el cual trabaja bajo el algoritmo de propagación hacia adelante), procesa dichos patrones, determinando cuales fueron los patrones introducidos y efectuando el movimiento correspondiente a cada patrón introducido para que el móvil posteriormente lo realice.

La aplicación consta de dos bloques principales, los que son mostrados en la figura 5.1



Bloques principales de la aplicación
Figura 5.1

Expandiendo los bloques se tiene que la aplicación puede ser representada por 5 módulos, los cuales se observan en la figura 5.2

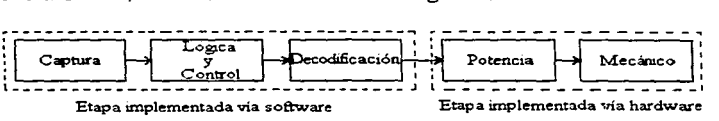


Diagrama a bloques de la aplicación
Figura 5.2

Etapa de Captura.

La etapa de Captura, permite tal y como su nombre lo indica, la captura de los símbolos a reconocer por la Red Neuronal. Esto se logra dibujando los símbolos en la pantalla, en una retícula provista por el programa de aplicación. Para dibujar los patrones, se emplea el mouse o cualquier dispositivo apuntador electrónico.

Etapa de Lógica y Control y Etapa de Decodificación.

La etapa de Lógica y Control, así como la etapa de Decodificación son realizadas vía software, por medio del programa de aplicación el cual se explicará más adelante.

Etapa de Potencia.

Para efectuar el movimiento, el programa de aplicación entrega como salida del bloque de Decodificación y como entrada del bloque de Potencia un número de 4 bits, dicho número es transmitido a la etapa de potencia vía el puerto paralelo de la computadora. Considerando que el puerto maneja niveles TTL, es decir, voltajes de 0 a 5 volts pero con una corriente del orden de 10 a 20 mA, fue necesario implementar una etapa de potencia para alimentar los motores del móvil mecánico, pues estos consumen alrededor de 500 mA cada uno.

La etapa de potencia implementada es mostrada en la figura 5.3

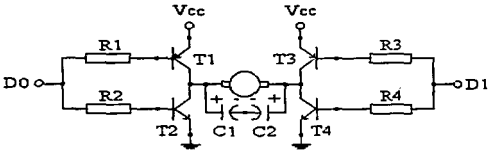


Diagrama de la etapa de potencia.
Figura 5.3

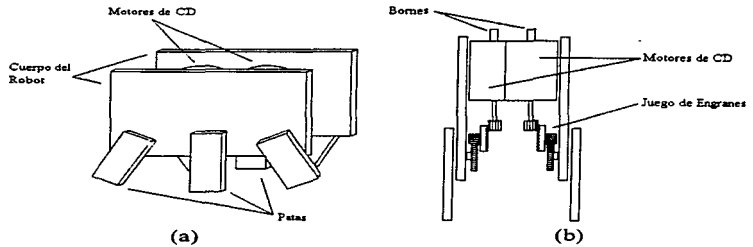
En donde:

Vcc es el voltaje de alimentación y acción de los motores, el cual es de 5 V.
D0 y D1 son las entradas digitales provenientes del puerto paralelo.
T1 y T3 Transistores NPN
T2 y T4 Transistores PNP
Se necesita una etapa de potencia para cada motor, pero ambas etapas trabajan de manera idéntica.

Etapa Mecánica.

Esta etapa se encuentra formada por el móvil a controlar

El móvil empleado es un pequeño "robot" prestado por el Laboratorio de Cibernética de la Facultad de Ciencias de la UNAM. Está formado por dos motores de C.D. que, mediante un juego de engranes para dar el par necesario, mueven un conjunto de 6 patas (3 de cada lado). El móvil se muestra en la figura 5.4



Esquema representativo del robot empleado
Figura 5.4

El robot puede realizar los siguientes movimientos:

- Movimiento hacia adelante
- Movimiento hacia atrás
- Giro

El giro se logra haciendo que las patas de ambos lados se muevan a la vez, pero en sentido opuesto, con lo cual se logra que el giro sea más rápido que moviendo sólo las patas de un lado permaneciendo las del otro lado inmóviles.

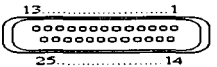
Con base a la figura 5.4.b, se observa que el robot consta de 2 motores, un motor para cada lado, teniendo 2 bornes cada uno de ellos. Por lo anterior, se necesitan 2 bits para la acción de cada motor, con lo cual se logran las combinaciones necesarias para polarizar cada uno de los motores para efectuar un movimiento dado.

Las combinaciones posibles y su respectivo movimiento son las mostradas en la tabla 5.1.

0000 Motores Apagados
0110 Vuelta
1010 Atrás
0101 Adelante

Combinaciones necesarias para efectuar el movimiento de los motores.
Tabla 5.1

Como se dijo anteriormente, estas salidas son provistas al móvil vía el puerto paralelo de la PC. Dado que se trabajó con 4 bits, sólo se emplearon 4 de las 8 líneas de datos que provee el puerto paralelo, tal y como se muestra en la figura 5.5.



Número de Pin	Señal
1	Estroboscopio 5
2	Bit de datos 0
3	Bit de datos 1
4	Bit de datos 2
5	Bit de datos 3
6	Bit de datos 4
7	Bit de datos 5
8	Bit de datos 6
9	Bit de datos 7
10	Acuse de recibo
11	Ocupado
12	Fin de papel
13	Selección
14	Salto de Línea
15	Error
16	Inicializar
17	Selección de entrada
18-25	Tierra

Localización de lo pines en un puerto paralelo y su patigrama.
Figura 5.5

Con base en lo escrito en el párrafo anterior, únicamente se utilizan las líneas D0 a D3, posteriormente se explicará la manera en que se dan estos valores de ceros y unos.

Dado que lo que se necesitaba era controlar los motores con la combinación mostrada en la Tabla 5.1 se implementó la etapa de potencia mostrada en la figura 5.3.

En la tabla 5.2 se explica su funcionamiento.

Cuando se presentan los datos mostrados en la Tabla 5.1, se tiene que:

D0	D1	T1	T2	T3	T4	Mov. de los motores
0	0	Abierto	Abierto	Abierto	Abierto	Ninguno
0	1	Abierto	Abierto	Cerrado	Cerrado	Mov. hacia adelante
1	0	Cerrado	Cerrado	Abierto	Abierto	Mov. hacia atrás
1	1	Cerrado	Cerrado	Cerrado	Cerrado	Ninguno

Tabla de acción de la etapa de potencia
Tabla 5.2

Desarrollo de la etapa de Software

El desarrollo e implementación de las Redes Neuronales se puede basar en 5 pasos fundamentales a saber:

I. Preparación de los datos de entrenamiento.
II. Creación de la Red, selección de la topología y del algoritmo de aprendizaje.
III. Entrenamiento de la Red.
IV. Prueba y uso de la Red (implementación).

Desarrollando los puntos anteriores.

I. Preparación de los datos de entrenamiento

Ya se señaló que para lograr el movimiento del móvil mecánico, se introducen a la PC una serie de "símbolos", mismos que se dibujan en una "retícula" provista por el programa de aplicación.

Para el control del móvil mecánico se utiliza el programa de aplicación, el cual provee de una interface amigable. Los símbolos que se introducen y que indican un movimiento dado del móvil son patrones a seguir, los cuales serán "analizados" por la Red Neuronal.

El programa de aplicación se puede considerar como un programa intérprete, el cual traducirá una serie de mnemónicos y ejecutará la acción que le corresponda.

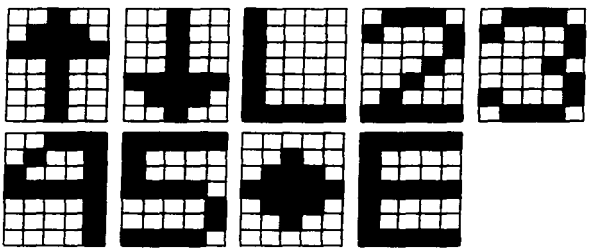
Teniendo en cuenta esta consideración, se puede ver que en el programa de aplicación se empleó como medio de "programación" una serie de símbolos o patrones básicos (mnemónicos), los cuales definen el movimiento del móvil. Dichos movimientos son:

- Movimiento hacia adelante
- Movimiento hacia atrás
- Vuelta de 45 grados en sentido antihorario

Para lo cual se empleó como simbología los siguientes patrones:

- Una flecha apuntando hacia arriba indicando el movimiento hacia adelante.
- Una flecha apuntando hacia abajo indicando el movimiento hacia atrás.
- Una letra "L" indicando la vuelta a 45 grados
- Números del 2 al 5 indicando el número de repeticiones del movimiento en cuestión.
- Un círculo relleno indicando terminación del programa.
- Una letra "E" marca el comienzo de la ejecución del programa.

En la figura 5.6 se muestran los patrones "básicos" con los cuales se comenzó a trabajar.



Patrones "ideales" empleados
Figura 5.6

Cabe hacer notar que los movimientos a realizar por el móvil, así como el número de los patrones empleados puede ser mayor, pero para fines demostrativos se limitó el número de los mismos. La elección de la forma

de los patrones empleados fue completamente al azar, teniendo como única restricción (por fines prácticos y demostrativos), que los patrones fueran diferenciables entre sí. También es posible emplear patrones ruidosos, con lo cual se puede demostrar que la red es capaz de generalizar.

En cuanto al tamaño de la matriz empleada se seleccionó que fuera de 5x7 por conveniencia, para evitar variaciones de los patrones por rotaciones y translaciones.

Para comenzar, se necesita de algún medio para "ir dibujando los patrones". Inicialmente se utilizó el programa *Reticula*[3], que como su nombre lo indica permite ir dibujando en una retícula o matriz de tamaño (**nxm**) diversos patrones de entrada con su correspondiente salida.

Hay que recordar que el algoritmo de Retropropagación del Error es un algoritmo de aprendizaje supervisado (ver figura 3.8).

Los archivos que contienen la información relacionada a los patrones quedan guardados en archivos de extensión **.pat** bajo el siguiente formato[4]:

```
BARRIDO
5
12
4
0.00 0.00 1.00 0.00 0.00 0.00 1.00 1.00 1.00 0.00 0.00 1.00
0.00 0.00 0.00 0.00
0.00 0.00 1.00 0.00 0.00 0.00 1.00 0.00 1.00 0.00 0.00 1.00
0.00 0.00 0.00 0.00
0.00 0.00 1.00 0.00 0.00 0.00 1.00 0.00 1.00 0.00 0.00 0.00
0.00 0.00 0.00 0.00
0.00 0.00 0.00 0.00 0.00 0.00 1.00 1.00 1.00 0.00 1.00 0.00
0.00 0.00 0.00 0.00
0.00 0.00 0.00 0.00 0.00 0.00 1.00 1.00 1.00 0.00 1.00 0.00
0.00 0.00 0.00 0.00
```

[3] Programa desarrollado por el Ing. Juan Manuel Gómez González. Facultad de Ingeniería. Universidad Nacional Autonoma de México. México D.F.

[4] El formato del archivo entregado por el Programa Reticula es el requerido (salvo pequeñas modificaciones), por el programa de entrenamiento Cibernet.

En los que se pueden identificar los siguientes elementos:

- Una cabecera, la cual es indicada mediante la cadena **BARRIDO**
- Número total de pares de patrones entrada.- salida capturados: **5**
- Número de elementos que forman el patrón de entrada: **12**
- Número de elementos que forman el patrón de salida: **4**
- Después vienen en secuencia los valores que toma cada celda de la matriz, tanto de entrada como de salida, que para este ejemplo es de 3x4=12 para la entrada y de 4x1=4 para la salida.

Los datos propios del patrón presentan el siguiente formato:

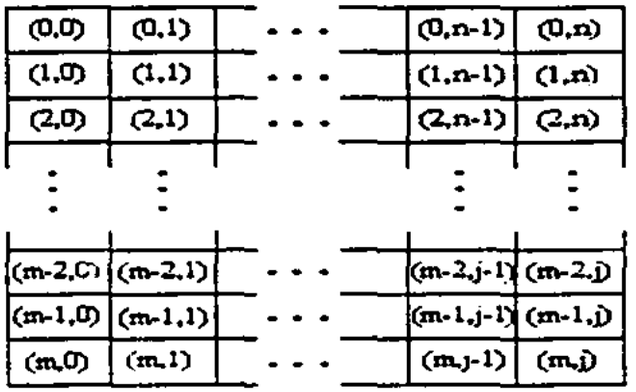
Para la entrada:

(0,0) (0,1) (0,2) (1,0) (1,1) (1,2) (2,0) (2,1) (2,2) (3,0) (3,1) (3,2)

Para la salida:

(0,0) (0,1) (0,2) (0,3)

Estos elementos son tomados de la matriz en la que se dibujan los patrones (ver figura 5.6). Los elementos son leídos de la manera en que se muestra en la figura 5.7.



Forma en que son leídos los patrones en la matriz
Figura 5.7

Inicialmente los valores posibles de cada uno de estos vectores fue 0 ó 1 (posteriormente se explicará el por qué de estos valores).

Este programa finalmente fue sustituido por uno incorporado en el propio programa de aplicación, cambiando los valores posibles a -1, 1. También cambió la estructura del archivo que entrega siendo la mostrada en la figura 5.8.

```
8
4
-1.00 -1.00 -1.00  1.00 -1.00 1.00 -1.00 -1.00 1.00 1.00  1.00  1.00
  1.00  1.00 -1.00 -1.00 -1.00 1.00 -1.00 -1.00 1.00 1.00  1.00  1.00
```

Estructura del archivo de patrones entregada por el programa de aplicación
Figura 5.8

En esta estructura[5] existen dos números de cabecera, el primer número (8) indica el número total de entradas, el segundo número (4) indica el número total de salidas, a continuación viene el código para cada patrón (un patrón por renglón), leyendo los primeros 8 números como las entradas y los últimos 4 como las salidas.

[5] La modificación de los valores posibles (-1 ó 1), así como el cambio del formato del archivo de patrones se debió al cambio del programa de entrenamiento.

II. Creación de la Red, selección de la topología y del algoritmo de aprendizaje

Para la creación de la red, se seleccionó una topología de 3 capas: capa de entrada, capa intermedia y capa de salida

La capa de entrada se formó de 35 nodos, los cuales vienen de la matriz de 7x5 en la que se dibujan los patrones, siendo un nodo por cada elemento de la matriz. La capa de salida está formada por 4 nodos, los cuales provienen de una matriz de 4x1. La dimensión de esta matriz se basa en los valores mostrados en la tabla 5.1.

Para elegir el número de nodos que debería tener la capa intermedia se empleó la ecuación 5.1.

$$\sqrt[3]{(pe)x(ps)} \qquad (5.1)$$

con:

pe = patrones de entrada

ps = patrones de salida

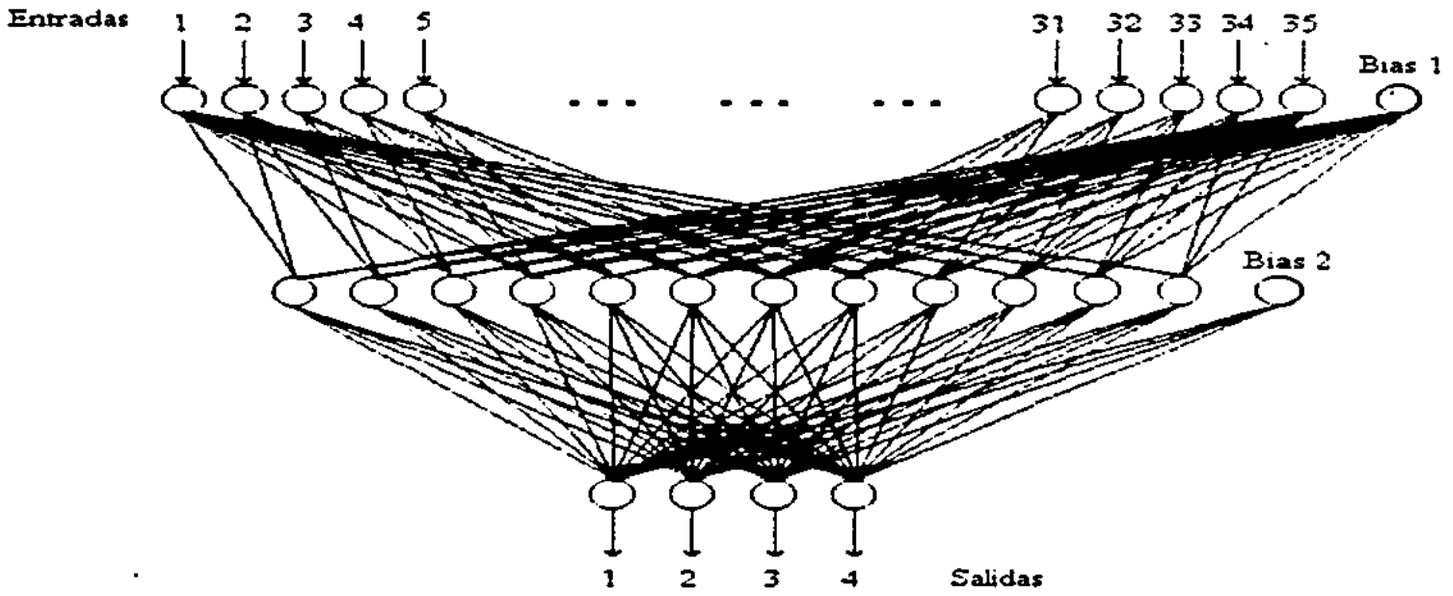
Por lo que nuestra capa intermedia consta de 12 nodos.

Se hicieron pruebas variando el número de nodos y de capas intermedias. Al hacer esto se pudo observar que a mayor número de nodos y de capas el *tiempo de procesador*[6] que requería para ser entrenada esta red se incrementaba sin que por ello se mejorara el *grado de aprendizaje*[7].

En la figura 5.9 se muestra la topología de la red empleada. Para efectos de impresión se han eliminado conexiones, pero debe considerarse que todas las neuronas de una capa están interconectadas con todas las neuronas de la capa siguiente, a excepción del término de bias.

[6] Se interpretara como tiempo de procesador, el tiempo que consume el CPU en realizar los cálculos necesarios para entrenar la red.

[7] Entiéndase como grado de aprendizaje el grado de error remanente en la red entrenada, el cual dependiendo de la aplicación en cuestión tenderá a ser próximo a cero.



Topología de la red empleada
Figura 5.9

III. Entrenamiento de la Red

El algoritmo de Retropropagación del error va ajustando (actualizando) los pesos de las conexiones entre los elementos procesadores. Dicho ajuste es causado por los patrones con los cuales se entrena la red. Con los pesos actualizados lo que se define es el grado de actividad de cada neurona, por lo tanto lo que importa es obtener esos pesos. Para lograrlo, se tiene que entrenar la red, lo cual se hizo inicialmente con el programa *Cibernet*[9], que utiliza el primer archivo de patrones mencionado anteriormente.

En este programa se tiene que especificar el tipo de algoritmo a emplear, es decir, el algoritmo de Retropropagación del Error y la topología de la red (número de nodos de la capa de entrada, número de capas intermedias, número de nodos para cada capa intermedia y el número de nodos de la capa de salida). El programa entrega un archivo de extensión **.pes**, el cual contiene los pesos correspondientes para esa red con esos patrones en particular. El archivo está en formato de números flotantes de pascal, los cuales, para poder ser utilizados, se tienen que convertir a formato ASCII.

[9] Programa desarrollado por Giovanni Leonetti del Negro. Laboratorio de Cibernética. Facultad de Ciencias. Universidad Nacional Autónoma de México. México D.F. 1992.

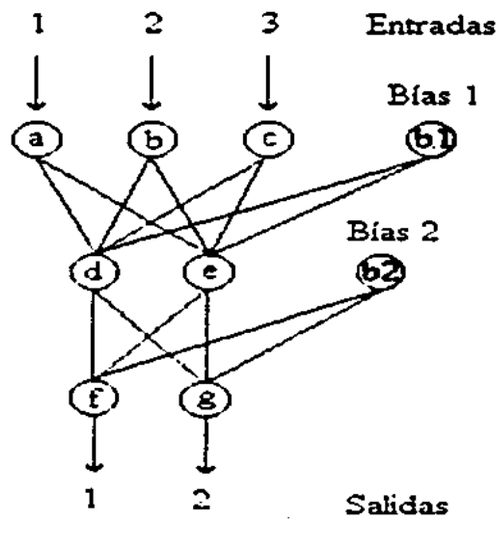
Esto se logra con el empleo del programa *Convert*[9], el que da como resultado un archivo bajo el siguiente esquema:

-1.111940770783
-1.111929950894
2.134968473605
2.134927923154
-5.939602632512
-5.939589004927
1.614010799634
1.614009016341

Como se puede observar, los valores vienen por pares, variando muy poco entre ellos El primer número del par indica el valor del peso anterior y el segundo indica el peso posterior a la iteración. Para el caso de la aplicación en particular, sólo interesa el último valor, por lo cual se descarta el primer valor.

El archivo de pesos obtenido, es empleado por el programa de aplicación, el cual va leyendo los valores y los va asociando a cada conexión entre neuronas. El programa inicialmente va tomando el primer valor del archivo de pesos y lo asocia a la conexión de la primera neurona de la capa de entrada con la primera neurona de la capa intermedia, siguiendo con el valor asociado al peso entre la primera neurona de la capa de entrada con la segunda neurona de la capa intermedia y así sucesivamente. En la figura 5.10 se ejemplifica este proceso.

[9] Programa desarrollado por el Ing. Alvaro Castiello. Facultad de Ingeniería. Universidad Nacional Autónoma de México. México D.F.



-1.111940770783	Peso neuronas a-d
-2.929950894322	Peso neuronas a-e
2.134968473605	Peso neuronas b-d
2.423569239876	Peso neuronas b-e
-5.939602632512	Peso neuronas c-d
-7. 900492939583	Peso neuronas c-e
1.614010799634	Peso neuronas b1-d
1. 901661403410	Peso neuronas b1-e
-1.111940770783	Peso neuronas d-f
5.349431650894	Peso neuronas d-g
2.134968473605	Peso neuronas e-f
-2. 23154149279	Peso neuronas e-g
-5.939602632512	Peso neuronas b2-f
-5. 900492723154	Peso neuronas b2-g

Ejemplo de cómo se van asociando los pesos a las conexiones entre neuronas.
Figura 5.10

El programa de aplicación procesa los patrones de entrada, siendo interpretados para posteriormente permitir el movimiento del móvil mecánico.

El total de valores de pesos que se requiere para el total de las neuronas se obtiene mediante la ecuación 5.2.

$$(nnce + 1)(nnci) + (nnci + 1)(nncs) \qquad (5.2)$$

donde:

nnce = número de neuronas de la capa de entrada
nnci = número de neuronas de la capa intermedia
nncs = número de neuronas de la capa de salida

En la ecuación anterior, se puede observar que aparece una constante unitaria. Esto se debe al término de bias, el cual fue explicado en el capítulo 4.

Se comenzó a trabajar con los archivos generados por el programa Cibernet y se descubrió un error. El programa genera menos pesos de los que debe, motivo por el cual las pruebas realizadas no eran del todo certeras y confiables. Fue por eso que se decidió cambiar la herramienta de entrenamiento eligiendo al programa comercial DynaMind *[4]*, siendo éste el motivo por el que se cambió de programa generador de patrones (ya visto en el punto 1).

El archivo de patrones todavía tiene que sufrir algunas modificaciones para poder ser reconocido por el programa DynaMind. En la figura 5.11 se muestra su estructura.

```
-1.00 -1.00 -1.00  1.00 -1.00 1.00 -1.00 -1.00 1.00 1.00  1.00  1.00
 1.00  1.00 -1.00 -1.00 -1.00 1.00 -1.00 -1.00 1.00 1.00  1.00  1.00
```

Estructura del archivo de patrones requerida por DynaMind.
Figura 5.11

Se puede ver que a diferencia de la estructura mostrada en la figura 5.8 sólo cambia en que se han eliminado las cabeceras del número total de entradas y el número total de salidas.

Una vez obtenido este archivo se tiene que construir el archivo que va a reconocer DynaMind. Esto se logra mediante un programa llamado **iobuild** (incorporado en DynaMind) mediante la siguiente sintaxis:

iobuild [archivo de patrones de entrada.pat] [archivo de entrada.io]

El cual entrega un archivo como el mostrado en la figura 5.12.

```
* prueba1
* Number of inputs:
8
* Number of outputs:
4

* pair 1
@ patron1
-1.000000 -1.000000 1.000000 -1.000000 -1.000000 -1.000000 1.000000
1.000000 1.000000 -1.000000 1.000000 1.000000

* pair 2
@ patron2
-1.000000 -1.000000 1.000000 -1.000000 -1.000000 -1.000000 1.000000
1.000000 1.000000 -1.000000 1.000000 1.000000

* The End of I/O Data Patterns.
```

Esquema del archivo de patrones generado por el programa iobuild.
Figura 5.12

Este es el archivo que DynaMind tomará para efectuar el entrenamiento de la red bajo el algoritmo empleado.

Para detalles de las características de los archivos de entrada al DynaMind y los archivos que entrega, ver la referencia 4 de la bibliografía del presente capítulo.

El archivo que entrega DynaMind una vez realizado el entrenamiento contiene los pesos de las interconexiones entre neuronas y presenta la extensión ***.net** . Estos pesos son los que emplea el programa de aplicación.

A continuación se muestra el formato en que entrega el archivo de patrones el programa DynaMind, tomando como ejemplo la red mostrada en la figura 5.10.

línea 1	2 3 2 0
línea 2	patrones
línea 3	Layer 0
línea 4	2 3 1.000
línea 5	1.000 0.900
línea 6	0.0135 1.1212 1.0000 0. 2727
línea 7	-0.2274 -0.0826 0.1927
línea 8	0.6042 -0.8176 -0.8187
línea 9	-0.2886 0.7339
línea 10	Layer 1
línea 11	2 3 1.000
línea 12	1.000 0.900
línea 13	1 12 1 4 1.000 0.300
línea 14	-1.7058 3.1806
línea 15	-2.2759 -3.3909
línea 16	0.5544 1.8590
línea 17	0 0
línea 18	EXTIO
línea 19	External In
línea 20	0 0 3
línea 21	-1
línea 22	External Out
línea 23	1 0
línea 24	-1

El programa entrega el listado mostrado en la segunda columna.

La línea 1 indica que va a ser una red de 2 capas (DynaMind considera solamente las capas intermedias y las de salida), la segunda línea es una etiqueta, la línea 3 indica que comienzan los valores para la primera capa intermedia (nivel 0), las líneas 4, 5 y 6 son valores relacionados a la capa intermedia, las líneas 7 y 8 son los valores de los pesos correspondientes a la primera neurona de la capa intermedia con la primera neurona de la capa de entrada, siguiendo con la primera neurona de la capa intermedia con la segunda neurona de la capa de salida y así sucesivamente.

La línea 9 contiene los pesos relacionados a cada una de las neuronas de la capa intermedia con la neurona de bias.

Lo mismo hay que desarrollar para la segunda capa (capa de salida).

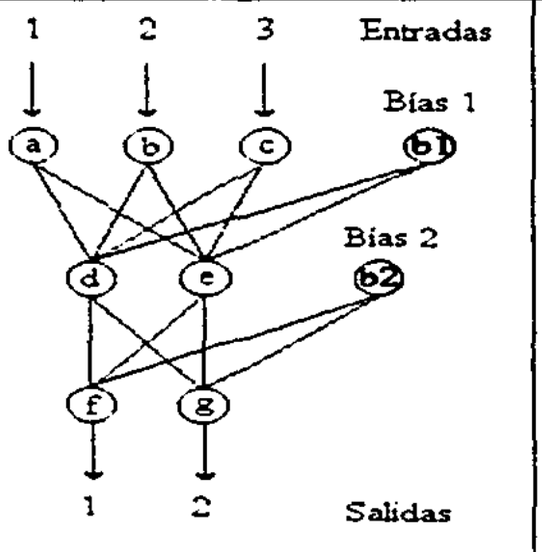
Para que este archivo pueda ser interpretado por el programa de aplicación, debe de sufrir las siguientes modificaciones:

1. Agregar antes de la línea 1 dos líneas más, la primera conteniendo el total de entradas y la segunda conteniendo el total de salidas.
2. Eliminar las líneas 1 a la 6.
3. Mover el primer valor de la línea 9 al final de la línea 7.
4. Mover el segundo valor de la línea 9 al final de la línea 8.
5. Eliminar las líneas 9 a la 13.
6. Mover el primer valor de la línea 16 al final de la línea 14.
7. Mover el segundo valor de la línea 16 al final de la línea 15.
8. Eliminar las líneas de la 16 a la 24.

Hecho lo anterior el archivo de pesos queda de la siguiente manera:

```
3
2
-0.2274 -0.0826 0.1927 -0.2886
0.6042 -0.8176 -0.8187 0.7339
-1.7058 3.1806 0.5544
-2.2759 -3.3909 1.8590
```

El programa de aplicación realiza la asignación de pesos mostrada en la figura 5.13



-0.2274	Peso neuronas a-d
-0.0826	Peso neuronas b-d
0.1927	Peso neuronas c-d
-0.2886	Peso neuronas b1-d
0.6042	Peso neuronas a-e
-0.8176	Peso neuronas b-e
-0.8187	Peso neuronas c-e
0.7339	Peso neuronas b1-e
-1.7058	Peso neuronas d-f
3.1806	Peso neuronas e-f
0.5544	Peso neuronas b2-f
-2.2759	Peso neuronas d-g
-3.3909	Peso neuronas e-g
1.8590	Peso neuronas b2-g

Ejemplo de asignación de pesos a partir del archivo entregado por DynaMind.
Figura 5.13

Para el entrenamiento se hicieron varias pruebas variando los patrones empleados y las características y valores de parámetros de entrenamiento en el DynaMind.

En la tabla 5.3 se indica el número de cada patrón empleado en el entrenamiento final. Para ver detalles del patrón consultar el apéndice A.

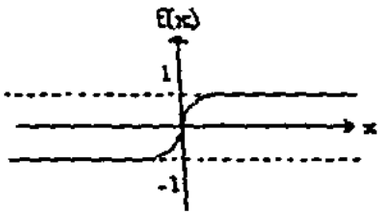
Símbolo	Número del patrón empleado.
Flecha hacia arriba	1,2,3,4,5
Flecha hacia abajo	38,39,40,41,42
Ejecutar	75,88,89,90
Parar	93,94
Vuelta	208,209,210,211
Dos repeticiones	103,104,105,163
Tres repeticiones	112,113,155,159,161,199
Cuatro repeticiones	120,123,128
Cinco repeticiones	145,146,206,207

Relación de patrones del entrenamiento final.
Tabla 5.3

En relación a los valores y características de los parámetros de entrenamiento solicitados por el DynaMind, se tiene que el programa para una red trabajando bajo el algoritmo de Retropropagación del error, pide los siguientes parámetros:

- Número de capas (intermedias y de salida), que para nuestro caso fueron:
  - 1 Capa intermedia.
  - 1 Capa de salida
- El número de neuronas de entrada, neuronas de la capa intermedia y neuronas de la capa de salida, que para la presente aplicación fue:
  - Neuronas de la capa de entrada: 35
  - Neuronas de la capa intermedia: 12
  - Neuronas de la capa de salida: 4

- Los límites de la función de transferencia, que para lograr una mayor separación y rango de acción se seleccionó que fueran de -1 a 1 como se muestra en la figura 5.14.



Función de transferencia empleada, mapeada a un rango de -1 a 1.
Figura 5.14

- La ganancia de la función de transferencia, tomándose el valor por default que proporciona el programa, es decir un valor de uno.
- El Learning Rate aceptando después de varias pruebas el valor por default del programa, es decir un valor de 0.2.
- Generación aleatoria de los pesos iniciales.
- Fluctuación de los pesos de la red fijándola a un valor de 5.
- Control de nivel de ruido. Recordemos que al agregar un nivel de ruido se ayuda a que la red pueda generalizar. Se seleccionó un nivel de ruido de 0.1.
- Condiciones de término de entrenamiento. Se optó que el entrenamiento terminara al alcanzar las 20000 iteraciones para lograr que el error disminuyera lo suficiente.

IV. Prueba y uso de la red (implementación).

La prueba de la red se hace una vez que se tiene el archivo entregado por DynaMind, archivo que hay que modificar un poco para ser compatible con el programa de aplicación.

Para probar y usar el archivo de pesos entrenados, es necesario que se implemente el sistema de aplicación, el cuál se muestra en la figura 5.15.

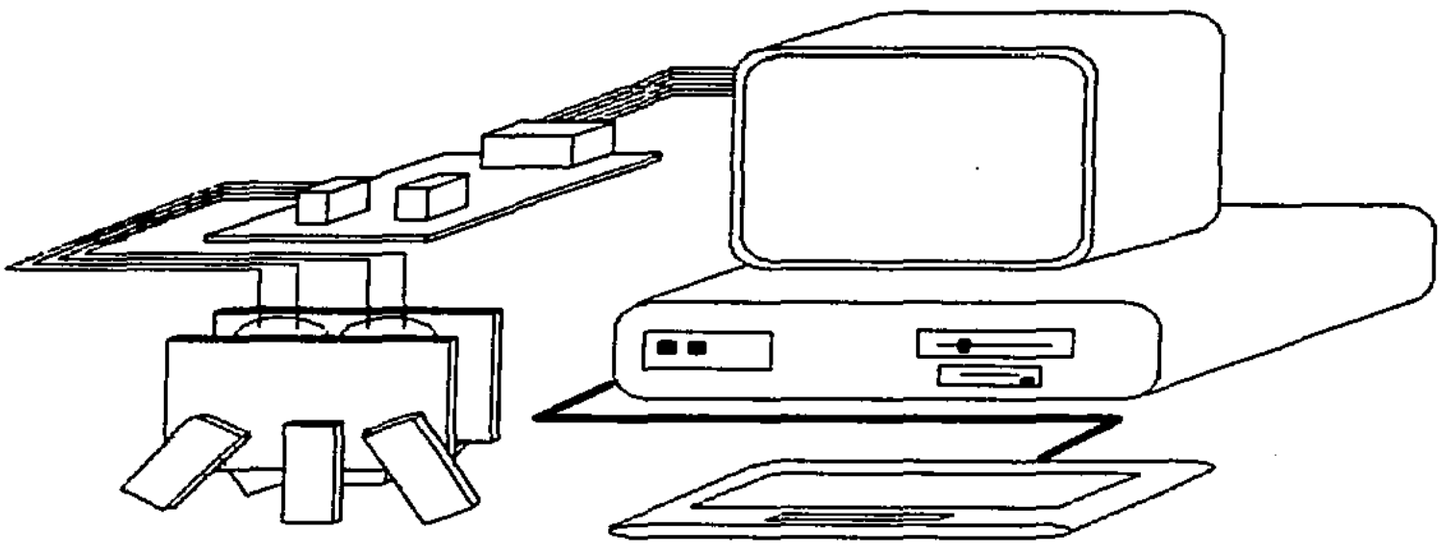
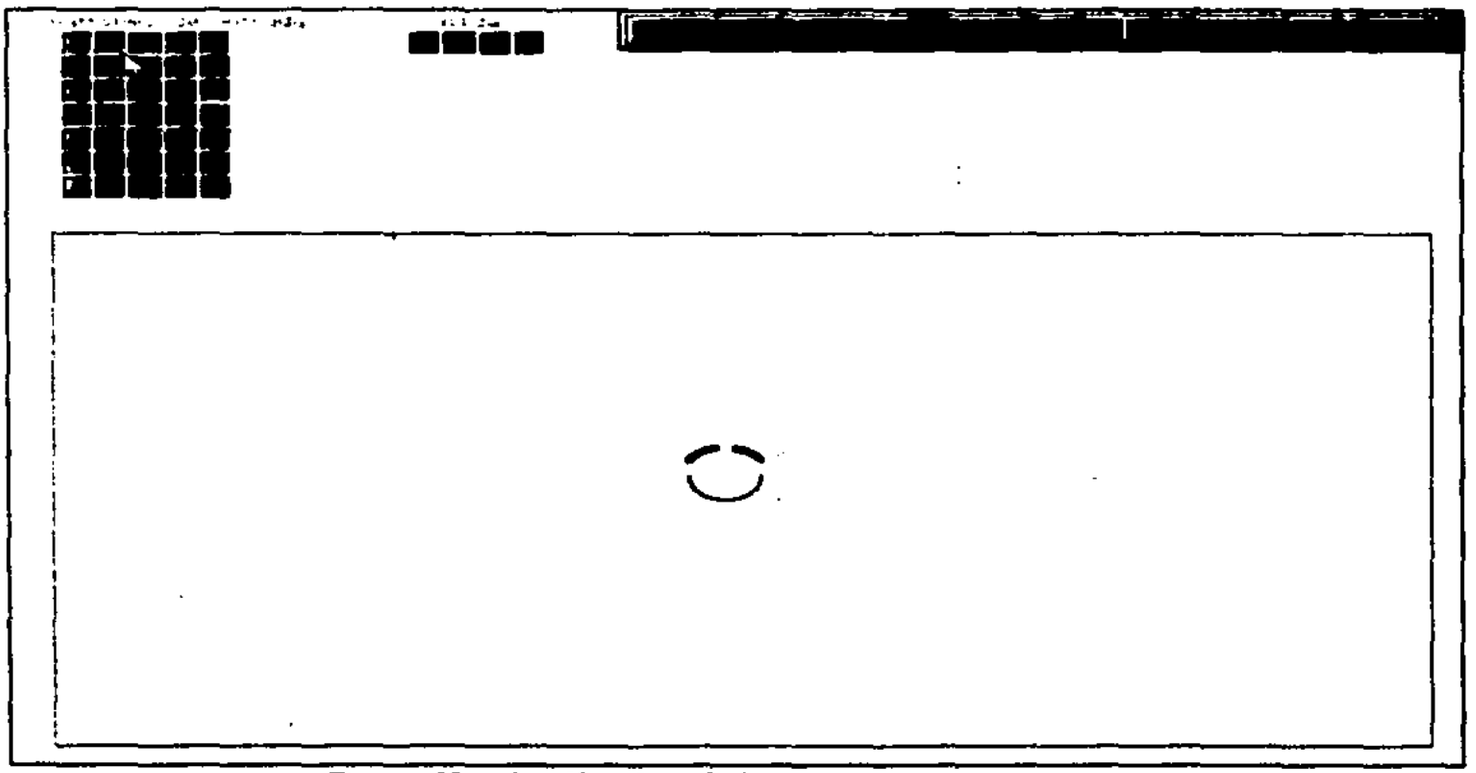


Ilustración del sistema de aplicación
Figura 5.15

El Software de aplicación consta principalmente de dos programas:

- patron.exe
- robot.exe

El programa patrón.exe es el que brinda la interface necesaria para ir dibujando los patrones (siendo esta su única función), guardándolos en un archivo bajo el formato mostrado en la figura 5.8. En la figura 5.16 se muestra la pantalla de diseño.



Pantalla de diseño del programa patron.exe
figura 5.16

En esta pantalla consta de varias secciones tales como el área de captura, la barra de herramientas y el área de visualización, las cuales son comunes tanto al programa patron.exe como al programa robot.exe (el área de visualización en el programa robot.exe es un poco diferente).

Para su uso hay que seguir los siguientes pasos:

Indicarle al programa que se va a comenzar la captura de patrones, para lo cual se selecciona la opción **Programar** de la barra de herramientas. Cada vez que se dibuje un patrón con su correspondiente salida, dar **Aceptar** con lo cual el área de captura de los Patrones de entrada y la salida se borran, permitiendo introducir un nuevo patrón. Para indicarle al programa que el proceso de captura ha terminado, se da **Esc**, con el ratón se selecciona el botón de **Salir** o se pulsa la letra **S**.

Ahora bien, para comprobar los patrones introducidos se realiza lo siguiente:

Mediante la opción **Ejecutar**, van apareciendo los patrones introducidos junto con su salida correspondiente, lo cual permite revisar el trabajo realizado.

El programa de aplicación también permite guardar, abrir, o borrar un archivo de patrones. Para ello basta con seleccionar la opción **Archivos** de la barra de herramienta para que aparezca la ventana correspondiente.

En cuanto al funcionamiento del programa robot.exe, básicamente es el mismo que para el programa patron.exe, con la diferencia de que al seleccionar la opción **Programar** de la barra de herramientas, solo hay que introducir el patrón de entrada (sin su salida). De igual forma que en el programa patron.exe hay que seleccionar el botón **Aceptar** para introducir otro patrón. Al seleccionar la opción **Ejecutar**, lo que sucede es lo siguiente:

Análogamente al programa patron.exe aparecen los patrones introducidos uno tras otro. Al término de esto, el símbolo mostrado en el área de visualización comienza a ejecutar los movimientos indicados. Los cuadros que rodean ésta área se van encendiendo (cambiando de color gris a rojo) indicando la dirección que sigue el símbolo. Al terminar de ejecutar los movimientos, la carita cambia de color amarillo a color rojo, para luego regresar a su color normal.

En este programa también podemos abrir, guardar o borrar un archivo mediante la opción **Archivo** de la barra de herramientas.

Además de presentar el resultado en pantalla, el programa robot.exe entrega una señal externa por medio del puerto paralelo de la computadora en la que se esté ejecutando el programa, dicha señal se pasa a la etapa de potencia y de ésta al móvil mecánico (procedimiento ya explicado), con lo cual se logra que dicho móvil realice los movimientos mostrados por el programa de aplicación.

Ejemplos de aplicación del programa de aplicación robot.exe

Ejemplo para la ejecución de un archivo conteniendo patrones ideales.

Primero se abre el archivo de los patrones deseados que en nuestro caso es el archivo que se llama exper.pat. Para abrirlo, sólo se selecciona **Archivo**, luego se elige la opción de **Abrir** y en la ventana que se habilita, se escribe sólo el nombre del archivo, es decir, sin extensión. Esto se muestra en la figura 5.17.

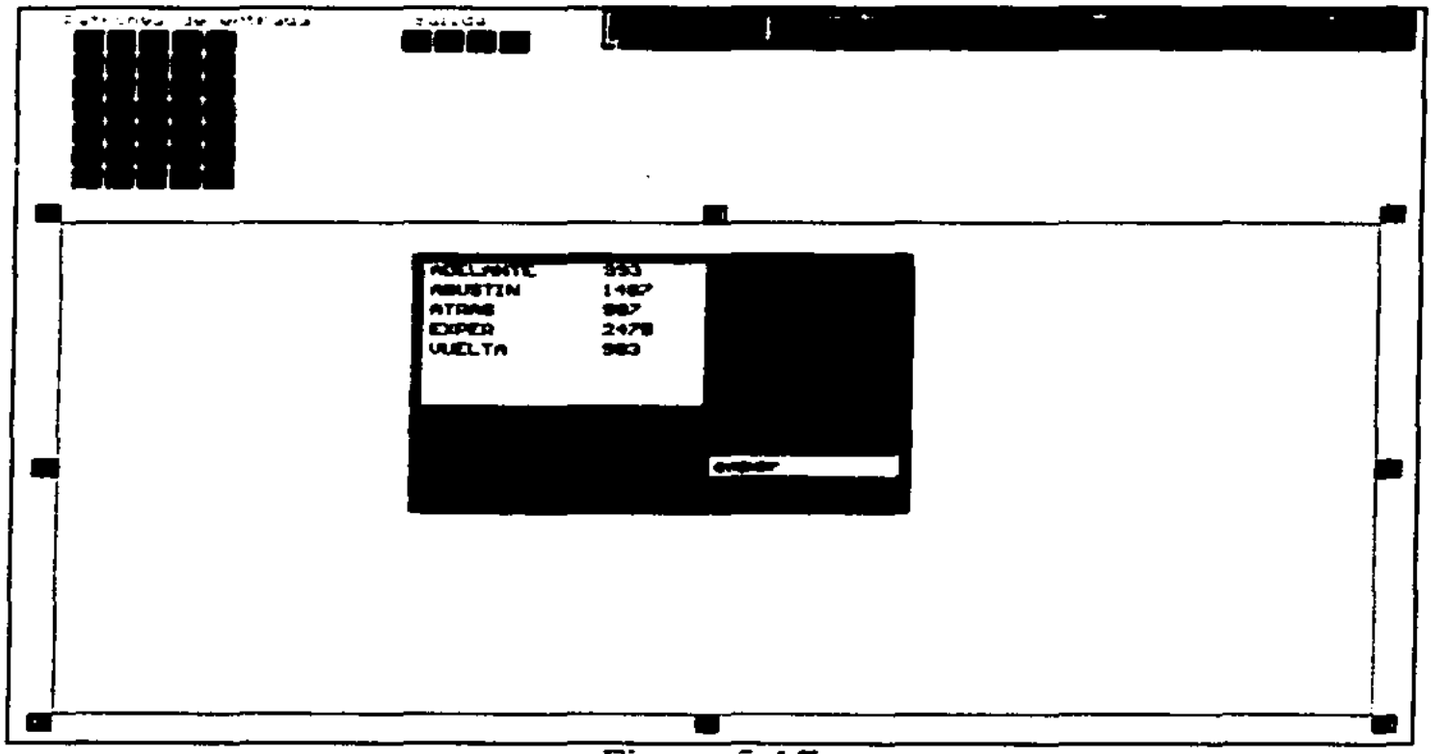
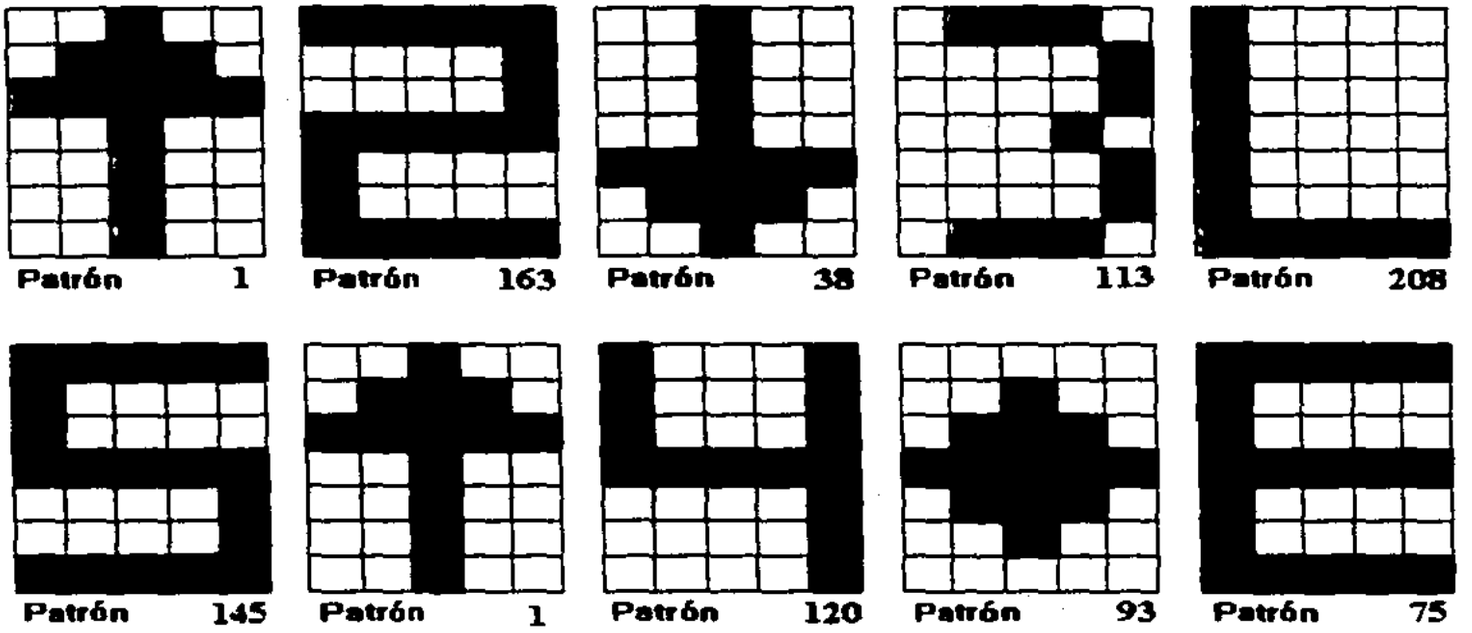


Figura 5.17

Una vez que se abre el archivo, el programa empieza a leer cada uno de los patrones que tiene el archivo, pasándolo por la red ya entrenada, y dependiendo del patrón, la red va a entregar la respuesta, que será el movimiento que tenga que realizar tanto el móvil mecánico (si está conectado al puerto paralelo de la computadora) como la imagen que está en pantalla.

Al momento de leer el patrón de la letra E, sabe que es el momento de empezar a ejecutar cada uno de los patrones que leyó anteriormente.

El archivo de patrones tomado como ejemplo, consta de los movimientos mostrados en la figura 5.18.



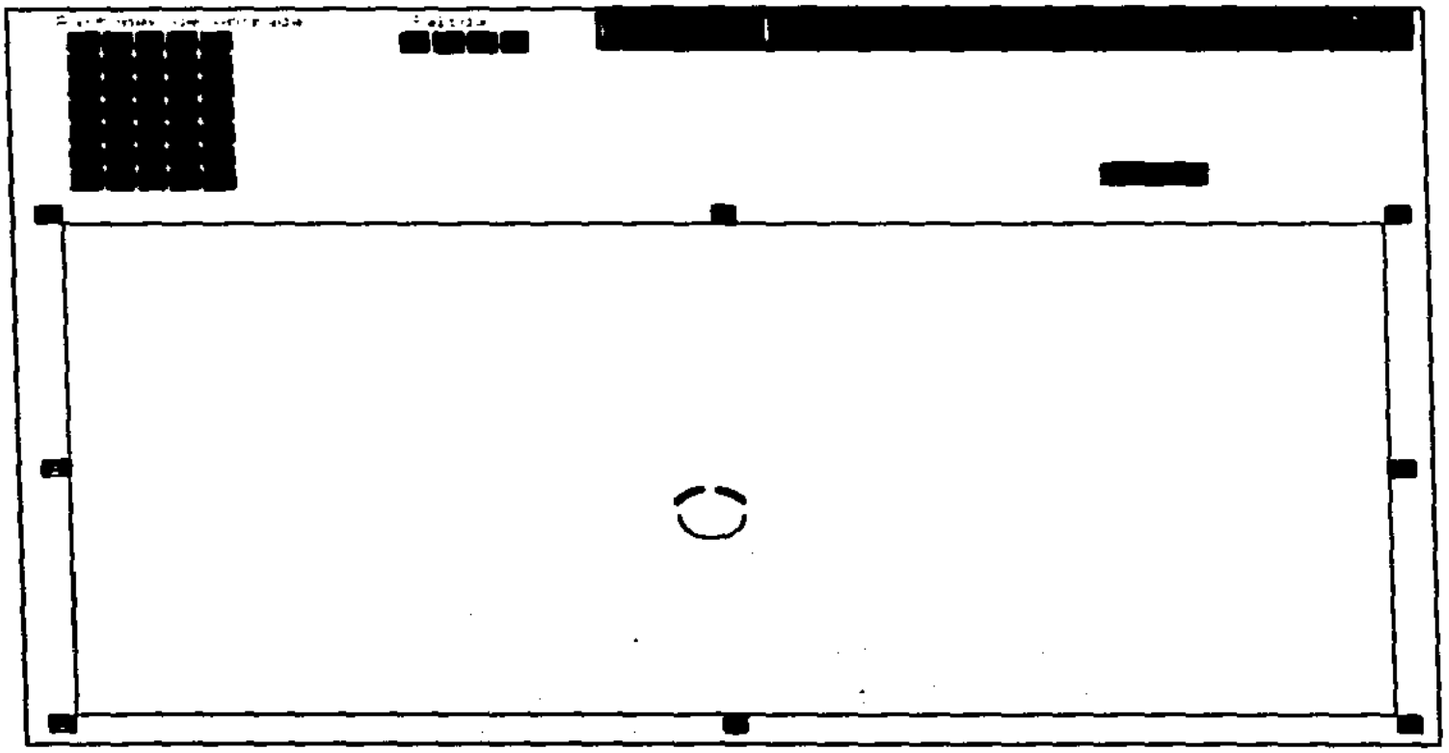
Patrones empleados para el primer entrenamiento.
Figura 5.18

Es decir, el móvil se mueve hacia adelante durante dos segundos, luego retrocede durante tres segundos, da 5 giros de 45 grados en sentido antihorario, y por último camina hacia adelante durante cuatro segundos. Al encontrar el patrón de fin de ejecución, deja de realizar cualquier acción e indica en la pantalla con un cambio en el color de la imagen que se terminó de ejecutar el archivo de patrones.

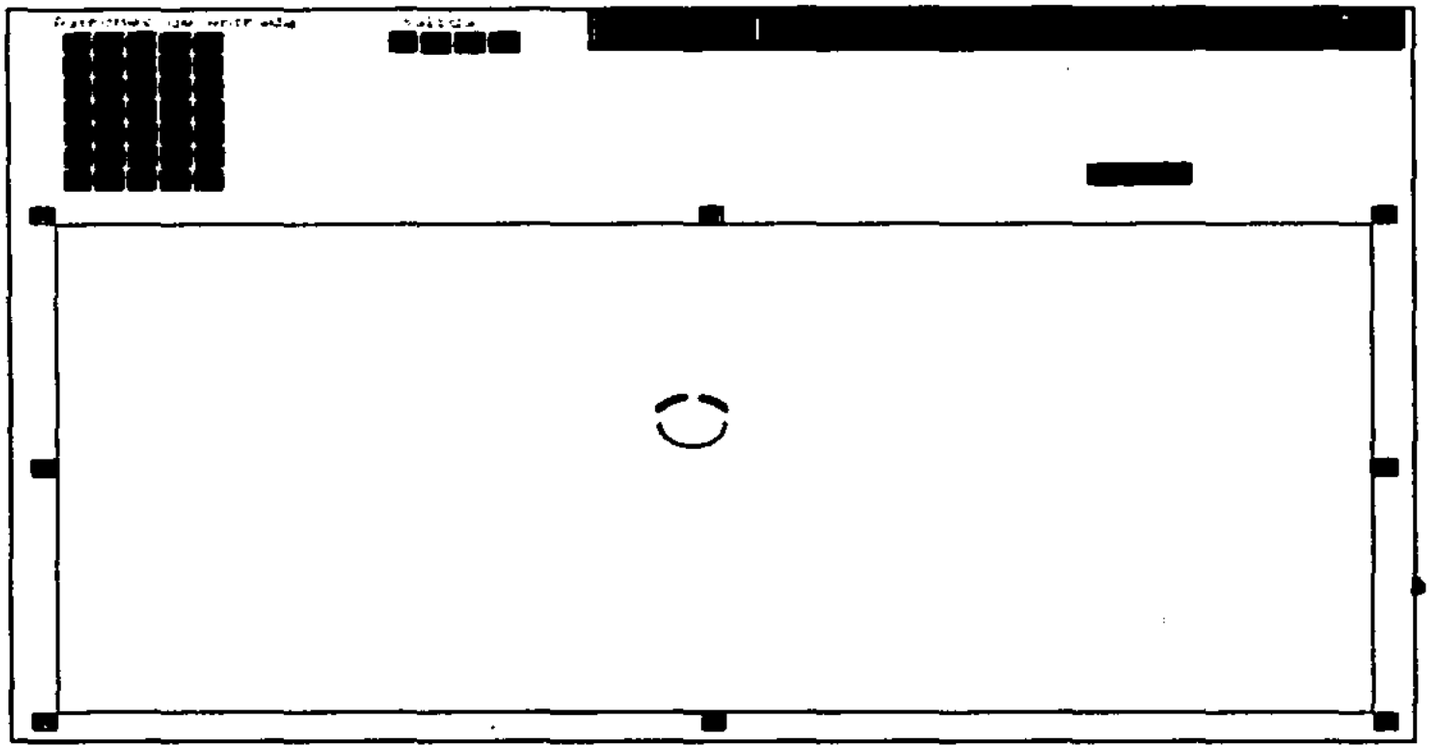
Por cada segundo de duración de algún movimiento, en la pantalla la imagen realiza un movimiento, es decir, si el robot se mueve un segundo hacia adelante, la imagen se mueve una vez hacia donde indique en la pantalla que es adelante.

Para saber el sentido en el que se mueve la imagen, en la pantalla principal se tienen 8 cuadros alrededor del marco principal que corresponden a los cuatro movimientos básicos: Arriba, Abajo, Derecha e Izquierda; y a las combinaciones de estos: Arriba a la Derecha, Abajo a la Derecha, Arriba a la Izquierda y Abajo a la Izquierda. El cuadro que está en rojo indica que hacia ese lado se considera adelante, y cuando el móvil está girando, el cuadro en rojo cambia hacia donde se gira la imagen.

Pantalla que presenta el primer movimiento de dos segundos avanzando hacia adelante.

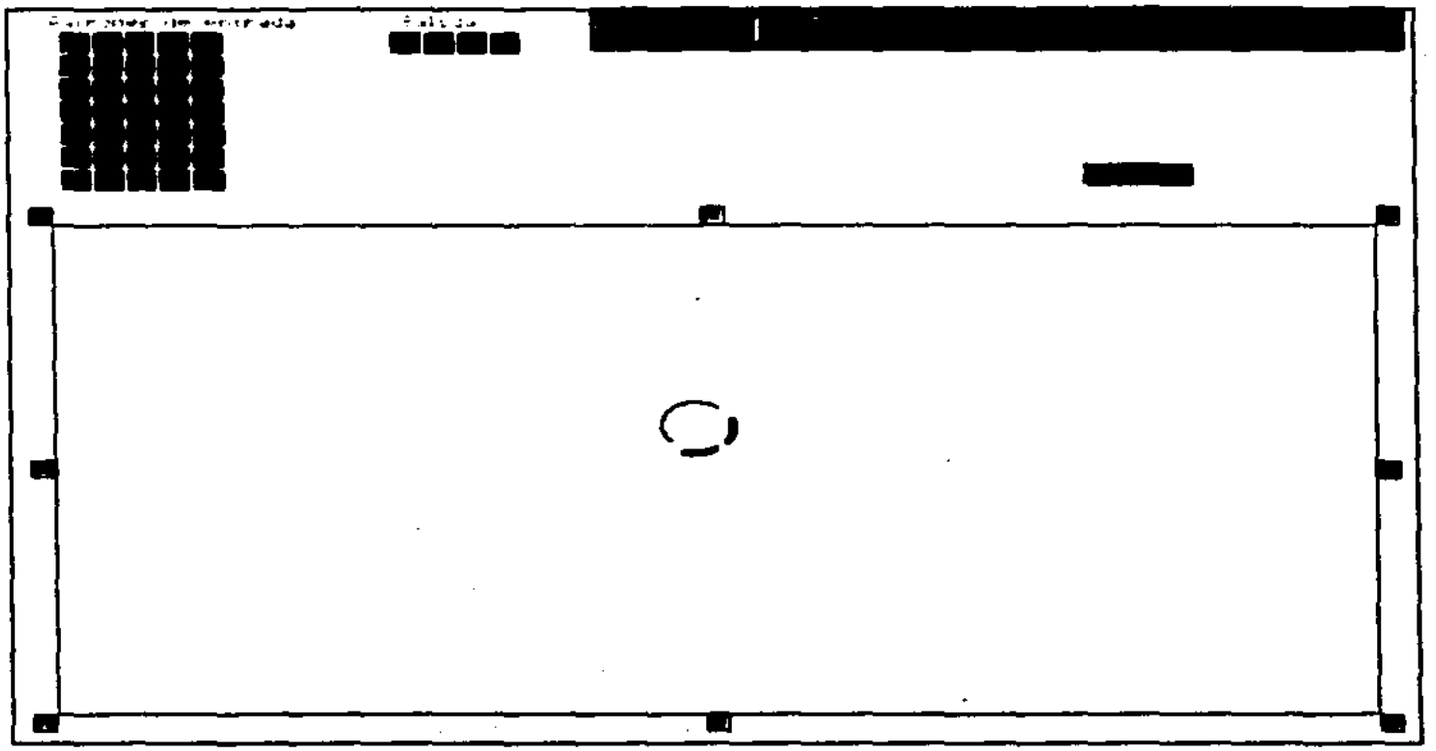


Pantalla que presenta los tres segundos de movimiento hacia atrás.

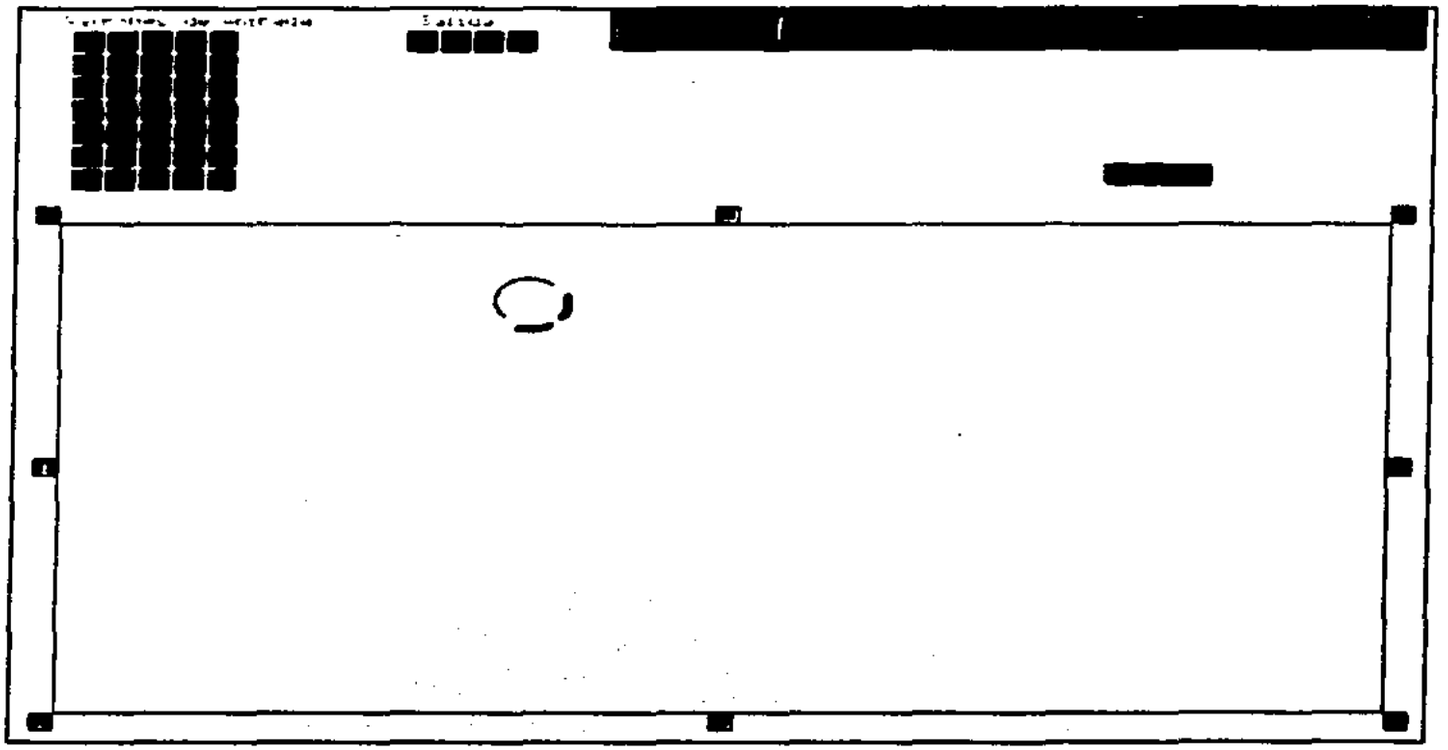


ESTA TESIS NO DEBE
SALIR DE LA BIBLIOTECA

Pantalla que presenta la imagen después de los cinco giros de 45 grados en sentido antihorario.



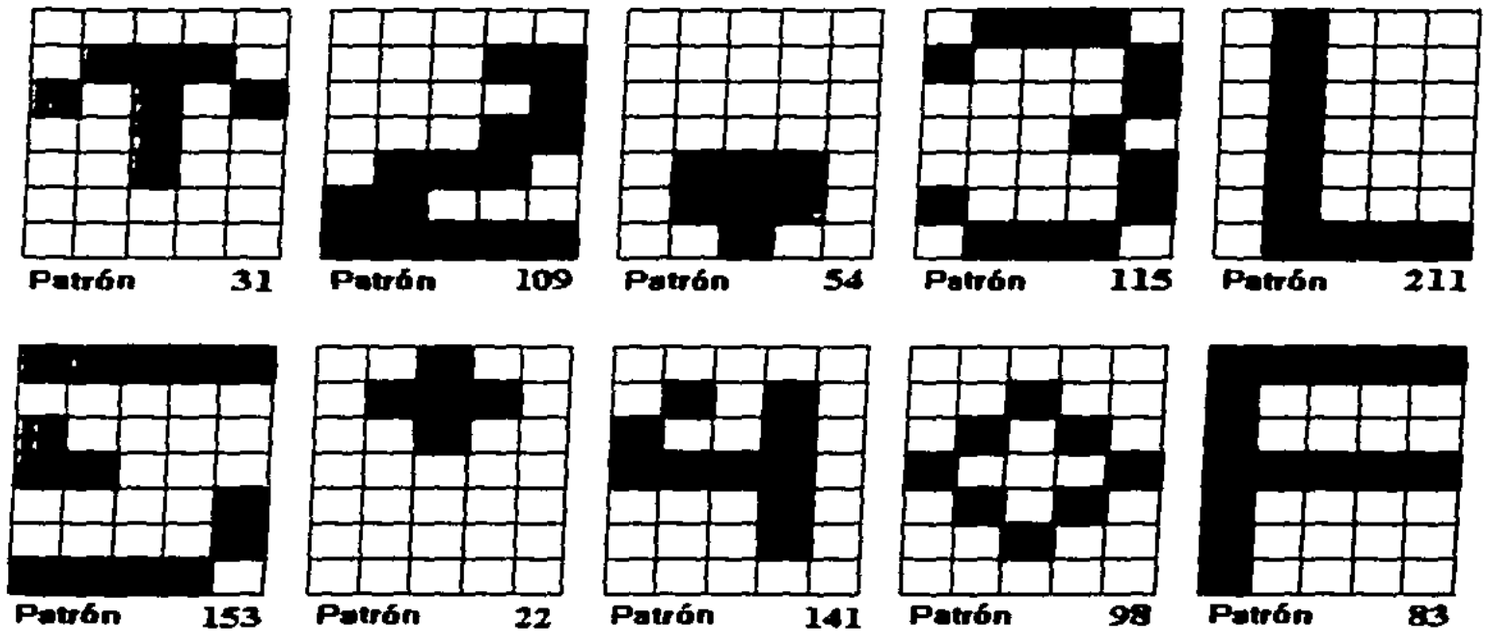
Pantalla que presenta el movimiento de avance hacia adelante durante cuatro segundos y posición final de la imagen.



Ejemplo para la ejecución de un archivo conteniendo patrones ruidosos.

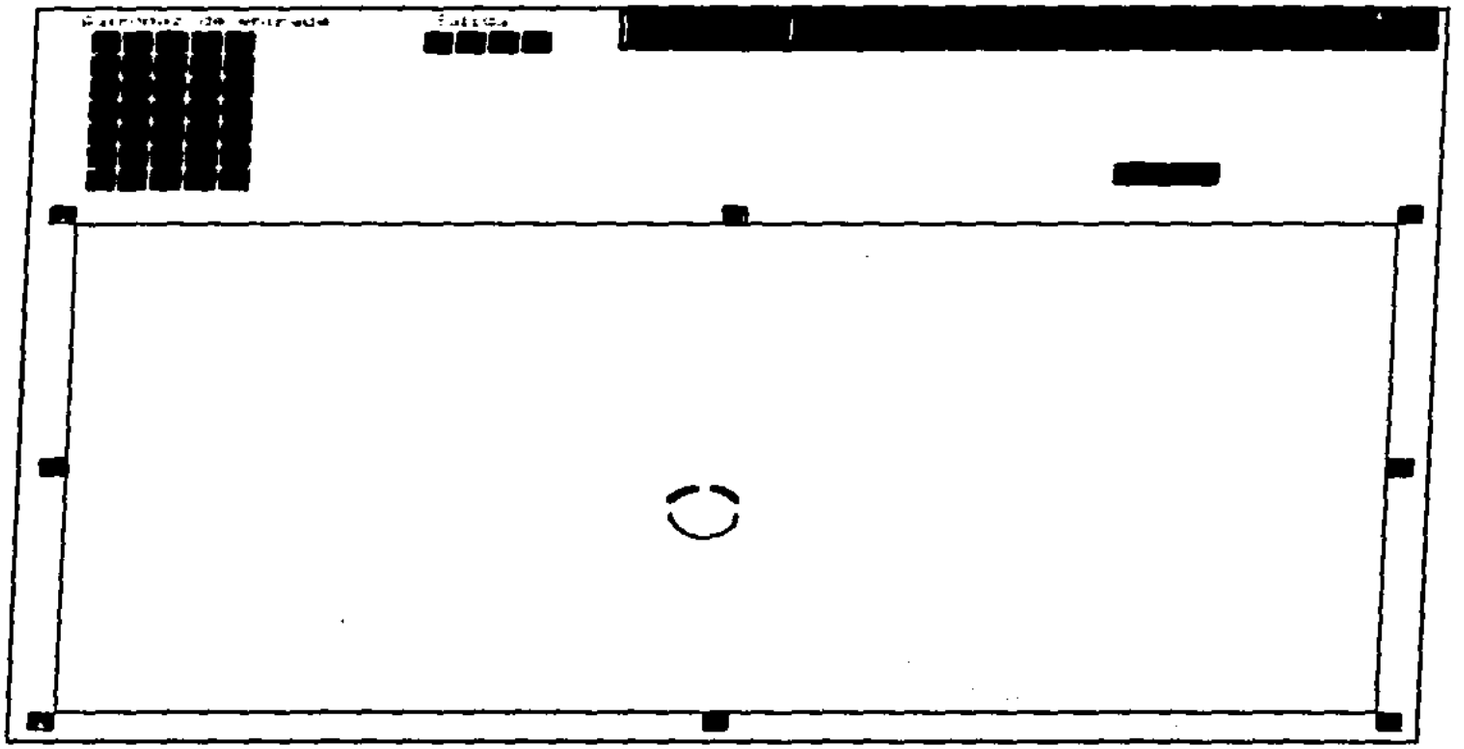
Se considerará los mismos movimientos que para el ejemplo anterior, solo que ahora se considerarán los patrones ruidosos de los mismos.

El archivo de patrones tomado como ejemplo, consta de los movimientos mostrados en la figura 5.19.

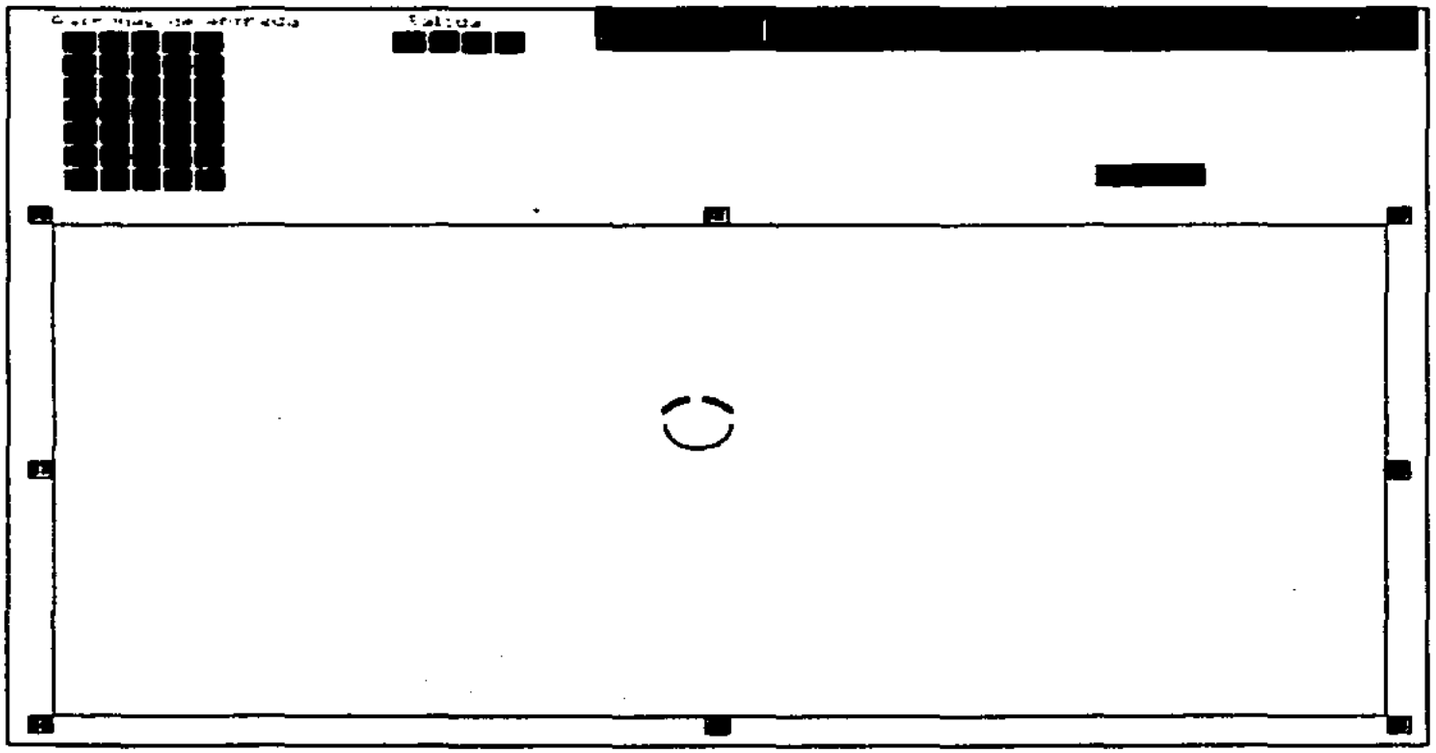


Conjunto de patrones ruidosos de prueba.
Figura 5.19

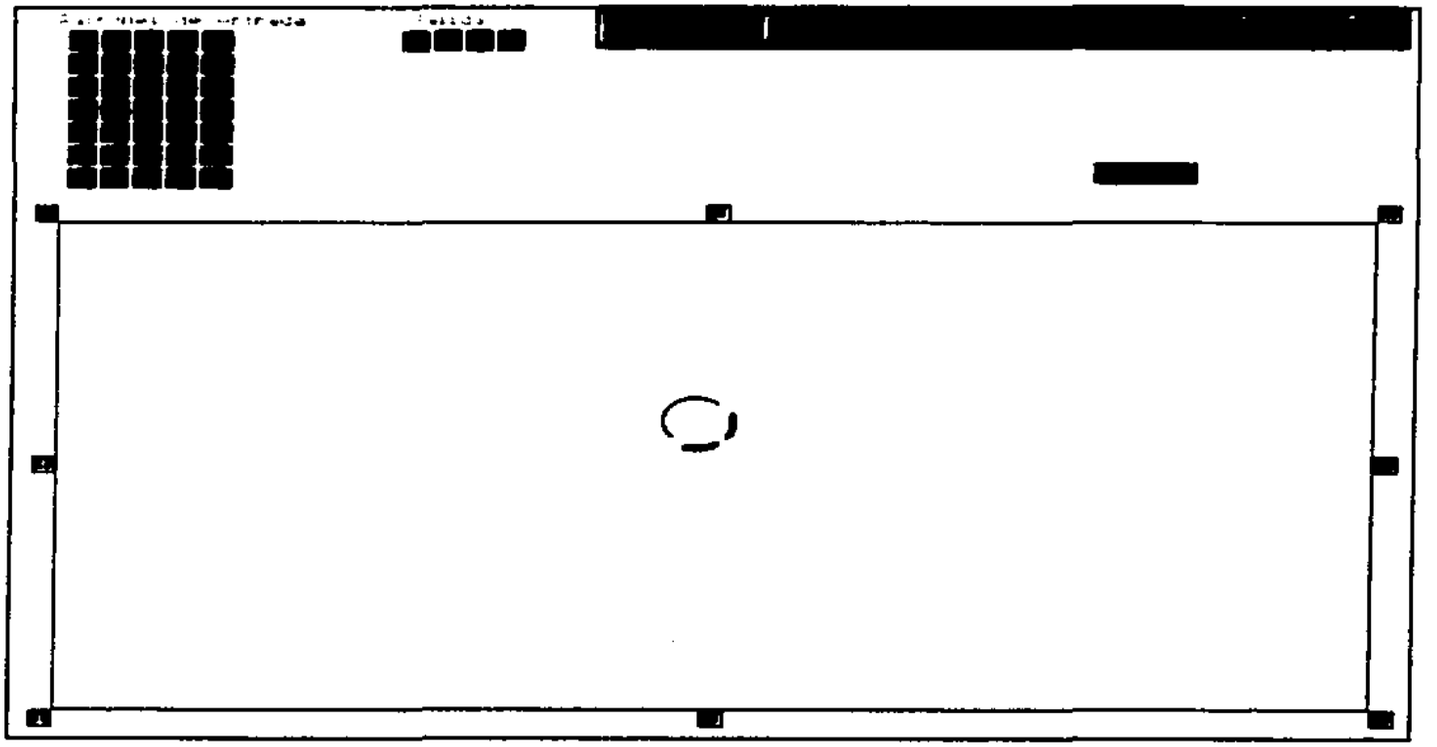
Pantalla que presenta el primer movimiento de dos segundos avanzando hacia adelante.



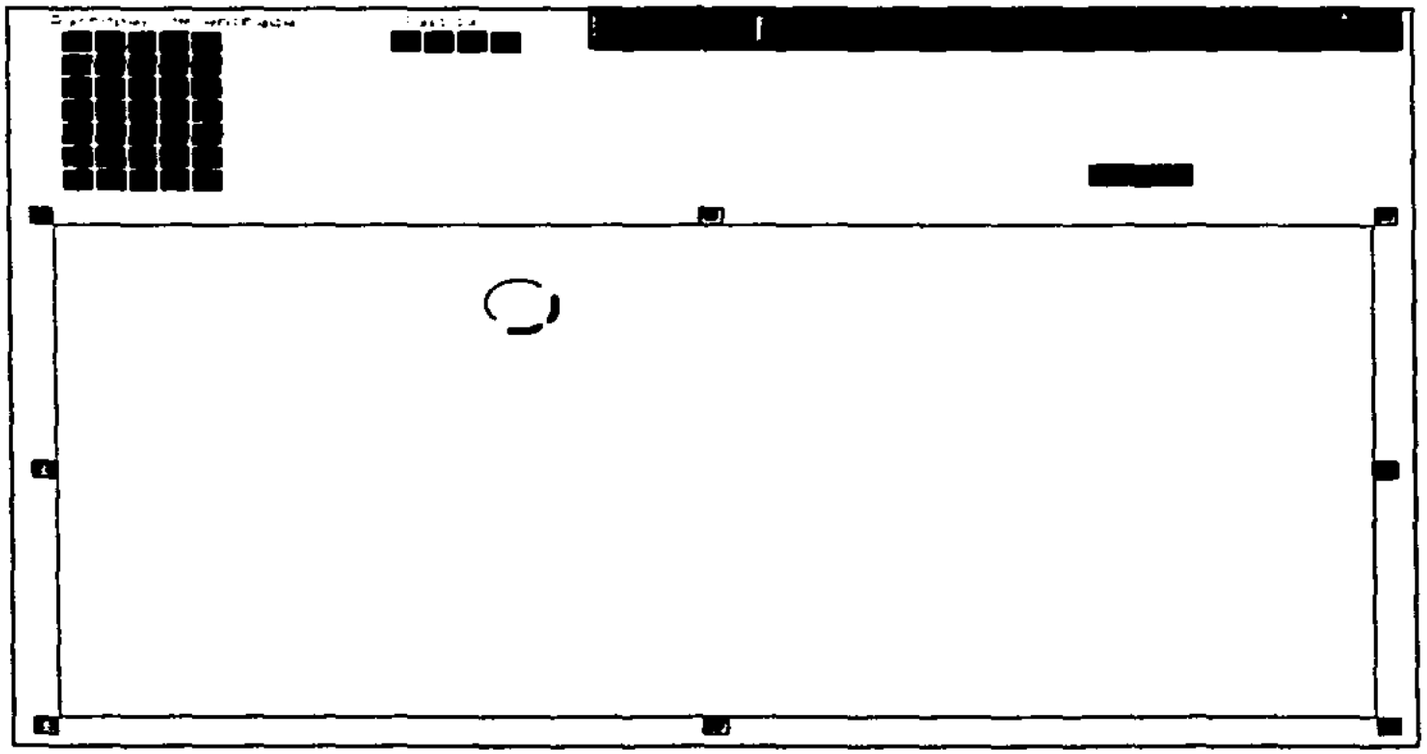
Pantalla que presenta los tres segundos de movimiento hacia atrás.



Pantalla que presenta la imagen después de los cinco giros de 45 grados en sentido antihorario.



Pantalla que presenta el movimiento de avance hacia adelante durante cuatro segundos y posición final de la imagen.



Ante la entrada de los patrones de entrenamiento ruidosos, la red es capaz de reconocerlos y asociarlos a su correspondiente movimiento, es decir, puede distinguir los patrones a pesar de que fueron ruidosos y de que no fueron contemplados todos en el grupo de patrones de entrenamiento, con lo cual se puede ver que la red neuronal se encuentra trabajando.

Bibliografía

Kröse, Ben & van der Smagt, Patrick. "*An introduction to Neural Networks*". University of Amsterdam. Netherland. 1993.

Leonetti del Negro, Giovanni. "*Simulador de Redes Neuronales Artificiales*". Tesis de Licenciatura. Universidad Nacional Autónoma de México. México D.F. 1992.

Guevara, José y Rivera, Luis. "*Reconocimiento de firmas utilizando redes neuronales artificiales*". Tesis de Licenciatura. Universidad Nacional Autónoma de México. México D.F. 1995.

Wulms, Alex. " *Project Study Neural Networks. On the dynamic behaviour of backpropagation networks*". St. Nr. 8949670. October 1994.

"*DynaMind. Developer User's Guide*". Neuro DynamX, Boulder, Co. U.S.A. 1992.

Vysniauskas, Vytautas. Groen, Frans. Kröse, Ben. "*The optimal number of learning samples and hidden units in function approximation with a feedforward network*". Technical Report CS-93-15. University of Amsterdam. Netherland, 1993

Gori, M. and Maggini, M. "*Optimal Convergence of On-Line Backpropagation*". Dipartimento di Sistemi e Informatica. Università di Firenze. Italia. http://www-dsi.ing.unifi.it

Bianchini, M and Gori, M. "*Optimal learning in artificial networks: a theorical view*". Dipartimento di Sistemi e Informatica. Università di Firenze. Italia.
http://www-dsi.ing.unifi.it

"*HP NetServer LF. Reference Guide*". Hewlett Packard, 1994.

Conclusiones

Conclusiones

Si bien es cierto que los programas simuladores, circuitos integrados e incluso los algoritmos existentes hasta la fecha distan mucho de simular el comportamiento del cerebro, también es cierto que ninguna ciencia se ha desarrollado de la noche a la mañana y que el estudio de las Redes Neuronales Artificiales se ha visto frenado por un sin fin de obstáculos, ya sean económicos o incluso intelectuales. A pesar de ello los grupos dedicados al estudio de las Redes Neuronales no se han quedado de brazos cruzados. Se han desarrollado algoritmos muy poderosos, capaces de atacar ciertos problemas y tareas muy específicas no solo del ser humano, si no de los seres vivos en general.

Con el desarrollo que se ha tenido en las últimas décadas, es natural que se haya empleado la tecnologia computacional como apoyo para la investigación en las Redes Neuronales. Por este motivo la mayor parte de los trabajos desarrollados hasta la fecha han sido a nivel software. A pesar de ello y de que son varios los productos que contienen esta tecnología y que son empleados "en la vida diaria", como se puede ver en la lista de empresas mostrada en el apéndice D, esta tecnología sigue siendo hasta cierto punto incomprensible e ignorada por la mayoría de las personas y, en algunos casos, al no entenderse es rechazada.

Hasta la fecha ya se han desarrollado varios trabajos sobre el tema, la mayoria de ellos son simuladores o entrenadores de Redes Neuronales trabajando bajo diversos algoritmos. Estos trabajos son herramientas invaluables para el desarrollo de las Redes Neuronales, pero aún asi no dejan claro en su totalidad los conceptos que, aunque son muchos, no son imposibles de explicar. Es absurdo que aún dentro del área ingenieril esta tecnologia se desconozca. Esto es lo que motivó la realización de la presente tesis, cuyo objetivo planteado fué mostrar un ejemplo didáctico del empleo de la tecnología de las Redes Neuronales Artificiales.

Para cubrir este objetivo se desarrolló el programa de aplicación en el que si bien, no se trata de descubrir el hilo negro, se basa en un algoritmo ampliamente estudiado. Tampoco se pretende hacer una copia de tantos trabajos e investigaciones realizados con anterioridad, por el contrario, se

quiso aprovechar todos esos conocimientos heredados y conjuntarlos en una aplicación sencilla pero no por eso menos importante.

El programa al estar dividido en módulos, nos brinda una importante base de desarrollo flexible para trabajos futuros, pues sus módulos pueden ser aprovechados para crear una aplicación más compleja.

Para el desarrollo de la aplicación se consideraron las limitaciones que existen en cuanto a equipamiento en nuestra Universidad, motivo por el cual casi la totalidad de la aplicación es a nivel software. Este software corre en cualquier PC con procesador 286 o superior, con memoria base de 640 K y monitor VGA a color. En el programa se pueden capturar los patrones y probarlos, viendo la respuesta en pantalla así como también externamente por medio de la salida que provee el programa por el puerto paralelo de la PC. Para poder trabajar con esa señal de salida se implementó una pequeña y sencilla etapa de potencia, que fue necesaria para poder mover el móvil mecánico, que bien pudo ser cualquier otro objeto, lo cual permite su empleo en diversas aplicaciones, quizá para algunos sea necesaria la modificación de unas cuantas líneas de código en el programa fuente.

A pesar de las bondades y ventajas que presenta la simulación de Redes Neuronales en Software, se sigue teniendo la limitante de la velocidad de proceso y de entrenamiento. Por este motivo en muchos lados se emplean desde estaciones de trabajo hasta supercomputadoras para la simulación y entrenamiento, con el alto costo que implica la obtención de un equipo de esta índole. Actualmente se están desarrollando microprocesadores dedicados a esa tarea (neurochips), los cuales en algunos casos, ya vienen con algoritmos incorporados, pero tambien es cierto que el desarrollo de esta tecnología hace que el costo de los mismos neurochips sea elevado y no fácilmente costeable por varias empresas y mucho menos por instituciones educativas, así que el desarrollo en software sigue siendo el más viable hasta la fecha.

Hay que hacer notar las limitaciones que implica el uso del algoritmo empleado en el desarrollo de esta tesis, es decir, el algoritmo de Retropropagación del Error, limitaciones que pueden hacer que el algoritmo no sea tan universalmente aceptado. Uno de los problemas es el tiempo de entrenamiento, el cual es en algunos casos excesivamente largo y puede ser

el resultado de que la red a entrenar sea muy grande, hasta errores en la selección de los parámetros de entrenamiento, como lo puede ser el de un valor equivocado del learning rate y del número de iteraciones realizadas en el entrenamiento, lo que puede provocar una falla: caer en un mínimo local, pues la superficie de error de una red está llena de valles y de cimas. Por medio del gradiente descendente, la red puede quedar atrapada en un mínimo local con lo cual la red daría resultados erroneos.

Una posible solución este problema era aumentar el número de capas intermedias con el aumento de sus respectivas unidades, sin embargo no hay una regla que nos indique la cantidad exacta, siendo más que nada al tanteo la obtención de estas capas. Nosotros efectuamos varias pruebas variando tanto el número de capas intermedias, como el número de elementos procesadores de la(s) misma(s), sin notar un cambio significativo en el desempeño, pero si en el tiempo de entrenamiento.

Existen otros algoritmos que superan este problema, pero caen en otros, a pesar de esto, retropropagación sigue siendo el algoritmo por excelencia cuando se trata de problemas de clasificación de patrones.

Para la aplicación se logró minimizar el error realizando varios entrenamientos variando el learning rate, el cual indica el tamaño del paso en el camino hacia el mínimo total. Un paso muy grande provocó errores muy grandes, lo cual si se viera en la superficie de pesos se notaría como una distorsión muy grande. Un paso pequeño evita este problema aunque aumenta el tiempo de entrenamiento. Como no era el objetivo crear un entrenador de Redes Neuronales, este detalle no importó mucho, y se solucionó entrenando la red en una computadora con procesador Pentium a 100 MHz, con lo cual el tiempo de entrenamiento disminuyó considerablemente.

A continuación mostramos los parámetros del entrenamiento final:

- 35 elementos procesadores de la capa de entrada.
- 12 elementos procesadores de la capa intermedia.
- 4 elementos procesadores de la capa de salida.
- Learning Rate 0.2 (valor óptimo por defecto del DynaMind).
- 20,000 iteraciones a realizar durante el entrenamiento.

- Fluctuación de los pesos = 5.
- Fluctuación del ruido a la entrada = 0.1 (10% de la entrada).
- Valores límites de la sigmoide = -1, 1
- Valor del error alcanzado después de 20,000 iteraciones = 1.69×10^{-6}
- Tiempo total de entrenamiento = 28.07 minutos.

En cuanto al desempeño de la red, a pesar de que no es óptimo, se puede considerar aceptable con un porcentaje de aciertos mayor al 70%.

Se considera que el objetivo planteado en esta tesis fue cubierto plenamente y que el trabajo realizado puede servir de base para iniciar una educación en el área, con lo cual podrían vencerse los obstáculos y resistencias intelectuales que mucha gente presenta, pues hay que recordar que antes de correr hay que aprender a caminar.

Agustín de Jesús Astorga de Riquer
y
José Luis Cano García

Apéndices

Apéndice A

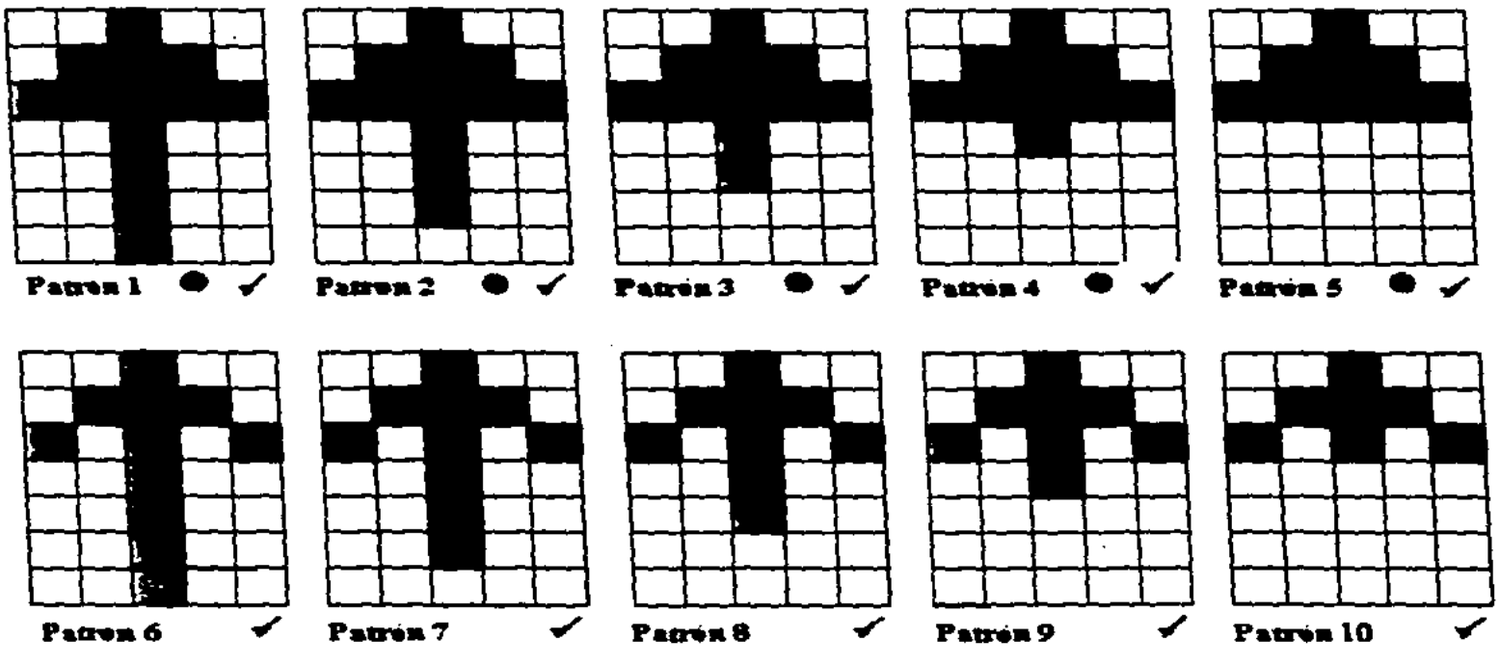
Apendice A

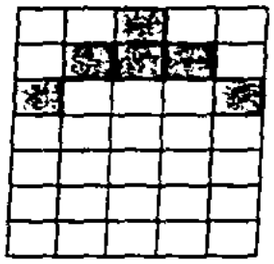
En las siguiente páginas se muestra el conjunto de patrones que sirvieron para efectuar los diversos entrenamientos, en la parte inferior de los mismos se indica cual fue aceptado y cual rechazado.

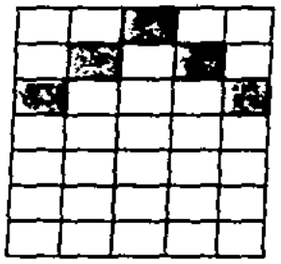
✓ Patrón reconocido

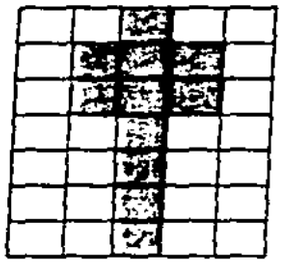
✖ Patrón rechazado

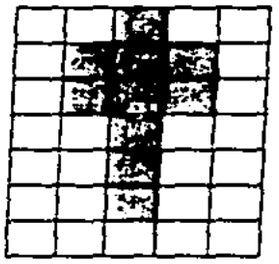
● Patrón empleado para realizar el entrenamiento

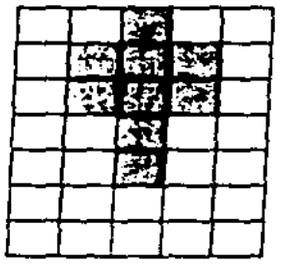


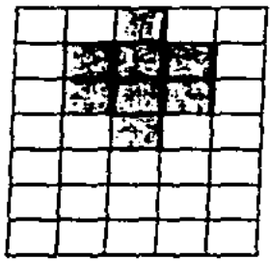

Patron 11 ✓

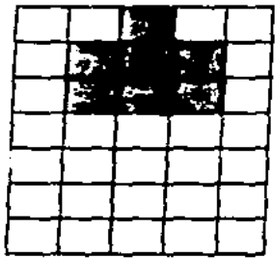

Patron 12 ✓

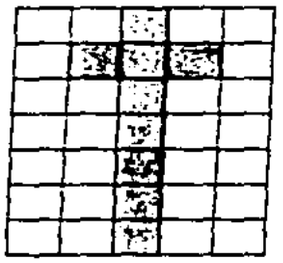

Patron 13 ✓

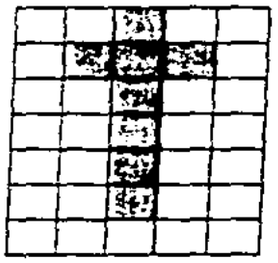

Patron 14 ✓

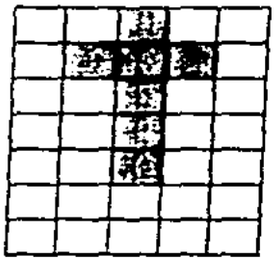

Patron 15 ✓

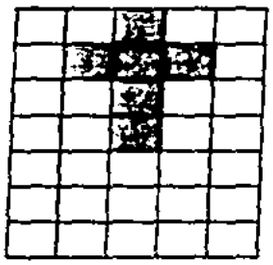

Patron 16 ✓

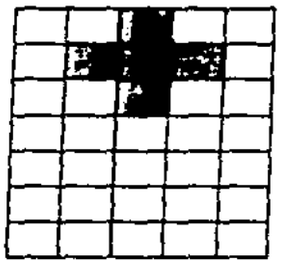

Patron 17 ✓

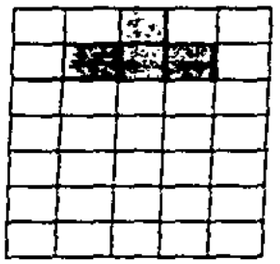

Patron 18 ✓

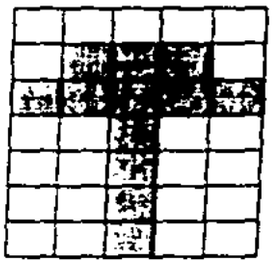

Patron 19 ✓

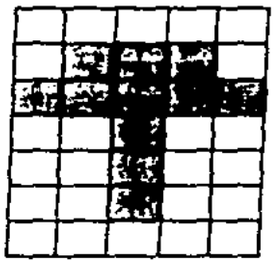

Patron 20 ✓

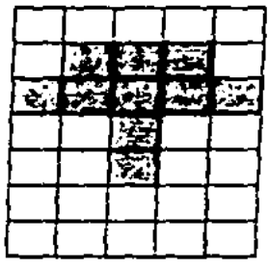

Patron 21 ✓

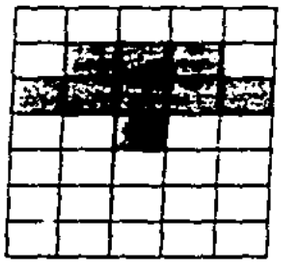

Patron 22 ✓

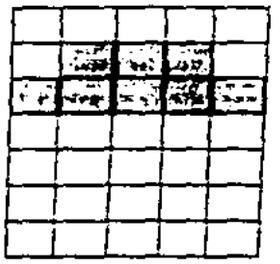

Patron 23 ✓

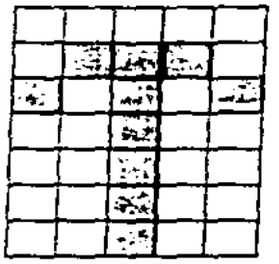

Patron 24 ✓

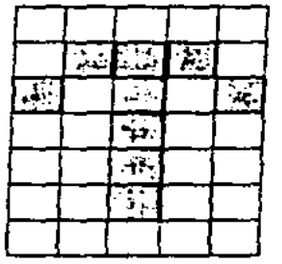

Patron 25 ✓

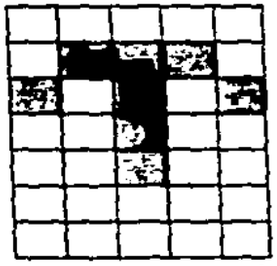

Patron 26 ✓

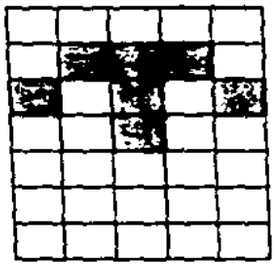

Patron 27 ✓

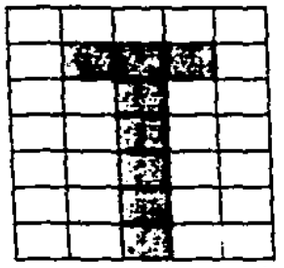

Patron 28 ✓

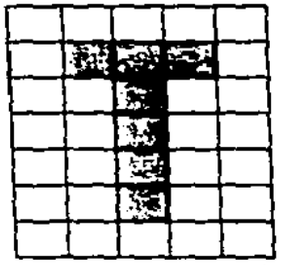

Patron 29 ✓

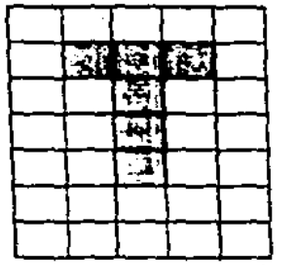

Patron 30 ✓

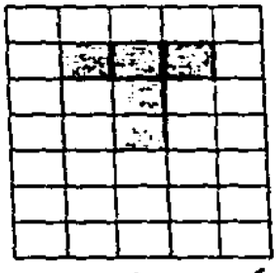

Patron 31 ✓

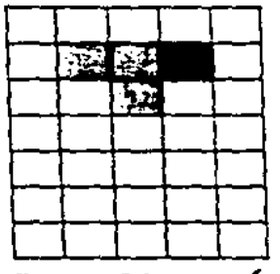

Patron 32 ✓

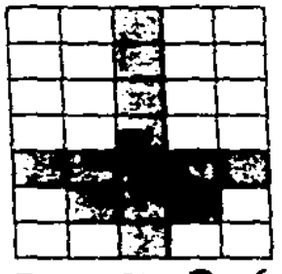

Patron 33 ✓

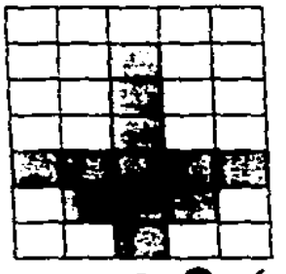

Patron 34 ✓

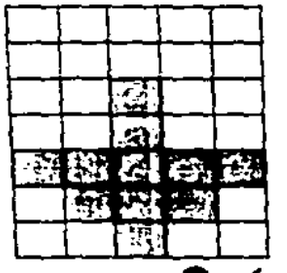

Patron 35 ✓

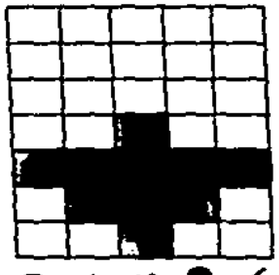

Patron 36 ✓

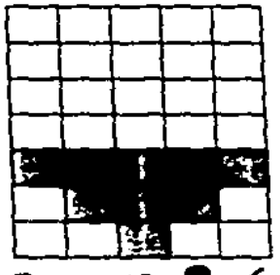

Patron 37 ✓

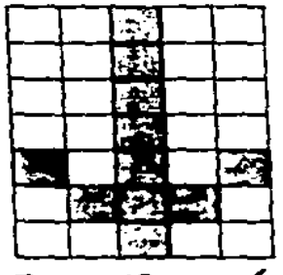

Patron 38 ● ✓

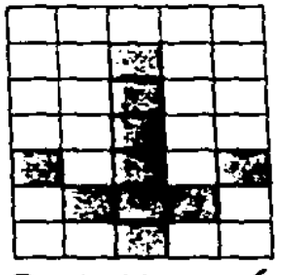

Patron 39 ● ✓

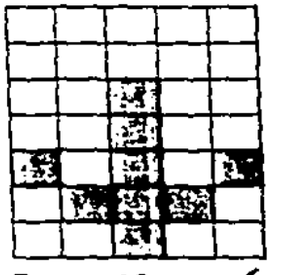

Patron 40 ● ✓

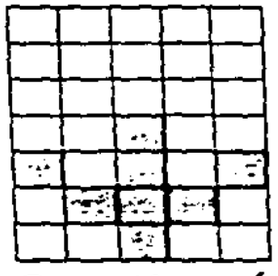

Patrón 41 ● ✓

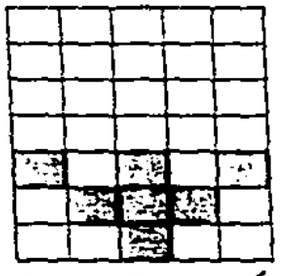

Patron 42 ● ✓

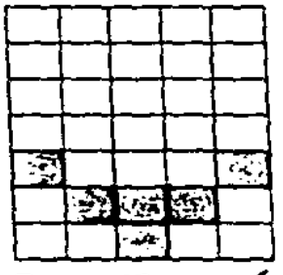

Patron 43 ✓

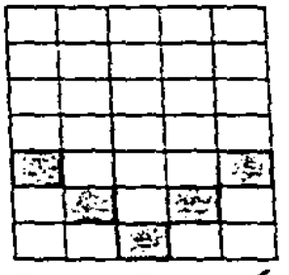

Patrón 44 ✓

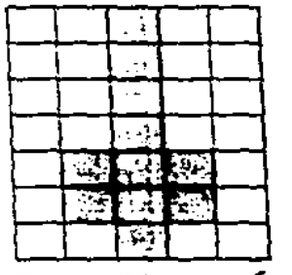

Patron 45 ✓

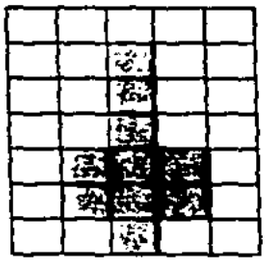

Patron 46 ✓

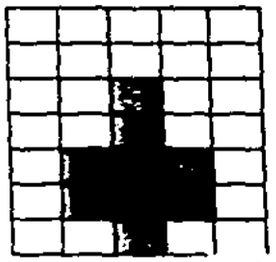

Patron 47 ✓

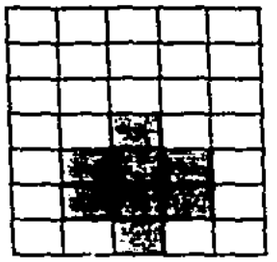

Patron 48 ✓

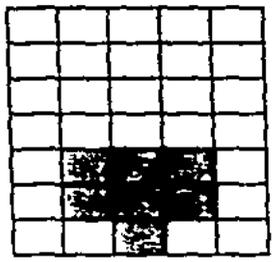

Patron 49 ✓

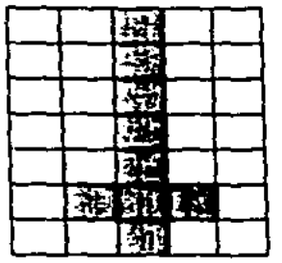

Patron 50 ✓

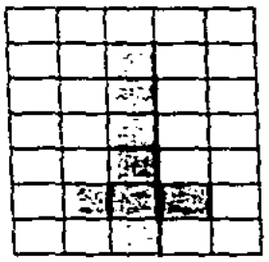

Patron 51 ✓

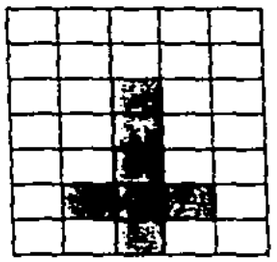

Patron 52 ✓

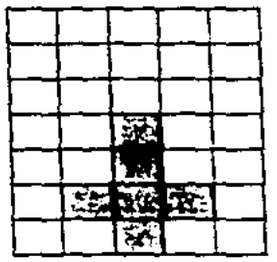

Patron 53 ✓

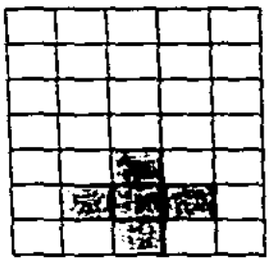

Patron 54 ✓

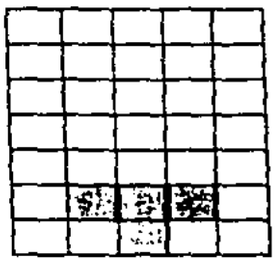

Patron 55 ✓

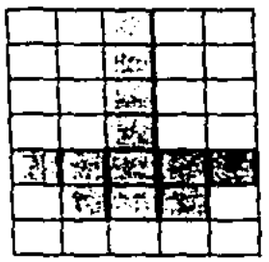

Patron 56 ✓

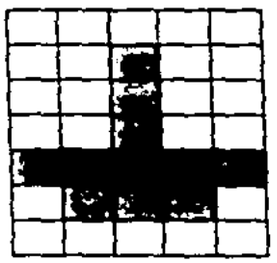

Patron 57 ✓

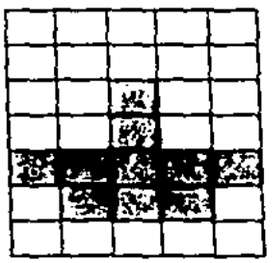

Patron 58 ✓

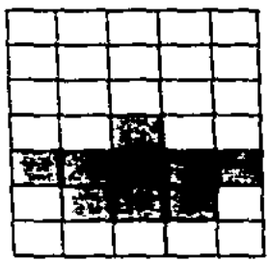

Patron 59 ✓

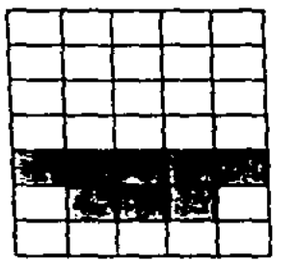

Patron 60

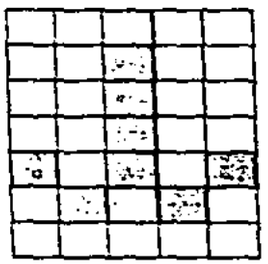

Patron 61 ✓

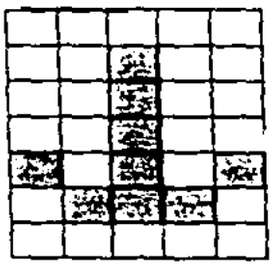

Patron 62 ✓

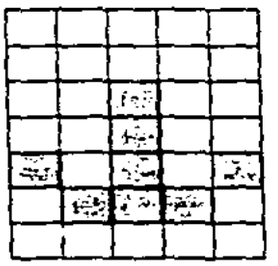

Patron 63 ✓

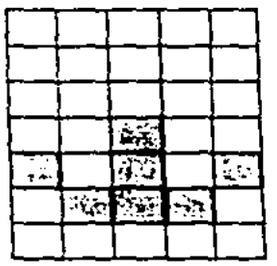

Patron 64 ✓

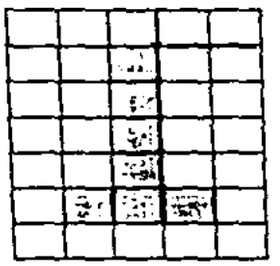

Patron 65 ✓

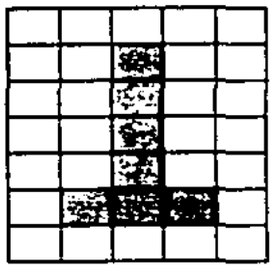

Patron 66 ✓

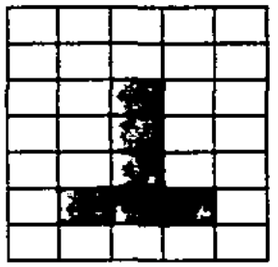

Patron 67 ✓

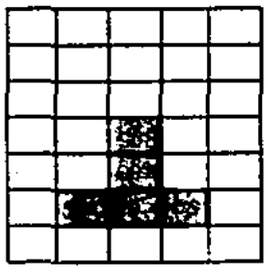

Patron 68 ✓

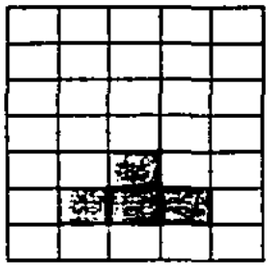

Patron 69 ✓

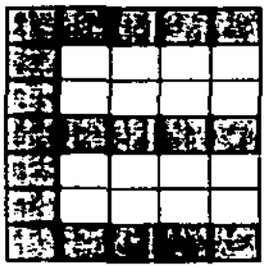

Patron 70 ✓

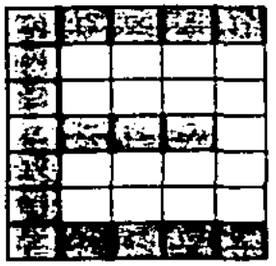

Patrón 71 ✓


Patrón 72 ✓

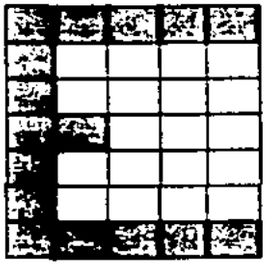

Patrón 73 ✗

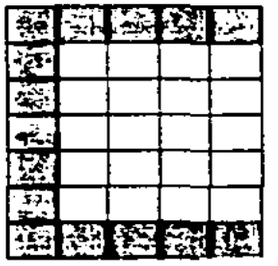

Patrón 74 ✗


Patrón 75 ● ✓

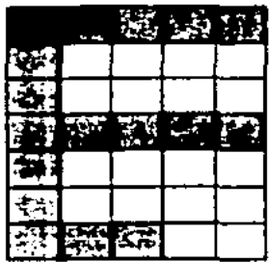

Patrón 76 ✗

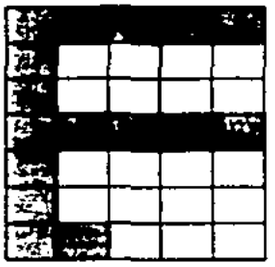

Patrón 77 ✗

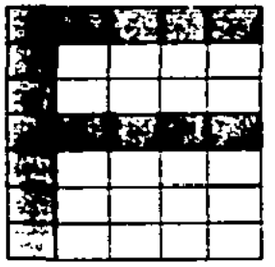

Patrón 78 ✗

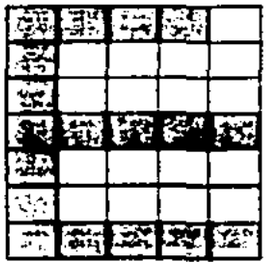

Patrón 79 ✗

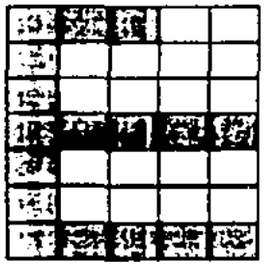

Patrón 80 ✓

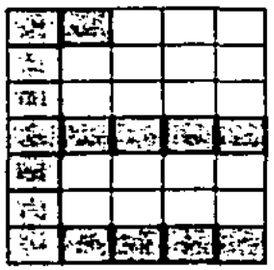

Patrón 81 ✓

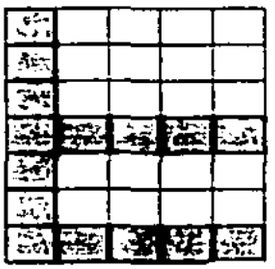

Patrón 82 ✓


Patrón 83 ✓

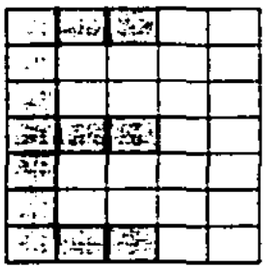

Patrón 84 ✓

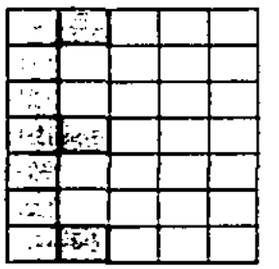

Patrón 85 ✓

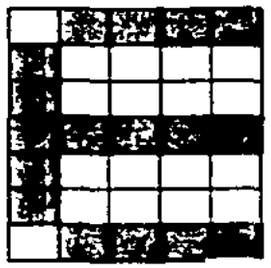

Patrón 86 ✓

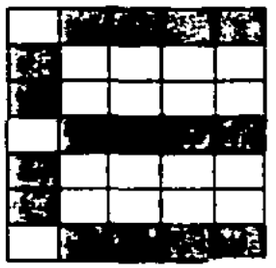

Patrón 87 ✗

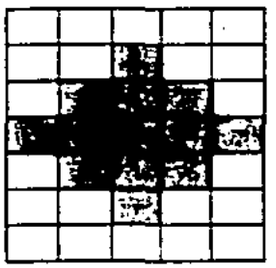

Patrón 88 ● ✓

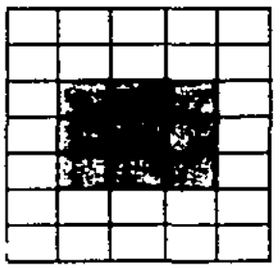

Patrón 89 ● ✓

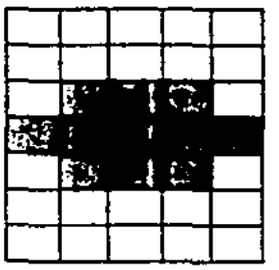

Patrón 90 ● ✓

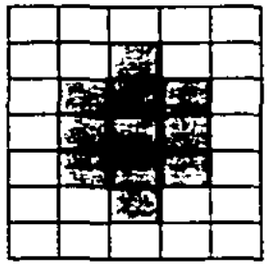

Patrón 91 ✱

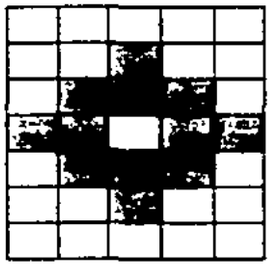

Patrón 92 ✱

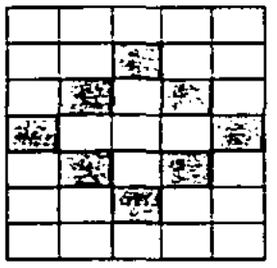

Patrón 93 ● ✓

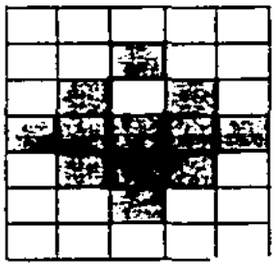

Patrón 94 ● ✓

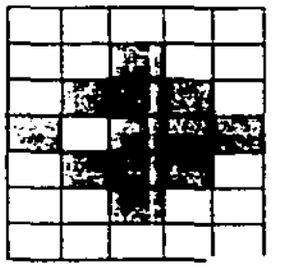

Patrón 95 ✓

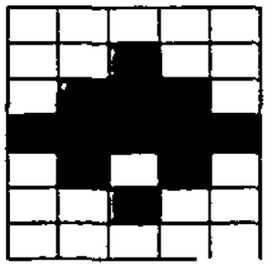

Patrón 96 ✓

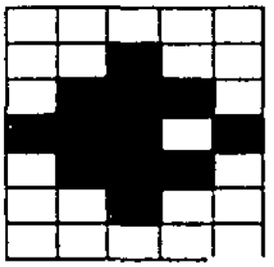

Patrón 97 ✓

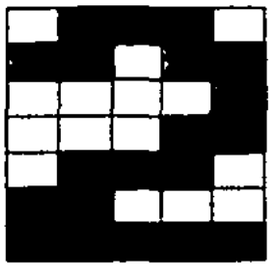

Patrón 98 ✓

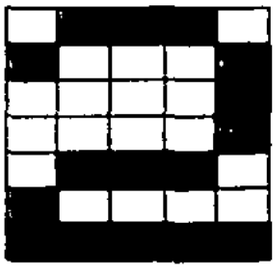

Patrón 99 ✱

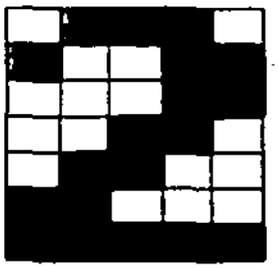

Patrón 100 ✱

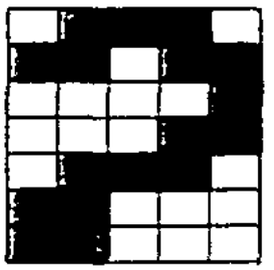

Patrón 101 ✓

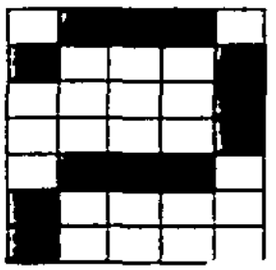

Patrón 102 ✱

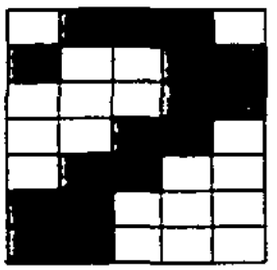

Patrón 103 ● ✓

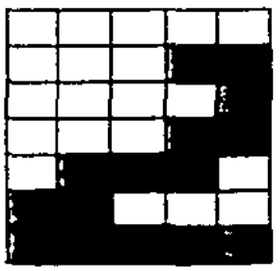

Patrón 104 ● ✓

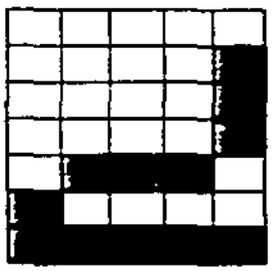

Patrón 105 ● ✓

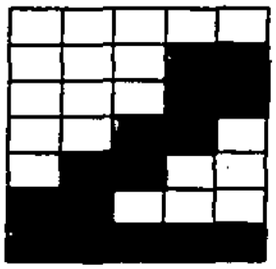

Patrón 106 ✓

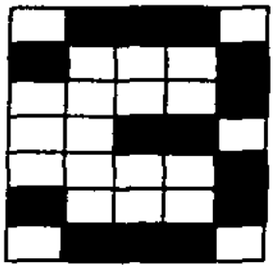

Patrón 107 ✱

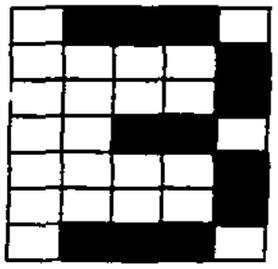

Patrón 108 ✓

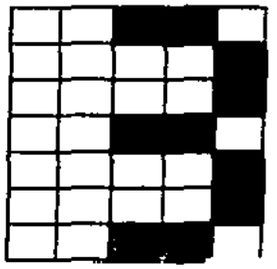

Patrón 109 ✓

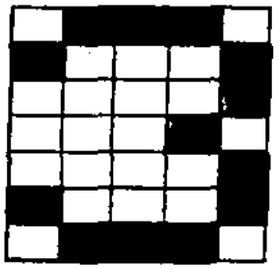

Patrón 110 ✓

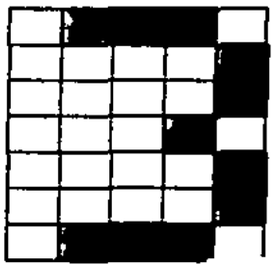

Patrón 111 ✓

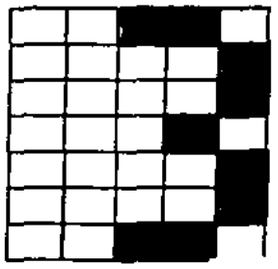

Patron 112 ● ✓

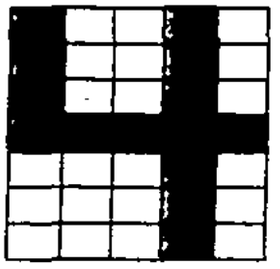

Patron 113 ● ✓

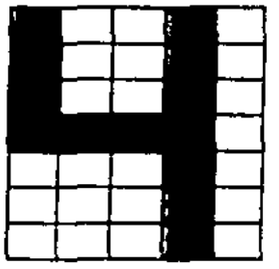

Patrón 114 ≈

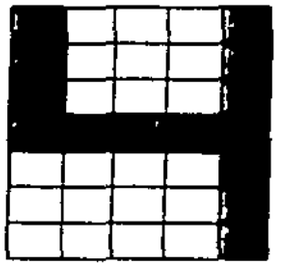

Patron 115 ✓

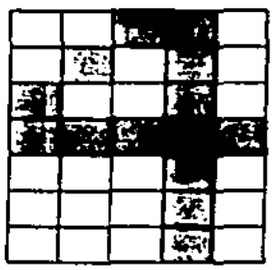

Patrón 116 ≈

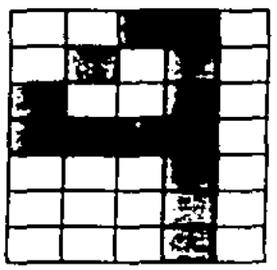

Patron 117 ≈

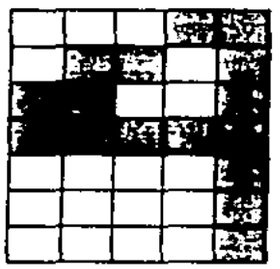

Patron 118 ✓

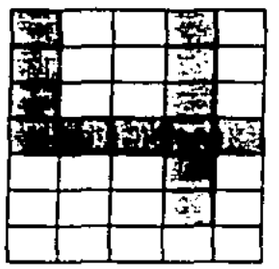

Patron 119 ✓

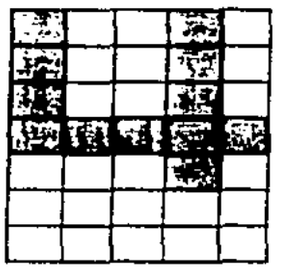

Patron 120 ● ✓

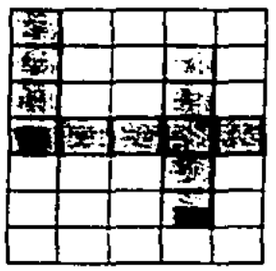

Patrón 121 ✓

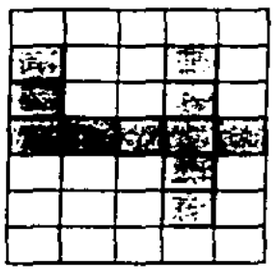

Patron 122 ✓

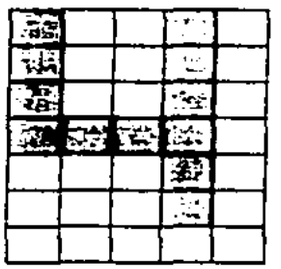

Patrón 123 ● ✓

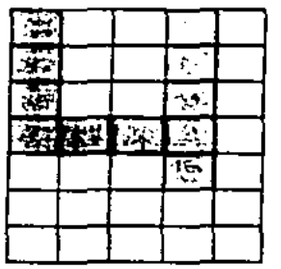

Patron 124 ✓

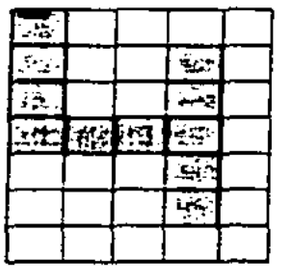

Patron 125 ✓

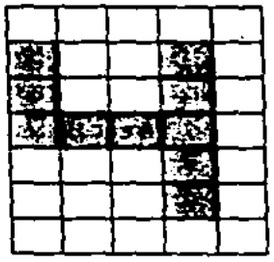

Patron 126 ✓

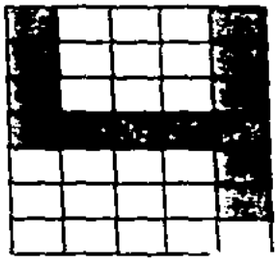

Patrón 127 ✓

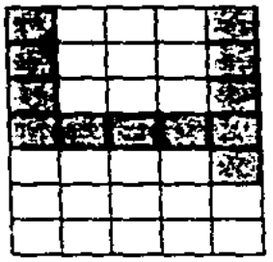

Patron 128 ● ✓

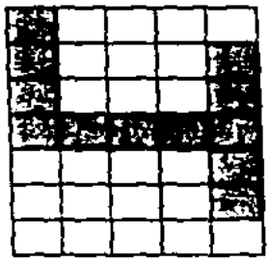

Patron 129 ✓

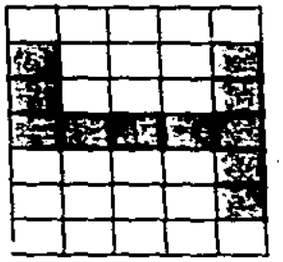

Patron 130 ✓

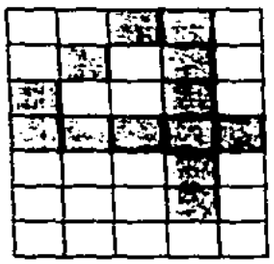

Patron 131 ✓

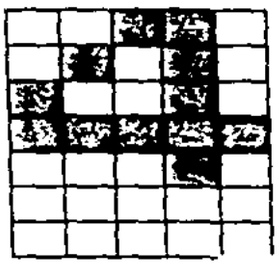

Patron 132 ✓

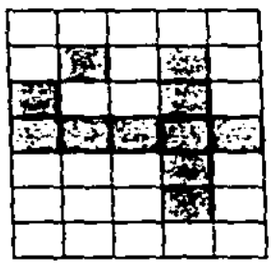

Patron 133 ✓

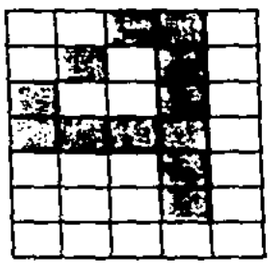

Patron 134 ✓

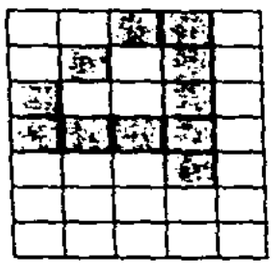

Patron 135 ✓

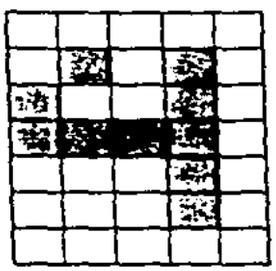

Patron 136 ✓

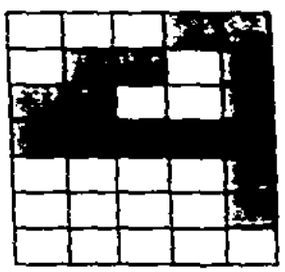

Patron 137 ≠

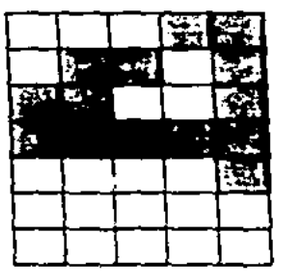

Patron 138 ✓

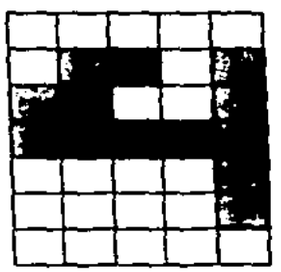

Patron 139 ✓

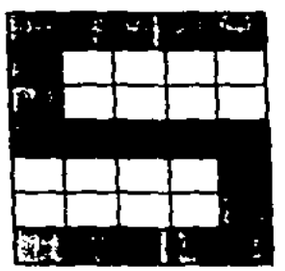

Patron 140 ✓

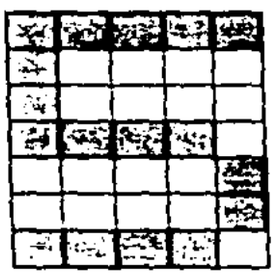

Patron 141 ✓

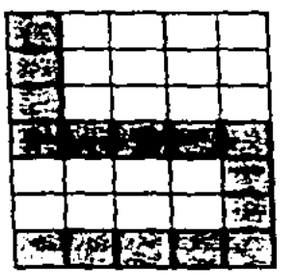

Patron 142 ✓

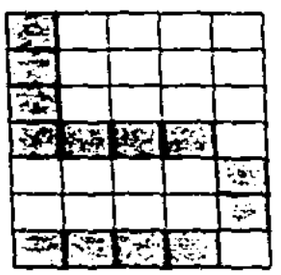

Patron 143 ✓

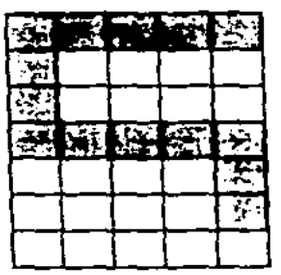

Patron 144 ✓

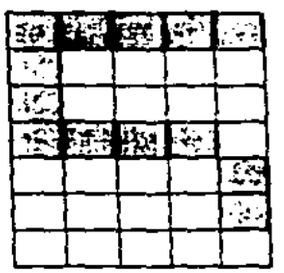

Patron 145 ● ✓

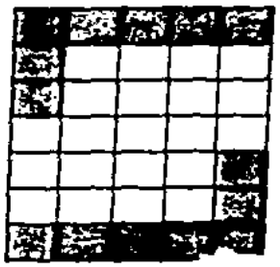

Patron 146 ● ✓

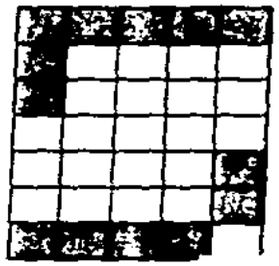

Patron 147 ≠

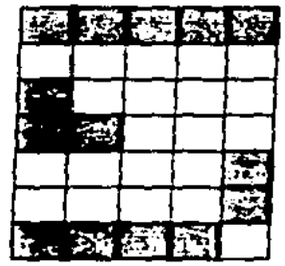

Patron 148 ✓

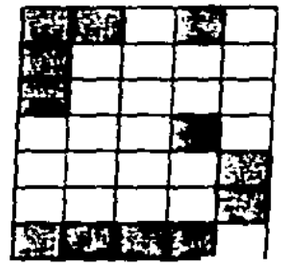

Patron 149 ✓

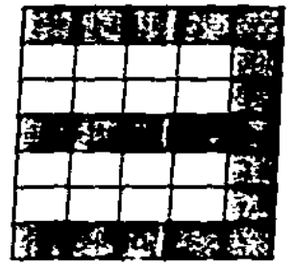

Patron 150 ✓

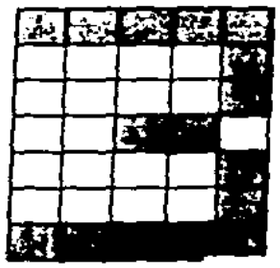

Patrón 151 ✱

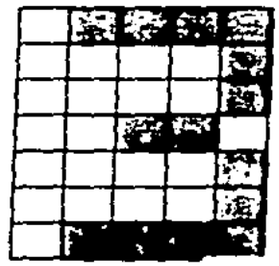

Patrón 152 ✱

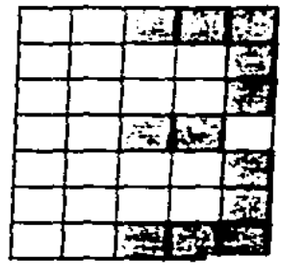

Patrón 153 ✓

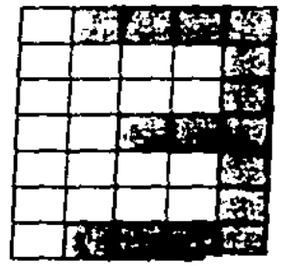

Patrón 154 ✱

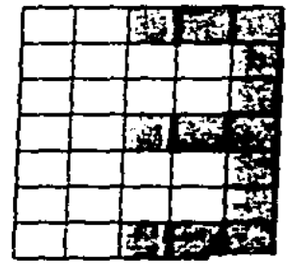

Patrón 155 ● ✓

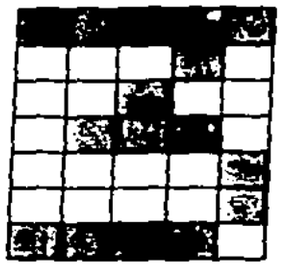

Patrón 156 ✱

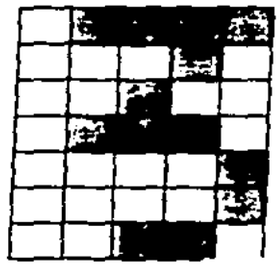

Patrón 157 ✱

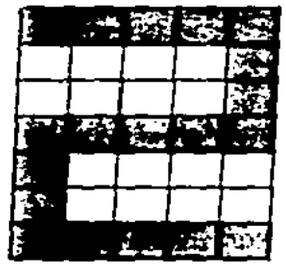

Patrón 158 ✱

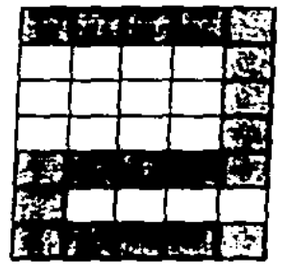

Patrón 159 ● ✱

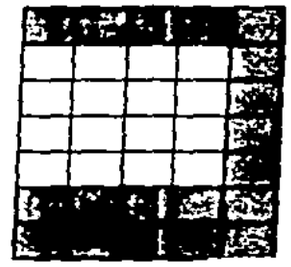

Patrón 160 ✱

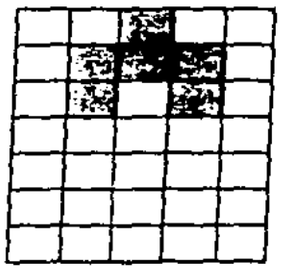

Patrón 161 ● ✓

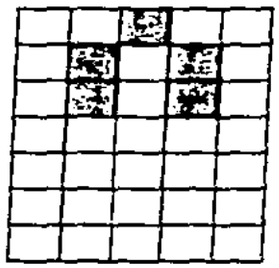

Patrón 162 ✱

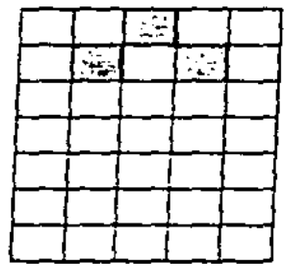

Patrón 163 ● ✓

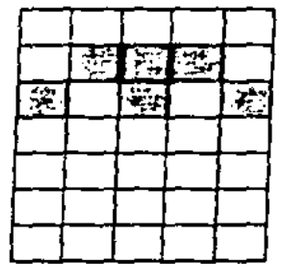

Patrón 164 ✓

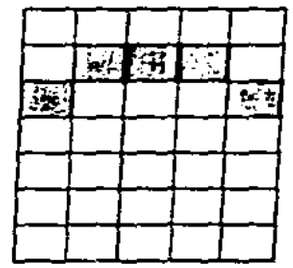

Patrón 165 ✱

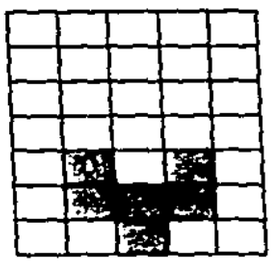

Patrón 166 ✓

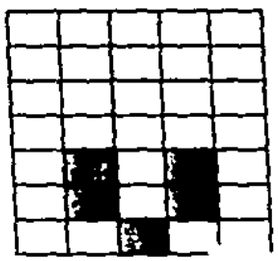

Patrón 167 ✓

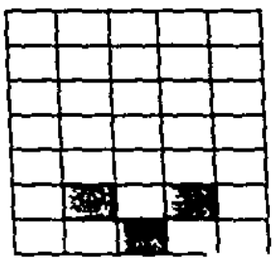

Patrón 168 ✓

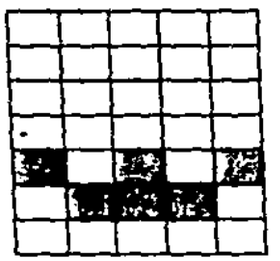

Patrón 169 ✓

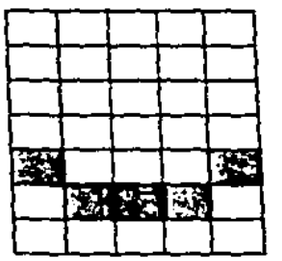

Patrón 170 ✓

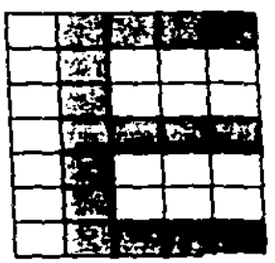

Patron 171 ✓

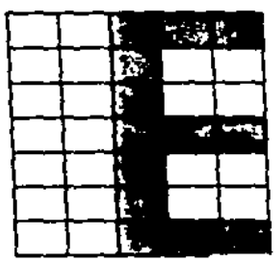

Patron 172 ✗

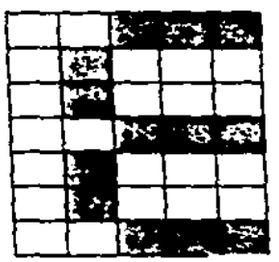

Patron 173 ✗

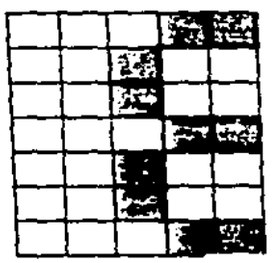

Patron 174 ✓

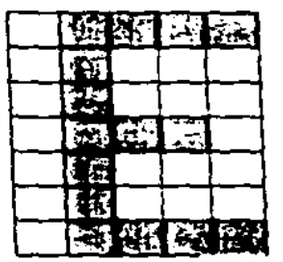

Patron 175 ✓

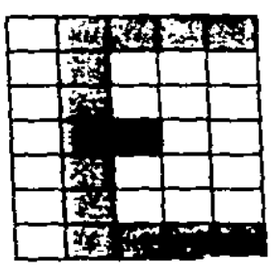

Patron 176 ✗

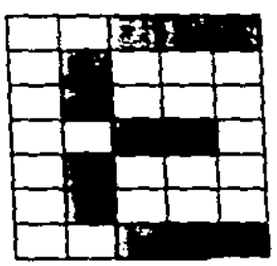

Patron 177 ✗

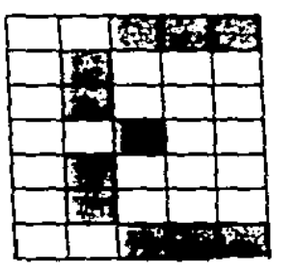

Patron 178 ✗

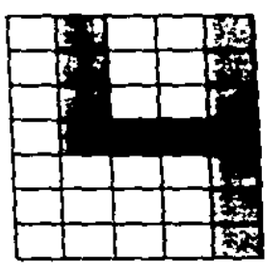

Patron 179 ✗

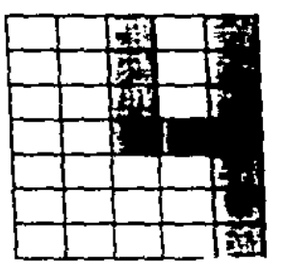

Patron 180 ✗

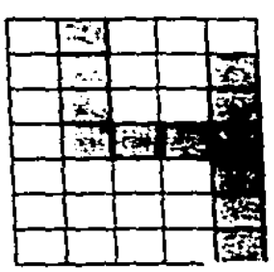

Patron 181 ✗

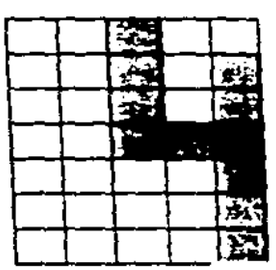

Patron 182 ✗

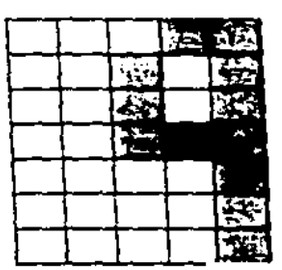

Patron 183 ✗

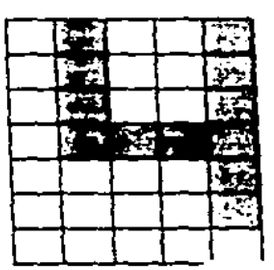

Patron 184 ✓

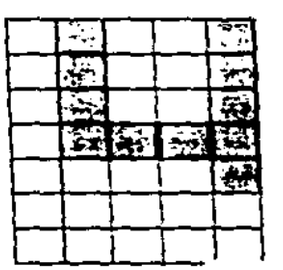

Patron 185 ✗

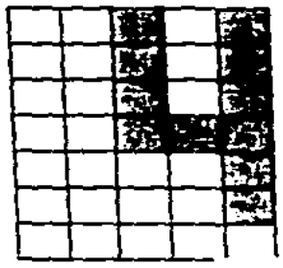

Patron 186 ✗

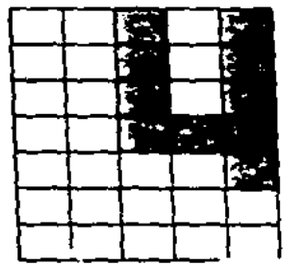

Patron 187 ✗

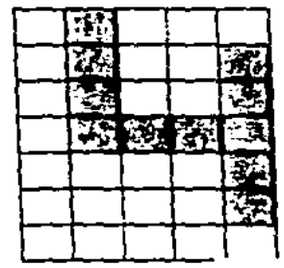

Patron 188 ✗

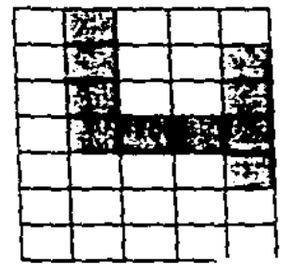

Patron 189 ✗

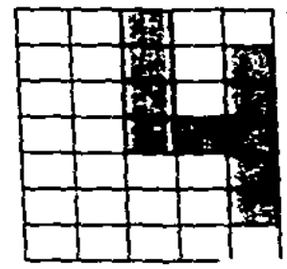

Patron 190 ✗

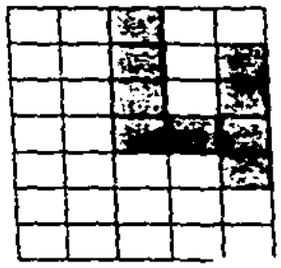

Patron 191 ✱

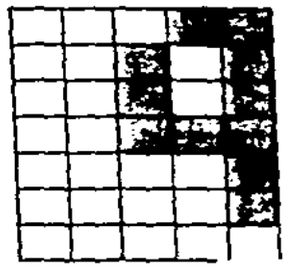

Patron 192 ✱

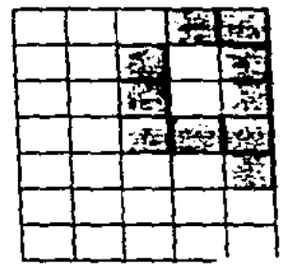

Patron 193 ✱

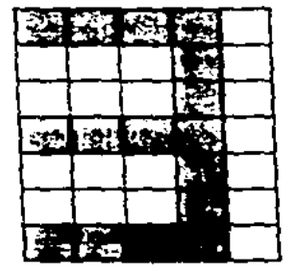

Patron 194 ✱

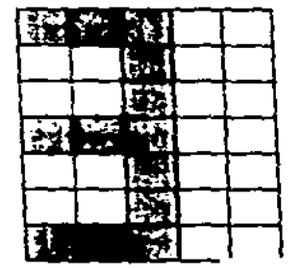

Patron 195 ✱

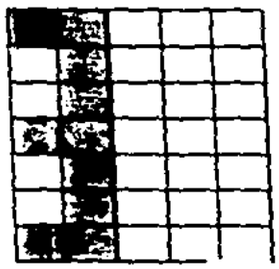

Patron 196 ✱

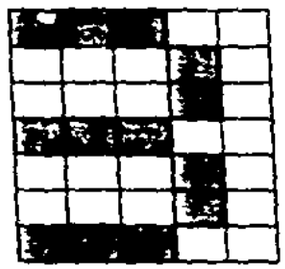

Patron 197 ✱

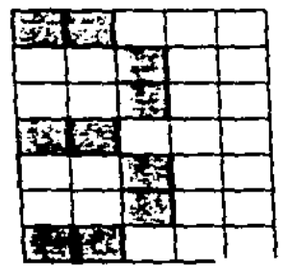

Patron 198 ✱

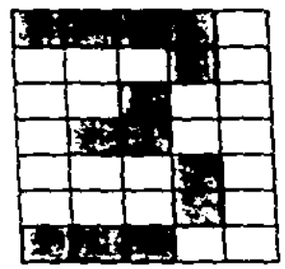

Patron 199 ● ✓

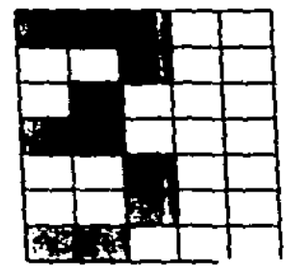

Patron 200 ✱

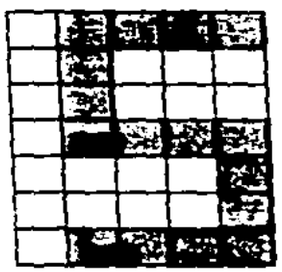

Patron 201 ✱

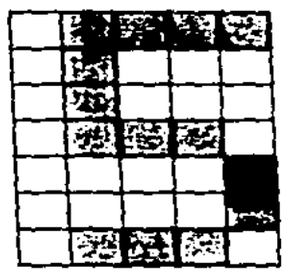

Patron 202 ✓

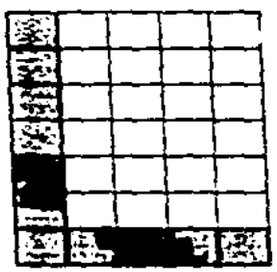

Patron 203 ✱

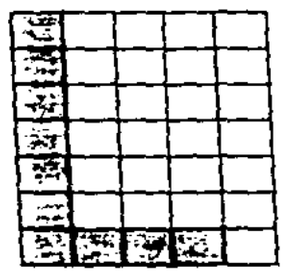

Patron 204 ✓

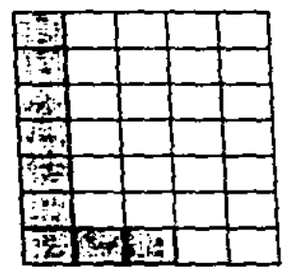

Patron 205 ✱

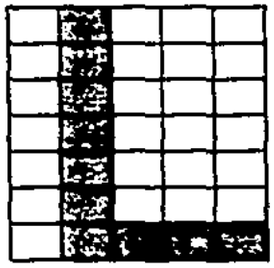

Patron 206 ● ✓


Patron 207 ● ✓


Patron 208 ● ✓


Patron 209 ● ✓


Patron 210 ● ✓



Patrón 211 ● ✓

Apéndice B

A continuación mostramos el fuente del programa desarrollado.

Cabeceras del Programa

Cabecera de la rutina de control del mouse (MOUSE.H)

```
/*
      Macros para el uso del raten.
      Archivo de Biblioteca.
*/

# define lo(f)  ((f) & 0Xff)
# define hi(f)  (lo(f >> 8))

// Definicion de los botones del raton

# define LEFT_MOUSE_PRESS       0xff01
# define RIGHT_MOUSE_PRESS      0xff02
# define LEFT_MOUSE_REL         0xff11
# define RIGHT_MOUSE_REL        0xff12

/* CONTROLADORES DEL RATON */

# define M_RESET        0
# define M_SHOW_CURS    1
# define M_HIDE_CURS    2
# define M_GET_STATUS   3
# define M_SET_CURS     4
# define M_GET_PRESS    5
# define M_GET_REL      6
# define M_SET_X_BOUNDS  7
# define M_SET_Y_BOUNDS  8
# define M_SET_G_CURS    9
# define M_SET_T_CURS    10

// DEFINICION DE OTRAS CONSTANTES

# define MOUSE_NEEDED    1
# define MOUSE_OPTIONAL  0
# define MOUSE_TEXT_MODE  0

// VARIABLES EXTERNAS PARA EL USO DEL RATON

extern int mouse_text_x;
extern int mouse_text_y;
extern int mouse_grph_x;
extern int mouse_grph_y;
extern int mouse_initialized;

// PROTOTIPO DE FUNCIONES PARA EL USO DEL RATON
```

```
extern void  mouse(int *m1,int *m2,int *m3,int *m4);
extern int   check_mouse_driver(int need_mouse);
extern int   init_mouse(int need_mouse,int gd,int gm);
extern int   mouse_reset(void);
extern int   move_mouse(int x,int y);
extern void  mouse_on(int code);
extern void  mouse_off(int code);
extern void  mouse_grph_posn(int *x,int *y);
extern void  mouse_text_posn(int *x,int *y);
extern int   mouse_in_box(int xt,int yt,int xb,int yb);
extern int   button_release(int b);
extern int   button_press(int b);
extern int   button_state(void);
extern int   mouse_trigger(int button_dir);
```

Cabecera de la rutina de diseño de las ventanas de trabajo (DESIGN.H)

```
extern void V_trabajo(int xt,int yt,int xb,int yb,int color,int fill);
extern void boton(int xt,int yt,int xb,int yb,int borde);
extern void A_trabajo(int xt,int yt,int xb,int yb,int color,int fill,int borde);
extern void _textos(char *str,int x,int y,int color);
extern void _texto_s(char *str,int x,int y,int color);
```

Cabecera de la rutina del programa principal (SPIDER.H)

```
# define ENT_BASE          5
# define ENT_ALTURA        7
# define SIZE_CAPA_INT     12
# define SAL_BASE          4
# define SAL_ALTURA        1
# define SIZE_PESOS  484
# define SIZE_ENTRADA   ENT_BASE*ENT_ALTURA
# define SIZE_SALIDA       SAL_BASE*SAL_ALTURA
# define TRUE          1
# define FALSE         0
# define CLIP_ON       1
# define MOV           20
# define CUADROS           15
# define MAX_FILES   100
# define LIM_INF           -1
# define MITAD           0.5
# define LIM_SUP    1

# define PGUP    0x4900
# define PGDN    0x5100
# define HOME    0x4700
# define UARROW  0x4800
# define DARROW  0x5000
# define LARROW  0x4b00
# define RARROW  0x4d00
# define BKSPACE 0x0e08
```

```
# define ENTER   0x1c0d
# define ESC     0x011b

# define F1      0x3b00

# define LETA    0x1e41
# define LETa    0x1e61
# define LETB    0x3042
# define LETb    0x3062
# define LETC    0x2e43
# define LETc    0x2e63
# define LETD    0x2044
# define LETd    0x2064
# define LETE    0x1245
# define LETe    0x1265
# define LETG    0x2247
# define LETg    0x2267
# define LETI    0x1749
# define LETi    0x1769
# define LETM    0x324d
# define LETm    0x326d
# define LETN    0x314e
# define LETn    0x316e
# define LETO    0x184f
# define LETo    0x186f
# define LETP    0x1950
# define LETp    0x1970
# define LETR    0x1f52
# define LETr    0x1f72
# define LETS    0x1f53
# define LETs    0x1f73
# define LETY    0x1559
# define LETy    0x1579
```

Rutina de control del mouse (mouse.cpp)[1]

```
*
     Estas cabeceras seran obligatorias en el caso de que
     su programa principal no las contenga.
*
*
# include <bios.h>
# include <conio.h>
# include <dos.h>
# include <process.h>
# include <stdio.h>
*
# include "mouse.h"
```

[1] Programa desarrollado por Pedro Lopez Urzua. Facultad de Ingenieria. Departamento de Ingenieria de Control. Universidad Nacional Autonoma de Mexico.

```
// Variables globales de uso externo e interno
int mouse_text_x,mouse_text_y;
int mouse_grph_x,mouse_grph_y;
int mouse_initialized = 0;
int mouse_mode=0;

static int prev_cursor_state = 0;
static char far *bios_video_area = (char far *) 0x00400049L;

// Funciones externas e internas
void mouse(int *m1,int *m2,int *m3,int *m4);
int  check_mouse_driver(int need_mouse);
int  init_mouse(int need_mouse,int gd,int gm);
int  mouse_reset();
int  move_mouse(int x,int y);
void mouse_on(int code);
void mouse_off(int code);
void mouse_grph_posn(int *x,int *y);
void mouse_text_posn(int *x,int *y);
int  mouse_in_box(int xt,int yt,int xb,int yb);
int  button_release(int b);
int  button_press(int b);
int  button_state();
int  mouse_trigger(int button_dir);

// Funciones internas
static void set_mouse_posn(int *x,int *y);
static int  low_res_mode(int gd,int gm);

void mouse(int *m1,int *m2,int *m3,int *m4)
{
     union REGS inregs,outregs;

     inregs.x.ax = *m1;
     inregs.x.bx = *m2;
     inregs.x.cx = *m3;
     inregs.x.dx = *m4;
     int86(0x33,&inregs,&outregs);
     *m1 = outregs.x.ax;
     *m2 = outregs.x.bx;
     *m3 = outregs.x.cx;
     *m4 = outregs.x.dx;
}

int check_mouse_driver(int need_mouse)
{
     void far *address;

     address = getvect(0x33);
     if(address==NULL || (*(unsigned char *) address==0xcf))
     {
          if(need_mouse)
               return 1;
          else
```

```
                return 0;
        }
        return 1;
}

int init_mouse(int need_mouse,int gd,int gm)
{
        int m1;

        mouse_initialized = 0;
        if(check_mouse_driver(need_mouse))
        {
                if(gd==7)
                        *bios_video_area = 6;
                mouse_mode=low_res_mode(gd,gm);
                m1 = mouse_reset();
                if(m1)
                {
                        mouse_initialized = 1;
                        move_mouse(0,0);
                        mouse_on(0);
                }
                else
                        return mouse_initialized;
        }
        return mouse_initialized;
}

static int low_res_mode(int gd,int gm)
{
        if((gd==2 || gd==3 || gd==9) && (gm>=0 && gm<=3))
                return 1;
        else
                return 0;
}

int mouse_reset()
{
        int m1,m2,m3,m4;

        mouse_off(1);
        m1=M_RESET;
        mouse(&m1,&m2,&m3,&m4);
        set_mouse_posn(&m3,&m4);
        return m1;
}

int move_mouse(int x,int y)
{
        int m1,m2,m3,m4;

        if(!mouse_initialized) return 0;
        m1 = M_SET_CURS;
        if(!mouse_mode)
```

```
    {
        m3 = x*8;
        m4 = y*8;
    }
    else
    {
        m3=x;
        m4=y;
    }
    mouse(&m1,&m2,&m3,&m4);
    set_mouse_posn(&m3,&m4);
    return 0;
}

void mouse_on(int restoreflag)
{
    int m1,m2,m3,m4;

    if(mouse_initialized)
    {
        if(!restoreflag || prev_cursor_state)
        {
            m1 = M_SHOW_CURS;
            mouse(&m1,&m2,&m3,&m4);
            prev_cursor_state = 1;
        }
    }
}

void mouse_off(int tempflag)
{
    int m1,m2,m3,m4;

    if(mouse_initialized)
        if(prev_cursor_state)
        {
            m1 = M_HIDE_CURS ;
            mouse(&m1,&m2,&m3,&m4);
            if(!tempflag)
                prev_cursor_state = 0;
        }
}

void mouse_grph_posn(int *x,int *y)
{
    int m1,m2;

    if(mouse_initialized)
    {
        m1 = M_GET_STATUS;
        mouse(&m1,&m2,x,y);
        set_mouse_posn(x,y);
    }
    else
```

```
        *x = *y = 0;
}

void mouse_text_posn(int *x,int *y)
{
     mouse_grph_posn(x,y);
     *x=mouse_text_x;
     *y=mouse_text_y;
}

int mouse_in_box(int xt,int yt,int xb,int yb)
{
     int x,y;

     if(mouse_initialized)
     {
          if(mouse_mode)
          {
               x = mouse_grph_x;
               y = mouse_grph_y;
          }
          else
          {
               x = mouse_text_x;
               y = mouse_text_y;
          }
          if( ((yt<=y) && (y<=yb)) && ((xt<=x) && (x<=xb)) )
               return 1;
     }
     return 0;
}

int button_release(int b)
{
     int m1,m2,m3,m4;

     if(mouse_initialized)
     {
          m1 = M_GET_REL;
          m2 = b;
          mouse(&m1,&m2,&m3,&m4);
          set_mouse_posn(&m3,&m4);
          if(m2)
               return 1;
     }
     return 0;
}

int button_press(int b)
{
     int m1,m2,m3,m4;

     if(mouse_initialized)
     {
```

```
        m1 = M_GET_PRESS ;
        m2 = b;
        mouse(&m1,&m2,&m3,&m4);
        set_mouse_posn(&m3,&m4);
        if(m2)
            return 1;
    }
    return 0;
}

int button_state()
{
    int m1,m2,m3,m4;

    if(mouse_initialized)
    {
        m1 = M_GET_STATUS;
        mouse(&m1,&m2,&m3,&m4);
        set_mouse_posn(&m3,&m4);
        return m2;
    }
    return 0;
}

static void set_mouse_posn(int *x,int *y)
{
    mouse_grph_x = *x;
    mouse_grph_y = *y;
    mouse_text_x = *x/8 + 1;
    mouse_text_y = *y/8 + 1;
}

int mouse_trigger(int button_dir)
{
    if(bioskey(1))
        return bioskey(0);
    else
    {
        if(button_dir)
        {
            if(button_press(0))
                return LEFT_MOUSE_PRESS;
            if(button_press(1))
                return RIGHT_MOUSE_PRESS;
        }
        else
        {
            if(button_release(0))
                return LEFT_MOUSE_REL;
            if(button_release(1))
                return RIGHT_MOUSE_REL;
        }
    }
    return 0;
```

```
}
```

Rutina para el control del video (VIDEO.CPP)

```
/*
     Funcien que inicializa el modo gr fico para el programa
*/
void inicializa_graficas(void);

/*
     Funcien que finaliza el modo gr fico para el programa
*/
void finaliza_graficas(void);

/*
     Funcien que indica si alguno de los archivos de tipos de letras en uso
     por la aplicacien existe o no.
*/
int fuente(int letra,int tam,int stdo);

void inicializa_graficas(void)
{
     int gdriver = DETECT, gmode, errorcode;
     initgraph(&gdriver, &gmode, "");
     errorcode = graphresult();
     if (errorcode != grOk)
     {
          cout<<"Error de Graficas: "<<grapherrormsg(errorcode)<<"\n";
          exit(1);
     }
     clearviewport();
     init_mouse(MOUSE_NEEDED,gdriver,gmode);
}

void finaliza_graficas(void)
{
     mouse_off(1);
     mouse_reset();
     closegraph();
}

int fuente(int letra,int tam,int stdo)
{
     int userfont=0;
     int errorcode;
     char *string[10]={{"EURO.CHR"},{"GOTH.CHR"},{"LCOM.CHR"},{"LITT.CHR"},
                       {"BOLD.CHR"},{"SANS.CHR"},{"SCRI.CHR"},{"SIMP.CHR"},
                       {"TRIP.CHR"},{"TSCR.CHR"}};

     if(letra-9 || letra<0)
     {
          cout <<"Archivo de fuentes inexistente."<<endl;
```

```
        return 0;
    }
    if(searchpath(string[letra])==NULL)
    {
        cout<<"Archivo "<<string[letra]<<" no esta presente, verifiquelo."<<endl;
        return 0;
    }
    userfont = installuserfont(string[letra]);
    settextstyle(userfont,stdo,tam);
    return 1;
}
```

Rutina de diseño de las ventanas de trabajo (DESIGN.CPP)

```
# include "design.h"      // archivos cabeceras que se crean

/*
    Funcien que dibuja un rect ngulo del color especificado
*/
void V_trabajo(int x1,int y1,int x2,int y2,int color,int fill);

/*
    Funcien que dibuja un boten en la pantalla
*/
void boton(int x1,int y1,int x2,int y2,int borde);

/*
    Funcien que dibuja el  rea de trabajo para dar efecto
    sumido, salido o parejo con la ventana de trabajo
*/
void A_trabajo(int x1,int y1,int x2,int y2,int color,int fill,int borde);

/*
    Funcien que escribe un texto en la pantalla de un color determinado
*/
void _textos(char *str,int x,int y,int color);

/*
    Funcien que escribe un texto en la pantalla de un color determinado
    y con la primer letra subrayada
*/
void _texto_s(char *str,int x,int y,int color);

void V_trabajo(int x1,int y1,int x2,int y2,int color,int fill)
{
    setfillstyle(fill,color);
    bar(x1,y1,x2,y2);
}

void boton(int x1,int y1,int x2,int y2,int borde)
{
    switch(borde)
```

```
    {
        case 0 : setcolor(BLACK);                    // Botén salido
                 rectangle(x1,y1,x2,y2);
                 setcolor(WHITE);
                 line(x1,y1,x2,y1);
                 line(x1,y2,x1,y1);
                 break;
        case 1 : setcolor(BLACK);                    // Botén sumido
                 rectangle(x1,y1,x2,y2);
                 setcolor(WHITE);
                 line(x2,y2,x2,y1);
                 line(x1,y2,x2,y2);
                 break;
        case 2 : setcolor(LIGHTGRAY);                // Botén parejo con el
                 rectangle(x1,y1,x2,y2);             //  rea de trabajo
                 break;
        default: return;
    }
}

void A_trabajo(int x1,int y1,int x2,int y2,int color,int fill,int borde)
{
    V_trabajo(x1,y1,x2,y2,color,fill);
    boton(x1,y1,x2,y2,borde);
}

void _textos(char *str,int x,int y,int color)
{
    setcolor(color);
    outtextxy(x,y,str);
}

void _texto_s(char *str,int x,int y,int color)
{
    setcolor(color);
    outtextxy(x,y,str);
    line(x,y+11,x+5,y+11);
}
```

Rutina que dibuja la reticula donde se dibujarán los patrones (RETICULA.CPP)

```
/*
    Funcien que inicializa los valores de la ret¡cula de entrada y de salida
    con valores de (-1)
*/
void inicia_reticula(void);

/*
    Funcien que muestra en la ret¡cula los patrones de un archivo de entrada
    con su correspondiente salida de la red neuronal
*/
```

```
void carga_reticula(void);

void inicia_reticula(void)
{
     int i, k, x, y;
     for(k=0;k<ENT_ALTURA;k++)
          for(i=0;i<ENT_BASE;i++)
          {
               x=10+CUADROS+CUADROS*i;
               y=CUADROS+CUADROS*k;
               A_trabajo(x,y,x+(CUADROS-2),y+(CUADROS-2),LIGHTGRAY,SOLID_FILL,1);
               boton(x,y,x+(CUADROS-2),y+(CUADROS-2),1);
               rectangle(x-1,y-1, x+(CUADROS-1), y+(CUADROS-1));
               entrada[i][k]=LIM_INF;
          }
     for(k=0;k<SAL_ALTURA;k++)
          for(i=0;i<SAL_BASE;i++)
          {
               x=160+CUADROS+CUADROS*i;
               y=CUADROS+CUADROS*k;
               A_trabajo(x,y,x+(CUADROS-2),y+(CUADROS-2),LIGHTGRAY,SOLID_FILL,1);
               boton(x,y,x+(CUADROS-2),y+(CUADROS-2),1);
               rectangle(x-1,y-1, x+(CUADROS-1), y+(CUADROS-1));
               salida[i][k]=LIM_INF;
          }
}

void carga_reticula(void)
{
     int i, j=0, k, x, y;
     for(k=0;k<ENT_ALTURA;k++)
          for(i=0;i<ENT_BASE;i++)
          {
               x=10+CUADROS+CUADROS*i;
               y=CUADROS+CUADROS*k;
               entrada[i][k]=U_patron_ent[j];
               j++;
               if(entrada[i][k]==1)
               {
                    A_trabajo(x,y,x+(CUADROS-2),y+(CUADROS-2),DARKGRAY,SOLID_FILL,1);
                    boton(x,y,x+(CUADROS-2),y+(CUADROS-2),0);
               }
               else
                    {
                         A_trabajo(x,y,x+(CUADROS-2),y+(CUADROS-
2),LIGHTGRAY,SOLID_FILL,1);
                         boton(x,y,x+(CUADROS-2),y+(CUADROS-2),1);
                    }
          }
     j=0;
     for(k=0;k<SAL_ALTURA;k++)
          for(i=0;i<SAL_BASE;i++)
          {
               x=160+CUADROS+CUADROS*i;
```

```
                y=CUADROS+CUADROS*k;
                salida[i][k]=U_patron_sal[j];
                j--;
                if(salida[i][k]==LIM_SUP)
                {
                    A_trabajo(x,y,x+(CUADROS-2),y+(CUADROS-2),DARKGRAY,SOLID_FILL,1);
                    boton(x,y,x+(CUADROS-2),y+(CUADROS-2),0);
                }
                else
                    {
                        A_trabajo(x,y,x+(CUADROS-2),y+(CUADROS-
2),LIGHTGRAY,SOLID_FILL,1);
                        boton(x,y,x+(CUADROS-2),y+(CUADROS-2),1);
                    }
            }
}
```

Rutina que dibuja a la cara, la cual representa al movil mecánico (CARITAX.CPP)

```
int   eje_x=290, eje_y=150, radio=30;
int   inicio_arco=180,fin_arco=360;

/*
    Funcien que dibuja la carita
*/
int carita(void);

/*
    Funcien que le da el color amarillo a la carita
*/
int carita_normal(void);

/*
    Funcien que le da el color rojo a la carita
*/
int carita_final(void);

/*
    Funcien que dibuja la carita en la pantalla de presentacien.
*/
int carita_presenta(void);

int carita_presenta(void)
{
    setfillstyle(SOLID_FILL,YELLOW);
    setcolor(BLACK);
    fillellipse(450,170,radio,radio);                    // Carita
    arc(450, 170, inicio_arco, fin_arco, radio/2);       // Boca
    setlinestyle(SOLID_LINE,SOLID_LINE,THICK_WIDTH);
    arc(450, 170, 30, 80, radio/2+3);                    // Ojo derecho
    arc(450, 170, 100, 150, radio/2+3);                  // Ojo izquierdo
```

```
        setlinestyle(SOLID_LINE,SOLID_LINE,NORM_WIDTH);
        return 0;
}

int carita(void)
{
        setcolor(BLACK);
        fillellipse(eje_x,eje_y,radio,radio);                    // Carita
        arc(eje_x, eje_y, inicio_arco, fin_arco, radio/2);       // Boca
        setlinestyle(SOLID_LINE,SOLID_LINE,THICK_WIDTH);
        arc(eje_x, eje_y, inicio_arco+210, fin_arco+80, radio/2+3);   // Ojo derecho
        arc(eje_x, eje_y, inicio_arco+280, fin_arco+150, radio/2+3);  // Ojo izquierdo
        setlinestyle(SOLID_LINE,SOLID_LINE,NORM_WIDTH);
        setviewport(0,0,getmaxx(),getmaxy(),CLIP_ON);
        return 0;
}

int carita_normal(void)
{
        setviewport(21,141,620,460,CLIP_ON);
        clearviewport();
        setfillstyle(SOLID_FILL,YELLOW);
        carita();
        return 0;
}

int carita_final(void)
{
        setviewport(21,141,620,460,CLIP_ON);
        clearviewport();
        setfillstyle(SOLID_FILL,RED);
        carita();
        return 0;
}
```

Rutina que calcula el espacio disponible en el Disco Duro (ESPACIO.CPP)[2]

```
/*
        Funcion que obtiene el espacio libre en disco duro
*/
void espacio(char *ptr);

void espacio(char *ptr)
{
        struct dfree free;
        char str[25]="\0";
        long avail;
        float temp;
```

[2] Programa desarrollado por Pedro López Urzúa. Facultad de Ingeniería. Departamento de Ingeniería de Control. Universidad Nacional Autónoma de México.

```
    int drive;

    drive = getdisk();
    getdfree(drive+1, &free);
    if (free.df_sclus == 0xFFFF)
        return;

    avail = (long) free.df_avail * (long) free.df_bsec * (long) free.df_sclus;

    temp=(float) avail/1048576;
    if(temp < 1.0)
        sprintf(str,"%.2f KBytes",(float) avail/1024);
    else
        sprintf(str,"%.2f MBytes",temp);
    strcpy(ptr,str);
}
```

Rutina que dibuja los controles del programa (BOTONES.CPP)

```
/*
    Funcion que indica cuando se selecciona un botón
*/
int boton_selec(int x1, int y1, int x2, int y2);

/*
    Funcion que dibuja un boton
*/
int dibuja_boton(int x1, int y1,int x2, int y2,
        int tipo, char *texto_n, char *texto_r);

/*
    Funcion que dibuja la ventana donde se despliegan
    los archivos
*/
int amb_archivos(void);

/*
    Funcion que dibuja la flecha hacia arriba en la
    ventana que despliega los archivos
*/
int flecha_arriba(void);

/*
    Funcion que dibuja la flecha hacia abajo en la
    ventana que despliega los archivos
*/
int flecha_abajo(void);

int flecha_arriba(void)
{
    int poly[8];
    setfillstyle(SOLID_FILL,BLACK);
```

```
        setlinestyle(SOLID_LINE,SOLID_LINE,NORM_WIDTH);
        setcolor(BLACK);
        poly[0] = 325; /* primer v,rtice  */
        poly[1] = 167;
        poly[2] = 322; /* segundo v,rtice */
        poly[3] = 173;
        poly[4] = 328; /* tercer v,rtice  */
        poly[5] = 173;
        poly[6] = 325; /* cuarto v,rtice  */
        poly[7] = 167;
        fillpoly(4,poly);
        return 0;
}

int flecha_abajo(void)
{
        int poly[8];
        setfillstyle(SOLID_FILL,BLACK);
        setlinestyle(SOLID_LINE,SOLID_LINE,NORM_WIDTH);
        setcolor(BLACK);
        poly[0] = 325; /* primer v,rtice  */
        poly[1] = 258;
        poly[2] = 322; /* segundo v,rtice */
        poly[3] = 252;
        poly[4] = 328; /* tercer v,rtice  */
        poly[5] = 252;
        poly[6] = 325; /* cuarto v,rtice  */
        poly[7] = 258;
        fillpoly(4,poly);
        return 0;
}

int amb_archivos(void)
{
        char ptr[20]="\0";

        V_trabajo(180,160,430,330,BLACK,SOLID_FILL);
        A_trabajo(180,160,430,330,LIGHTGRAY,SOLID_FILL,0);
        A_trabajo(185,165,315,260,WHITE,SOLID_FILL,1);

        dibuja_boton(345,180,405,195,0,"Abrir","A");
        dibuja_boton(345,205,405,220,0,"Guardar","G");
        dibuja_boton(345,230,405,245,0,"Eliminar","E");

        boton(320,165,330,175,0);
        flecha_arriba();

        boton(322,175,328,250,1);
        boton(322,176,328,180,0);

        boton(320,250,330,260,0);
        flecha_abajo();

        dibuja_boton(185,265,254,280,1,"Nombre","");
```

```
        dibuja_boton(256,265,315,280,1,"Tamaoo","");
        dibuja_boton(345,265,405,280,0,"Cerrar","C");

        dibuja_boton(185,290,315,305,1,"Nombre del Archivo","");
        A_trabajo(317,290,405,305,WHITE,SOLID_FILL,1);
        _textos("Espacio libre",190,312,BLACK);
        boton(185,310,295,325,1);
        boton(297,310,405,325,1);
        espacio(ptr);
        _textos(ptr,310,312,BLACK);
        return 0;
}

int boton_selec(int x1, int y1, int x2, int y2)
{
        mouse_off(1);
        boton(x1,y1,x2,y2,1);
        delay(150);
        boton(x1,y1,x2,y2,0);
        return 0;
}

int dibuja_boton(int x1, int y1,int x2, int y2, int tipo, char *texto_n, char *texto_r)
{
        setviewport(x1,y1,x2,y2,CLIP_ON);
        A_trabajo(0,0,x2-x1,y2-y1,LIGHTGRAY,SOLID_FILL,0);
        boton(0,0,x2-x1,y2-y1,tipo);
        _textos(texto_n,10,2,BLACK);
        _textos(texto_r,10,2,RED);
        setviewport(0,0,getmaxx(),getmaxy(),CLIP_ON);
        return 0;
}
```

Rutina que lee una cadena de caracteres y la despliega en las coordenadas dadas (CURSOR.CPP)

```
/*
        Funcien que lee los caracteres que se introducen
        para dar el nombre de un archivo
*/
int _leer_datos(int ndat,int xpos,int ypos,int tipo);

/*
        Funcien que indica cuando hay un error porque no
        existe un archivo que se quiere leer o que se
        quiere borrar, etc.
*/
int error(int x1, int y1, int x2, int y2, char *string);

int _leer_datos(int ndat,int xpos,int ypos,int tipo)
{
        int x=0,ptr,flag=-1,cond=1,xc;
```

```
char str[8]={{' '},{'\0'}};
char *string;

string=(char *) malloc(ndat+1);
mouse_off(1);
for(;flag==-1;)
{
      if(cond)
      {
            _textos("_",xpos,ypos,BLACK);
            cond=0;
            delay(90);
      }
      if(!cond)
      {
            _textos("_",xpos,ypos,WHITE);
            cond=1;
            delay(90);
      }
      if(bioskey(1))
      {
            ptr=bioskey(0);
            switch(ptr)
            {
                  case ESC :
                                    flag=1;
                                    strcpy(string,".");
                                    break;
                  case ENTER :
                                    if(tipo==1)
                                          for(xc=0;xc<strlen(string);xc++)
                                          {
                                                if ((48<=string[xc] && string[xc]<=57)||
                                                      (65<=string[xc] && string[xc]<=90)||
                                                      (97<=string[xc] && string[xc]<=122))
                                                   continue;
                                                else
                                                      {
                                                            error(170,50,405,120," Nombre no valido");
                                                            break;
                                                      }
                                          }
                                    if(!strlen(string))
                                    {
                                          error(170,50,410,120," Escribe el nombre");
                                          xc=-1;
                                    }
                                    else
                                          if(xc==strlen(string))
                                                flag=1;
                                    break;

                  case BKSPACE :
                                          if(!x)
```

```
                                        break;
                                    xpos-=6;
                                    x--;
                                    _textos(str,xpos,ypos,WHITE);
                                    string[x]='\0';
                                    str[0]=string[x-1];
                                    break;

                        default :
                                    if(x < ndat)
                                    {
                                        str[0]=(char) lo(ptr);
                                        if(33<=str[0] && str[0]<=126)
                                        {
                                            string[x]=str[0];x++;
                                            string[x]='\0';
                                            _textos(str,xpos,ypos,BLACK);
                                            xpos+=6;
                                        }
                                    }
                                    break;
                    }
                }
            }
            strcpy(nombre,string);
            mouse_off(1);
            free(string);
            return 0;
}

int error(int x1, int y1, int x2, int y2, char *string)
{
        unsigned key;
        int flag=-1;

        mouse_off(1);
        V_trabajo(x1,y1,x2,y2,DARKGRAY,SOLID_FILL);
        A_trabajo(x1+2,y1+2,x2-2,y2-2,LIGHTGRAY,SOLID_FILL,0);

        _textos("ERROR",x1+50,y1+10,BLACK);
        boton(x1+10,y1+25,x2-80,y1+26,1);
        _textos(string,x1+10,y1+45,BLACK);

        dibuja_boton(x2-55,y1+25,x2-25,y2-25,0,"Ok","O");

        for(;flag==-1;)
        {
            mouse_on(1);
            key = mouse_trigger(0);
            switch(key)
            {
                case LEFT_MOUSE_REL :  if (!mouse_in_box(x2-55,y1+25,x2-25,y2-25))
                                            break;
                case ENTER :
```

```
                case LETO :
                case LETo :
                                boton_selec(x2-55,y1+25,x2-25,y2-25);
                                flag=1;
                                break;
        }
    }
    mouse_off(1);
    V_trabajo(x1,y1,x2,y2,BLACK,SOLID_FILL);
    return 0;
}
```

Rutinas que muestran la dirección del móvil mecánico en pantalla (ADDRESS.CPP)

```
/*
    Funcien que dibuja los 8 puntos cardinales
    en que se puede mover la carita
*/
int direccion(void);

/*
    Funcien que indica la direccien que tiene la
    carita en un instante determinado
*/
int que_direccion();

int direccion(void)
{
    // 135 grados
    V_trabajo(9,129,19,139,RED,SOLID_FILL);
    A_trabajo(10,130,18,138,LIGHTGRAY,SOLID_FILL,0);
    boton(10,130,18,138,1);
    // 90 grados
    V_trabajo(315,129,325,139,RED,SOLID_FILL);
    A_trabajo(316,130,324,138,LIGHTGRAY,SOLID_FILL,0);
    boton(316,130,324,138,1);
    // 45 grados
    V_trabajo(622,129,632,139,RED,SOLID_FILL);
    A_trabajo(623,130,631,138,LIGHTGRAY,SOLID_FILL,0);
    boton(623,130,631,138,1);
    // 180 grados
    V_trabajo(9,295,19,305,RED,SOLID_FILL);
    A_trabajo(10,296,18,304,LIGHTGRAY,SOLID_FILL,0);
    boton(10,296,18,304,1);
    // 0/360 grados
    V_trabajo(622,295,632,305,RED,SOLID_FILL);
    A_trabajo(623,296,631,304,LIGHTGRAY,SOLID_FILL,0);
    boton(623,296,631,304,1);
    // 225 grados
    V_trabajo(9,462,19,472,RED,SOLID_FILL);
    A_trabajo(10,463,18,471,LIGHTGRAY,SOLID_FILL,0);
```

```
        boton(10,463,18,471,1);
        // 270 grados
        V_trabajo(315,462,325,472,RED,SOLID_FILL);
        A_trabajo(316,463,324,471,LIGHTGRAY,SOLID_FILL,0);
        boton(316,463,324,471,1);
        // 315 grados
        V_trabajo(622,462,632,472,RED,SOLID_FILL);
        A_trabajo(623,463,631,471,LIGHTGRAY,SOLID_FILL,0);
        boton(623,463,631,471,1);
        return 0;
}

int que_direccion()
{
     if (inicio_arco==0)
     {
          A_trabajo(316,130,324,138,RED,SOLID_FILL,0);
          boton(316,130,324,138,0);
     }
     if (inicio_arco==45)
     {
          A_trabajo(10,130,18,138,RED,SOLID_FILL,0);
          boton(10,130,18,138,0);
     }
     if (inicio_arco==90)
     {
          A_trabajo(10,296,18,304,RED,SOLID_FILL,0);
          boton(10,296,18,304,0);
     }
     if (inicio_arco==135)
     {
          A_trabajo(10,463,18,471,RED,SOLID_FILL,0);
          boton(10,463,18,471,0);
     }
     if (inicio_arco==180)
     {
          A_trabajo(316,463,324,471,RED,SOLID_FILL,0);
          boton(316,463,324,471,0);
     }
     if (inicio_arco==225)
     {
          A_trabajo(623,463,631,471,RED,SOLID_FILL,0);
          boton(623,463,631,471,0);
     }
     if (inicio_arco==270)
     {
          A_trabajo(623,296,631,304,RED,SOLID_FILL,0);
          boton(623,296,631,304,0);
     }
     if (inicio_arco==315)
     {
          A_trabajo(623,130,631,138,RED,SOLID_FILL,0);
          boton(623,130,631,138,0);
     }
```

```
    return 0;
}
```

Rutina que despliega en pantalla las opciones a elegir así como la pantalla principal (MENUS.CPP)

```
/*
    Funcien que despliega el t¡tulo de Patrones de entrada y el de Salida
*/
int reticula(void);

/*
    Funcien que dibuja el marco delimitador de la carita en el programa
*/
int marco_carita(void);

/*
    Funcien que dibuja las cajas de aviso utilizadas en algunas partes
    del programa
*/
int caja_de_avisos(void);

/*
    Funcien que muestra las opciones de Aceptar o Borrar un patr€n dibujado
*/
int menu_entrenamiento(void);

/*
    Funcien que dibuja la barra del men£ principal
*/
int menu_principal(void);

/*
    Funcien que dibuja la barra del men£ principal en letras iniciales rojas
*/
int menu_rojo(void);

/*
    Funcien que dibuja la barra del men£ principal con letras iniciales
    negras excepto Archivos
*/
int menu_archivos(void);

/*
    Funcien que dibuja la barra del men£ principal con letras iniciales
    negras excepto Ejecutar
*/
int menu_ejecutar(void);

int reticula(void)
{
    _textos("Patrones de entrada",20,1,LIGHTGRAY);
```

```
    _textos("Salida",180,1,LIGHTGRAY);
    return 0;
}

int marco_carita(void)
{
    setbkcolor(BLACK);
    V_trabajo(20,140,621,461,DARKGRAY,SOLID_FILL);
    A_trabajo(21,141,620,460,LIGHTGRAY,SOLID_FILL,0);
    boton(22,142,619,459,2);
    return 0;
}

int caja_de_avisos(void)
{
    V_trabajo(170,90,370,125,DARKGRAY,SOLID_FILL);
    A_trabajo(172,92,368,123,LIGHTGRAY,SOLID_FILL,0);
    boton(175,95,365,120,2);
    return 0;
}

int menu_entrenamiento(void)
{
    dibuja_boton(270,5,341,20,0,"Archivos","");
    dibuja_boton(414,5,485,20,0,"Ejecutar","");

    V_trabajo(425,35,600,80,DARKGRAY,SOLID_FILL);
    A_trabajo(427,37,598,78,LIGHTGRAY,SOLID_FILL,0);

    dibuja_boton(435,40,500,55,0,"Aceptar","A");
    dibuja_boton(435,60,500,75,0,"Borrar","B");
    boton(510,40,595,75,1);
    _textos("Botón  Botón",520,42,BLACK);
    _textos(" Izq.  Der.",520,52,BLACK);
    _textos("  1     0  ",520,62,RED);
    return 0;
}

int menu_principal(void)
{
    V_trabajo(265,0,635,25,DARKGRAY,SOLID_FILL);
    A_trabajo(267,2,633,23,LIGHTGRAY,SOLID_FILL,0);

    dibuja_boton(270,5,341,20,0,"Archivos","A");
    dibuja_boton(342,5,413,20,0,"Programar","P");
    dibuja_boton(414,5,485,20,0,"Ejecutar","E");
    dibuja_boton(486,5,557,20,0," Ayuda","   y");
    dibuja_boton(559,5,630,20,0," Salir","  S");
    direccion();
    que_direccion();
    return 0;
}

int menu_rojo(void)
```

```
{
      dibuja_boton(270,5,341,20,0,"Archivos","A");
      dibuja_boton(342,5,413,20,0,"Programar","P");
      dibuja_boton(414,5,485,20,0,"Ejecutar","E");
      dibuja_boton(486,5,557,20,0," Ayuda"," y");
      dibuja_boton(559,5,630,20,0," Salir"," S");
      return 0;
}

int menu_archivos(void)
{
      dibuja_boton(342,5,413,20,0,"Programar","");
      dibuja_boton(414,5,485,20,0,"Ejecutar","");
      dibuja_boton(486,5,557,20,0," Ayuda","");
      dibuja_boton(559,5,630,20,0," Salir","");
      return 0;
}

int menu_ejecutar(void)
{
      dibuja_boton(270,5,341,20,0,"Archivos","");
      dibuja_boton(342,5,413,20,0,"Programar","");
      dibuja_boton(486,5,557,20,0," Ayuda","");
      dibuja_boton(559,5,630,20,0," Salir","");
      return 0;
}
```

Rutina que guarda en disco el patrón introducido por el mouse (SALVARET.CPP)

```
/*
      Funcien que almacena los patrones guardados en el archivo temporal
      y borra el archivo temporal
*/
int  salvar_arch(void);

/*
      Funcien que guarda los patrones dibujados en un archivo temporal
      para su manipulacien en el programa
*/
void arch_temporal(void);

/*
      Funcien que pregunta si se quieren guardar los patrones dibujados
      en un archivo
*/
int  salvar(void);

char *extension=".pat";
char nombre_temporal[12]=" atrones.$$$";

int salvar(void)
```

```
{
    int resp=FALSE;
    if (strcmpi(nombre_temporal,"patrones.SSS")==0)
    {
        int don=FALSE;
        dibuja_boton(400,100,430,115,0,"Si","S");
        dibuja_boton(440,100,470,115,0,"No","N");
        caja_de_avisos();
        _textos("Deseas almacenar los patrones ",180,97,BLACK);
        _textos("que estan en memoria a disco? ",180,107,BLACK);
        do
        {
            mouse_on(1);
            unsigned key = mouse_trigger(0);
            if(mouse_in_box(405,100,430,115) &&
                    (key==LEFT_MOUSE_REL) || (key==LETS||key==LETs))
            {
                //Almacenar patrones
                mouse_off(1);
                V_trabajo(170,90,370,125,BLACK,SOLID_FILL);
                boton_selec(400,100,430,115);
                don=TRUE;
                resp=TRUE;
            }
            else
                if(mouse_in_box(440,100,470,115) && (key==LEFT_MOUSE_REL) ||
                    (key==LETN||key==LETn))
                {
                    //No almacenar patrones
                    mouse_off(1);
                    V_trabajo(170,90,370,125,BLACK,SOLID_FILL);
                    boton_selec(440,100,470,115);
                    remove ("patrones.SSS");
                    don=TRUE;       // Salir
                }
        }while(don!=TRUE);
        V_trabajo(400,100,475,115,BLACK,SOLID_FILL);
    }
    return resp;
}

int salvar_arch(void)
{
    int resp=FALSE;
    if (strcmpi(nombre_temporal,"patrones.SSS")==0)
    {
        _leer_datos(8,322,292,1);
        strcat(nombre,extension);
        if (nombre[0]!='.')
        {
            if (rename(nombre_temporal,nombre)==0)
            {
                strcpy(nombre_temporal,"patrones.SSS");
                nombre_temporal[0]=' ';
```

```
                    resp=TRUE;
                }
                else
                {
                    error(145,50,410,120," Ese nombre ya existe.");
                    strcpy(nombre,"patrones.pat");
                }
            }
        }
        else
        {
            error(145,50,410,120," No hay patrones a guardar.");
            resp=TRUE;
        }
        return resp;
}

void arch_temporal(void)
{
        if (nombre_temporal[0]==' ')
        {
            nombre_temporal[0]='p';
            ofstream f_out(nombre_temporal, ios::out);
            f_out << SIZE_ENTRADA << "\n";
            f_out << SIZE_SALIDA;
            f_out.close();
        }
        ofstream f_out(nombre_temporal, ios::out|ios::app);
        f_out << "\n";
        for(int k=0;k<ENT_ALTURA;k++)
            for(int i=0;i<ENT_BASE;i++)
            {
                if(entrada[i][k]==LIM_SUP)
                    f_out << "  1.00";
                else
                    f_out << " -1.00";
            }
        f_out.close();
}
```

Rutina que muestra el movimiento a realizar por el móvil (MVCARITA.CPP)

```
/*
    Funcion que realiza las comparaciones necesarias para
    que se mueva la carita en la pantalla y manda los datos
    necesarios al puerto paralelo para que se mueva el movil
    mecanico
*/
int salida_pantalla(void);

/*
```

```
      Funcien que va almacenando en memoria el conjunto de las
      salidas de la red neuronal, para que posteriormente se
      se ejecuten los movimientos
*/
int salidas(void);

#define INSTRUC 100

float salida_0[INSTRUC], salida_1[INSTRUC];
float salida_2[INSTRUC], salida_3[INSTRUC];
int   w=0, tiempo=0;

int salida_pantalla(void)
{
     for(int i=0;i<w;i++)
     {
          unsigned key = mouse_trigger(0);
          switch(key)
          {
               case LETP :
               case LETp :   boton_selec(490,100,540,115);
                             carita_final();
                             delay(1000);
                             carita_normal();
                             i=w;
                             w=0;
                             outp(888,0);
                             break;
               default :

                             if((salida_0[i]==LIM_INF)&&(salida_1[i]==LIM_INF)&&
                                  (salida_2[i]==LIM_SUP)&&(salida_3[i]==LIM_SUP))
                             {
                                  w=0;
                                  outp(888,0);
                             }
                             tiempo=6;
                             if((salida_0[i]==LIM_INF)&&(salida_1[i]==LIM_SUP)&&
                                  (salida_2[i]==LIM_INF)&&(salida_3[i]==LIM_SUP))
                             {
                                  tiempo=6;
                                  i--;
                             }
                             if((salida_0[i]==LIM_INF)&&(salida_1[i]==LIM_SUP)&&
                                  (salida_2[i]==LIM_SUP)&&(salida_3[i]==LIM_INF))
                             {
                                  tiempo=1;
                                  i--;
                             }
                             if((salida_0[i]==LIM_INF)&&(salida_1[i]==LIM_SUP)&&
                                  (salida_2[i]==LIM_SUP)&&(salida_3[i]==LIM_SUP))
                             {
                                  tiempo=2;
                                  i--;
```

```
}
if((salida_0[i]==LIM_SUP)&&(salida_1[i]==LIM_INF)&&
    (salida_2[i]==LIM_INF)&&(salida_3[i]==LIM_INF))
{
    tiempo=3;
    i--;
}
if((salida_0[i]==LIM_SUP)&&(salida_1[i]==LIM_INF)&&
    (salida_2[i]==LIM_INF)&&(salida_3[i]==LIM_SUP))
{
    tiempo=4;
    i--;
}
for(int j=0;j<tiempo;j++)
{
    unsigned key = mouse_trigger(0);
    switch(key)
    {
        case LETP :
        case LETp :
        case ESC  :
                    boton_selec(490,100,540,115);
                    carita_final();
                    delay(1000);
                    carita_normal();
                    i=w;
                    j=tiempo;
                    w=0;
                    outp(888,0);
                    break;
        default :
                    direccion();
                    if ((salida_0[i]==LIM_INF)&&
                       (salida_1[i]==LIM_INF)&&
                       (salida_2[i]==LIM_INF))
                    {
                        if (salida_3[i]==LIM_INF)
                        {
                            que_direccion();
                            if ((inicio_arco==0)&&(eje_y>radio+10))
                                eje_y-=MOV;
                            if (inicio_arco==45)
                            {
                                if (eje_x>radio+10)
                                    eje_x-=MOV;
                                if (eje_y>radio+10)
                                    eje_y-=MOV;
                            }
                            if ((inicio_arco==90)&&(eje_x>radio+10))
                                eje_x-=MOV;
                            if (inicio_arco==135)
                            {
                                if (eje_x>radio+10)
                                    eje_x-=MOV;
```

```
                    if (eje_y<getmaxy()-200)
                        eje_y+=MOV;
                }
                if ((inicio_arco==180)&&
                   (eje_y<getmaxy()-200))
                    eje_y+=MOV;
                if (inicio_arco==225)
                {
                    if (eje_x<getmaxx()-80)
                        eje_x+=MOV;
                    if (eje_y<getmaxy()-200)
                        eje_y+=MOV;
                }
                if ((inicio_arco==270)&&
                   (eje_x<getmaxx()-80))
                    eje_x+=MOV;
                if (inicio_arco==315)
                {
                    if (eje_x<getmaxx()-80)
                        eje_x+=MOV;
                    if (eje_y>radio+10)
                        eje_y-=MOV;
                }
                outp(888,0);
                outp(888,5);                    // Adelante
            }
            else
                if (salida_3[i]==LIM_SUP)
                {
                    que_direccion();
                    if ((inicio_arco==0)&&
                       (eje_y<getmaxy()-200))
                        eje_y+=MOV;
                    if (inicio_arco==45)
                    {
                        if (eje_x<getmaxx()-80)
                            eje_x+=MOV;
                        if (eje_y<getmaxy()-200)
                            eje_y+=MOV;
                    }
                    if ((inicio_arco==90)&&
                       (eje_x<getmaxx()-80))
                        eje_x-=MOV;
                    if (inicio_arco==135)
                    {
                        if (eje_x<getmaxx()-80)
                            eje_x+=MOV;
                        if (eje_y>radio+10)
                            eje_y-=MOV;
                    }
                    if ((inicio_arco==180)&&
                       (eje_y>radio+10))
                        eje_y-=MOV;
                    if (inicio_arco==225)
```

```
                                        {
                                            if (eje_x>radio+10)
                                                eje_x-=MOV;
                                            if (eje_y>radio+10)
                                                eje_y-=MOV;
                                        }
                                        if ((inicio_arco==270)&&
                                           (eje_x>radio+10))
                                            eje_x-=MOV;
                                        if (inicio_arco==315)
                                        {
                                            if (eje_x>radio+10)
                                                eje_x-=MOV;
                                            if (eje_y<getmaxy()-200)
                                                eje_y+=MOV;
                                        }
                                        outp(888,0);
                                        outp(888,10);              // Atras
                                    }
                                }
                                if((salida_0[i]==LIM_INF)&&
                                  (salida_1[i]==LIM_SUP)&&
                                  (salida_2[i]==LIM_INF)&&
                                  (salida_3[i]==LIM_INF))
                                // Vuelta (A la izquierda - sentido antihorario)
                                {
                                    inicio_arco+=45;
                                    fin_arco+=45;
                                    if (inicio_arco>=360)
                                        inicio_arco-=360;
                                    if (fin_arco>=360)
                                        fin_arco-=360;
                                    que_direccion();
                                    outp(888,0);
                                    outp(888,6);
                                }
                                que_direccion();
                                if((salida_0[i]==LIM_INF)&&
                                  (salida_1[i]==LIM_INF)&&
                                  (salida_2[i]==LIM_SUP)&&
                                  (salida_3[i]==LIM_SUP))
                                    carita_final();
                                else
                                    carita_normal();
                                que_direccion();
                                delay(1000);
                                if (tiempo==6)
                                    j=7;
                                break;
                        }
                    }
                    if ((tiempo==1)||(tiempo==2)||(tiempo==3)||(tiempo==4)||(tiempo==5))
                    i++;
                    break;
```

```
            }
        }
        return 0;
}

int salidas(void)
{
        if ((U_patron_sal[0]==LIM_INF)&&(U_patron_sal[1]==LIM_INF)&&
            (U_patron_sal[2]==LIM_SUP)&&(U_patron_sal[3]==LIM_INF))
            salida_pantalla();
        else
        {
            salida_0[w]=U_patron_sal[0];
            salida_1[w]=U_patron_sal[1];
            salida_2[w]=U_patron_sal[2];
            salida_3[w]=U_patron_sal[3];
            w++;
        }
        return 0;
}
```

Rutina que realiza el algoritmo de propagación hacia adelante (RETROPRO.CPP)

```
/*
    Función que lee el archivo de pesos que nos entrega el programa
    DynaMind
*/
void lectura_pesos(void);

/*
    Función que realiza el Algoritmo de Propagación hacia Adelante
*/
void retropropagacion(void);

float pesos[SIZE_PESOS];
float U_capa_int[SIZE_CAPA_INT],S_capa_int[SIZE_CAPA_INT];
float S_patron_sal[SIZE_SALIDA];
float weight;

void lectura_pesos(void)
{
    ifstream file_in("pesos.net");
    int i=0;
    while (!file_in.eof())
    {
        file_in >> weight;
        pesos[i]=weight;
        i++;
    }
    file_in.close();
}
```

```
void retropropagacion(void)
{
// capa intermedia
      for (int i=0;i<SIZE_CAPA_INT;i++)
            S_capa_int[i]=0;
      int j=0;
      for (int k=0;k<SIZE_CAPA_INT;k++)
            for (i=0;i<SIZE_ENTRADA+1;i++)
            {
                  S_capa_int[k]=S_capa_int[k]+U_patron_ent[i]*pesos[j];
                  j++;
            }
      for (i=0;i<SIZE_CAPA_INT;i++)
      {
            double x=exp(-S_capa_int[i]);          // Funcién Sigmoide
            U_capa_int[i]=1/(1+x);
      }
      U_capa_int[SIZE_CAPA_INT]=1;
// capa de salida
      for (i=0;i<SIZE_SALIDA;i++)
            S_patron_sal[i]=0;
      for (k=0;k<SIZE_SALIDA;k++)
            for (i=0;i<SIZE_CAPA_INT+1;i++)
            {
                  S_patron_sal[k]=S_patron_sal[k]+U_capa_int[i]*pesos[j];
                  j++;
            }
      for (i=0;i<SIZE_SALIDA;i++)
      {
            double y=exp(-S_patron_sal[i]);          // Funcién Sigmoide
            U_patron_sal[i]=1/(1+y);
            if (U_patron_sal[i]<MITAD)
                  U_patron_sal[i]=LIM_INF;
            else
                  U_patron_sal[i]=LIM_SUP;
      }
}
```

Rutina que lee un patrón almacenado anteriormente en disco y /o en memoria (LEERET.CPP)

```
/*
      Funcien que ejecuta un conjunto de funciones
*/
int procedimientos(void);

/*
      Funcien que va leyendo cada uno de los patrones almacenados, ya sea
      en el archivo temporal o en un archivo con patrones almacenados
      con anterioridad
*/
```

```
int avanza(void);

/*
      Funcion que detecta si el archivo de patrones almacenado que se quiere
      leer corresponde al programa
*/
int carga_patrones(void);

/*
      Funcion que detecta si se dibujaron patrones y se guardaron en un
      archivo temporal
*/
int carga_patrones_temporal(void);

int procedimientos(void)
{
      retropropagacion();
      salidas();
      if (!((U_patron_sal[0]==LIM_INF)&&(U_patron_sal[1]==LIM_INF)&&
            (U_patron_sal[2]==LIM_SUP)&&(U_patron_sal[3]==LIM_INF)))
      {
            carga_reticula();
            delay(1000);
      }
      else
            carita_normal();
      inicia_reticula();
      return 0;
}

int avanza(void)
{
      char nombre2[7];

      ifstream file_in(nombre_temporal,ios::in);
      file_in >> nombre2;
      file_in >> nombre2;
      V_trabajo(565,95,625,120,BLACK,SOLID_FILL);
      dibuja_boton(490,100,540,115,0,"Parar","P");
      int don=FALSE;
      do
      {
            int j=0;
            for(int k=0;k<ENT_ALTURA;k++)
                  for(int i=0;i<ENT_BASE;i++)
                  {
                        file_in >> nombre2;
                        if(strcmp(nombre2,"1.00")==0)
                        {
                              entrada[i][k]=LIM_SUP;
                              U_patron_ent[j]=LIM_SUP;
                        }
                        else
                              {
```

```
                                entrada[i][k]=LIM_INF;
                                U_patron_ent[j]=LIM_INF;
                            }
                        j++;
                    }
                U_patron_ent[SIZE_ENTRADA]=1;
                unsigned key = mouse_trigger(0);
                switch(key)
                {
                    case LETP :
                    case LETp :
                    case ESC  :
                                boton_selec(490,100,540,115);
                                w=0;
                                outp(888,0);
                                don=TRUE;
                                break;
                    default :
                                procedimientos();
                                don=FALSE;
                                break;
                }
            }while ((!file_in.eof())&&(don!=TRUE));
            file_in.close();
            V_trabajo(490,100,540,115,BLACK,SOLID_FILL);
            w=0;
            outp(888,0);
            return 0;
}

int carga_patrones(void)
{
    int no_ent=0, no_sal=0, datos;
    int resp=FALSE;

    V_trabajo(317,290,405,305,WHITE,SOLID_FILL);
    A_trabajo(317,290,405,305,WHITE,SOLID_FILL,1);
    _leer_datos(8,322,292,1);
    strcat(nombre,extension);
    if (nombre[0]!='.')
    {
        ifstream file_in(nombre,ios::in|ios::nocreate);
        if (file_in.good())
        {
            file_in >> datos;
            no_ent=datos;
            file_in >> datos;
            no_sal=datos;
            if((no_ent!=(ENT_BASE*ENT_ALTURA))||(no_sal!=(SAL_BASE*SAL_ALTURA)))
            {
                file_in.close();
                error(145,50,410,120,"No corresponde al programa ...");
            }
            else
```

```
                {
                    strcpy(nombre_temporal,nombre);
                    V_trabajo(180,160,410,330,BLACK,SOLID_FILL);
                    carita_normal();
                    avanza();
                    resp=TRUE;
                }
            }
            else
                error(170,50,405,120," El archivo no existe");
        }
        return resp;
}

int carga_patrones_temporal(void)
{
        int datos;

        if (nombre_temporal[0]==' ')
            error(170,50,405,120," No hay patrones en memoria");
        else
        {
            ifstream file_in(nombre_temporal,ios::in|ios::nocreate);
            file_in >> datos;
            file_in >> datos;
            avanza();
        }
        return 0;
}
```

Programa que muestra los archivos de patrones almacenados en el disco (LISTA.CPP)

```
char string[MAX_FILES][21];

/*
    Funcion que obtiene la informacion de todos los archivos con
    la extension ".pat"
*/
int  informa();

/*
    Funcion que muestra la informacion de los archivos con la
    extension ".pat"
*/
void despliega_inf(int ndat);

/*
    Funcion que ordena alfabeticamente los archivos de extension ".pat"
*/
int  sort_function(const void *a, const void *b);
```

```
/*
     Funcien que borra un archivo
*/
int  sup_archivos(void);

/*
     Funcien que despliega en la ventana los archivos con extensién ".pat"
*/
void ver_archivos(void);

/*
     Funcien que refresca la ventana donde se muestran los archivos de
     extensién ".pat"
*/
int  ventana(void);

int informa()
{
     struct ffblk ffblk;
     int done,x=0;

     done = findfirst("*.pat",&ffblk,0);
     while (!done)
     {
          x++;
          done = findnext(&ffblk);
     }
     return x;
}

void despliega_inf(int ndat)
{
     struct ffblk ffblk;
     char *ptr,backup[10]="\0";
     int done,x=0,y;

     done = findfirst("*.pat",&ffblk,0);
     while (!done && ndat < MAX_FILES)
     {
          string[x][0]='\0';
          strcpy(string[x],ffblk.ff_name);
          ptr=strchr(string[x],'.');
          string[x][ptr-string[x]]='\0';
          for(y=strlen(string[x]);y < 13; y++)
               string[x][y]=' ';
          string[x][13]='\0';
          ltoa(ffblk.ff_fsize,backup,10);
          strcat(string[x],backup);
          string[x][strlen(string[x])]='\0';
          strcat(string[x],"\n");
          x++;
          done = findnext(&ffblk);
          backup[0]='\0';
     }
          *
```

```
        qsort((void *)string, ndat, sizeof(string[0]), sort_function);
    }

    int sort_function(const void *a, const void *b)
    {
        return( strcmp((char *)a,(char *)b) );
    }

    int sup_archivos()
    {
        char string[13]="\0";
        char sptr[20]="\0";
        int  resp=FALSE;

        _leer_datos(8,322,292,1);
        strcpy(string,nombre);
        strcat(string,".pat");
        if(strlen(string))
        {
            if(searchpath(string)!=NULL)
            {
                remove(string);
                resp=TRUE;
            }
            else
                error(145,50,410,120,"Da el nombre");
        }

        mouse_off(1);
        A_trabajo(317,290,405,305,WHITE,SOLID_FILL,1);
        V_trabajo(298,311,404,324,LIGHTGRAY,SOLID_FILL);
        espacio(sptr);
        _textos(sptr,310,312,BLACK);

        return resp;
    }

    int ventana(void)
    {
        V_trabajo(185,165,315,260,WHITE,SOLID_FILL);
        boton(185,165,315,260,1);
        V_trabajo(322,175,328,250,LIGHTGRAY,SOLID_FILL);
        boton(322,175,328,250,1);
        return 0;
    }

    void ver_archivos(void)
    {
        int don=FALSE,x,cont=0,y=0;
        int numero, multiplo;

        numero=informa();
        despliega_inf(numero);
        mouse_off(1);
```

```
x=0;
amb_archivos();
do
{
      mouse_on(1);
      numero=informa();
      if (numero>6)
            multiplo=75/(numero-6);
      else
            multiplo=2;
      unsigned key= mouse_trigger(0);
      if((key==LEFT_MOUSE_REL && mouse_in_box(345,265,405,280)) ||
            key== ESC || key== LETC || key== LETc )
            // Cerrar la ventana
      {
            boton_selec(345,265,405,280);
            V_trabajo(180,160,410,330,BLACK,SOLID_FILL);

            carita_normal();
            don=TRUE;
      }
      if((mouse_in_box(320,250,330,260) && key==LEFT_MOUSE_REL) ||
            key==DARROW || key==RARROW)
            // Avanzar en la lista
      {
            if(x < numero)
            {
                  cont=0;
                  y++;
                  x=y;
                  boton_selec(320,250,330,260);
                  ventana();
                  boton(322,176+x*multiplo,328,176+(1+x)*multiplo,0);
            }
            else
                  continue;
      }
      if((mouse_in_box(320,165,330,175)&&key==LEFT_MOUSE_REL) ||
            key==UARROW || key==LARROW)
            // Retroceder en la lista
      {
            if(!y)
                  continue;
            cont=0;
            y--;
            x=y;
            boton_selec(320,165,330,175);
            ventana();
            boton(322,176+x*multiplo,328,176+(1+x)*multiplo,0);
      }
      if((mouse_in_box(345,180,405,195)&&key==LEFT_MOUSE_REL) ||
            key==LETA || key==LETa)
            // Abrir un archivo
      {
```

```
        mouse_off(1);
        boton(345,180,405,195,1);
        if (carga_patrones())
            don=TRUE;
        else
        {
            V_trabajo(317,290,405,305,WHITE,SOLID_FILL);
            A_trabajo(317,290,405,305,WHITE,SOLID_FILL,1);
            boton(345,180,405,195,0);
        }
    }
    if((mouse_in_box(345,205,405,220)&&key==LEFT_MOUSE_REL) ||
        key==LETG || key==LETg)
        // Guardar un archivo
    {
        mouse_off(1);
        boton(345,205,405,220,1);
        if (salvar_arch())
        {
            boton(345,205,405,220,0);
            despliega_inf(numero);
            x=cont=0;
            ventana();
            boton(322,176,328,176+multiplo,0);
        }
        else
        {
            boton(345,205,405,220,0);
        }
        V_trabajo(317,290,405,305,WHITE,SOLID_FILL);
        A_trabajo(317,290,405,305,WHITE,SOLID_FILL,1);
    }
    if((mouse_in_box(345,230,405,245) && key==LEFT_MOUSE_REL) ||
        key==LETE || key==LETe)
        // Eliminar un archivo
    {
        mouse_off(1);
        boton(345,230,405,245,1);
        if(sup_archivos())
        {
            boton(345,230,405,245,0);
            despliega_inf(numero);
            x=cont=0;
            ventana();
            boton(322,176,328,176-multiplo,0);
        }
        else
            boton(345,230,405,245,0);
    }
    if(x < numero && cont < 7)
    {
        mouse_off(1);
        _textos(string[x],190,165-13*cont,BLACK);
        x++;
```

```
            cont--;
        }
    }while(don!=TRUE);
}
```

Rutina que realiza las opciones de los menús (OPMENUS.CPP)

```
/*
    Funcien que hace las comparaciones necesarias para que se
    ejecute cualquiera de las opciones del men£ de entrenamiento
*/
int opciones_menu_entrenamiento(void);

/*
    Funcien que hace las comparaciones necesarias para que se
    ejecute cualquiera de las opciones del men£ principal
*/
int opciones_menu_principal(void);

int opciones_menu_entrenamiento(void)
{
    int don=FALSE;
    do
    {
        mouse_on(1);
        unsigned key = mouse_trigger(0);
        if (mouse_in_box(435,40,500,55) &&
                (key==LEFT_MOUSE_REL) || (key==LETA||key==LETa))
        {                                   //A - Almacena patron en memoria
            boton_selec(435,40,500,55);
            arch_temporal();
            inicia_reticula();
        }
        else
            if (mouse_in_box(435,60,500,75) &&
                    (key==LEFT_MOUSE_REL) || (key==LETB || key==LETb))
            {
                boton_selec(435,60,500,75);
                inicia_reticula();      //B - Borra reticula
            }
            else
                if(mouse_in_box(486,5,557,20) &&
                        (key==LEFT_MOUSE_REL) || (key==LETY||key==LETy))
                {                                   // Ayuda
                    boton_selec(486,5,557,20);
                    ayuda();
                    menu_entrenamiento();
                    marco_carita();
                    carita_normal();
                    direccion();
                    que_direccion();
                }
```

```
        . else
              if (mouse_in_box(559,5,630,20) && (key==LEFT_MOUSE_REL) ||
                   (key==ESC || key==LETS || key==LETs))
              {
                   boton_selec(559,5,630,20);
                   inicia_reticula();
                   w=0;
                   outp(888,0);
                   nombre[0]=' ';
                   V_trabajo(425,35,600,100,BLACK,SOLID_FILL);
                   don=TRUE;          // S - Salir
              }
if (button_press(0))
{
     int x,y;
     mouse_off(1);
     if (mouse_in_box(10+CUADROS,CUADROS,10+CUADROS*(ENT_BASE+1)-2,
          CUADROS*(ENT_ALTURA+1)-2))
          do
          {
               button_state();
               for(int k=0; k<ENT_ALTURA; k++)
                    for(int i=0; i<ENT_BASE;i++)
                    {
                         x=10+CUADROS+CUADROS*i;
                         y=CUADROS+CUADROS*k;

                         if (mouse_in_box(x,y,x+(CUADROS-2),y+(CUADROS-2)))
                         {
                              A_trabajo(x,y,x+(CUADROS-2),y+(CUADROS-2),
                                        DARKGRAY,SOLID_FILL,1);
                              boton(x,y,x+(CUADROS-2),y+(CUADROS-2),0);
                              entrada[i][k]=LIM_SUP;
                         }
                    }
          }while(!button_release(0));
}
if (button_press(1))
{
     int x,y;
     mouse_off(1);
     if (mouse_in_box(10+CUADROS,CUADROS,10+CUADROS*(ENT_BASE+1)-2,
          CUADROS*(ENT_ALTURA+1)-2))
          do
          {
               button_state();
               for(int k=0; k<ENT_ALTURA; k++)
                    for(int i=0; i<ENT_BASE;i++)
                    {
                         x=10+CUADROS+CUADROS*i;
                         y=CUADROS+CUADROS*k;
                         if (mouse_in_box(x, y, x+(CUADROS-2), y+(CUADROS-2)))
                         {
                              A_trabajo(x,y,x+(CUADROS-2),y+(CUADROS-2),
```

```
                                        LIGHTGRAY,SOLID_FILL,1);
                                    boton(x,y,x+(CUADROS-2),y+(CUADROS-2),1);
                                    entrada[i][k]=LIM_INF;
                                }
                            }
                        }while(!button_release(1));
                }
        }while(don!=TRUE);
        menu_rojo();
        return 0;
}

int opciones_menu_principal(void)
{
        int don=FALSE;
        do
        {
                mouse_on(1);
                unsigned key = mouse_trigger(0);

                if (mouse_in_box(270,5,341,20) && (key==LEFT_MOUSE_REL) ||
                        (key==LETA||key==LETa))
                {
                        //Manipulacion de archivos de entrenamiento
                        mouse_off(1);
                        boton(270,5,341,20,1);
                        menu_archivos();
                        ver_archivos();
                        boton(270,5,341,20,0);
                        menu_rojo();
                        don=TRUE;
                }
                else
                        if (mouse_in_box(342,5,413,20) && (key==LEFT_MOUSE_REL) ||
                                (key==LETP||key==LETp))
                        {
                                //Programar entrenamiento
                                mouse_off(1);
                                boton(342,5,413,20,1);
                                if (salvar())
                                {
                                        boton(270,5,341,20,1);
                                        ver_archivos();
                                        boton(270,5,341,20,0);
                                }
                                w=0;
                                outp(888,0);
                                strcpy(nombre_temporal,"patrones.SSS");
                                nombre_temporal[0]=' ';
                                menu_entrenamiento();
                                opciones_menu_entrenamiento();
                                boton(342,5,413,20,0);
                                don=TRUE;
                        }
```

```
                else
                    if (mouse_in_box(414,5,485,20) && (key==LEFT_MOUSE_REL) ||
                            (key==LETE||key==LETe))
                    {
                        //Carga archivo de entrenamiento temporal
                        mouse_off(1);
                        boton(414,5,485,20,1);
                        menu_ejecutar();
                        carga_patrones_temporal();
                        boton(414,5,485,20,0);
                        menu_rojo();
                        don=TRUE;
                    }
                    else
                        if (mouse_in_box(486,5,557,20) &&
                                (key==LEFT_MOUSE_REL) || (key==LETY||key==LETy))
                        {                                       // Ayuda
                            boton_selec(486,5,557,20);
                            ayuda();
                            marco_carita();
                            carita_normal();
                            direccion();
                            que_direccion();
                            don=TRUE;
                        }
                        else
                            if (mouse_in_box(559,5,630,20) &&
                                    (key==LEFT_MOUSE_REL) || (key==LETS||key==LETs))
                            {
                                boton_selec(559,5,630,20);
                                if (salvar())
                                {
                                    boton(270,5,341,20,1);
                                    ver_archivos();
                                    boton(270,5,341,20,0);
                                }
                                don=TRUE;       // Salir
                                salir=TRUE;
                            }
    }while(don!=TRUE);
    return 0;
}
```

Rutina que despliega la ayuda del programa (AYUDA.CPP)

```
/*
    Funcion que despliega la ventana de ayuda del programa
*/
void ayuda();

void ayuda()
{
```

```
A_trabajo(110,40,600,500,LIGHTGRAY,SOLID_FILL,0);
fuente(9,1,0);
_textos("Ayuda",320,50,BLACK);
fuente(3,4,0);

_textos("Para seleccionar cualquier opcien del men£, s¢lo es necesario presionar la",130,85,BLACK);
_textos("tecla de la letra que est  en color rojo, ya sea  min£scula o may£scula, o",130,95,BLACK);
_textos("presionar con el boten izquierdo del raten el boten deseado.",130,105,BLACK);

dibuja_boton(115,128,180,145,2,"Archivos","A");
_textos("Abre una ventana con las siguientes opciones:",210,130,BLACK);

_textos("Abrir",150,150,BLACK);
_textos("Abre archivos con patrones ya almacenados.",210,150,BLACK);
_textos("Guardar",150,160,BLACK);
_textos("Almacena los patrones dibujados en un archivo nuevo.",210,160,BLACK);
_textos("Eliminar",150,170,BLACK);
_textos("Permite borrar archivos de patrones.",210,170,BLACK);
_textos("Cerrar",150,180,BLACK);
_textos("Cierra la ventana de archivos.",210,180,BLACK);

dibuja_boton(115,208,180,225,2,"Programar","P");
_textos("Va al modo de edicien de patrones para dibujarlos en la ret¡cula.",210,210,BLACK);

_textos("Aceptar",150,230,BLACK);
_textos("Guarda en memoria el patren dibujado.",210,230,BLACK);
_textos("Borrar",150,240,BLACK);
_textos("Borra de la ret¡cula el patren dibujado.",210,240,BLACK);

_textos("Para dibujar los patrones en la ret¡cula, es necesario el mouse, ya que",150,260,BLACK);
_textos("para activar un elemento de la matriz (1), hay que presionar el boten",150,270,BLACK);
_textos("izquierdo, mientras que para apagarlo (-1) hay que presionar el boten",150,280,BLACK);
_textos("derecho.",150,290,BLACK);

dibuja_boton(115,318,180,335,2,"Ejecutar","E");
_textos("Procesa el conjunto de patrones cargados en memoria, ya sea",210,320,BLACK);
_textos("de los que se acaban de dibujar, o de los que se cargaron y",210,330,BLACK);
_textos("ejecutaron anteriormente de un archivo.",210,340,BLACK);

dibuja_boton(115,368,180,385,2,"Ayuda"," y");
_textos("Muestra la presente pantalla de ayuda.",210,370,BLACK);

dibuja_boton(115,398,180,415,2,"Salir","S");
_textos("Sale del modo de edicien de patrones o tambi,n termina la",210,400,BLACK);
_textos("aplicacien.",210,410,BLACK);

dibuja_boton(115,438,180,455,2,"Esc","E");
_textos("Cancela una opcien seleccionada.",210,440,BLACK);
_textos("Presiona una tecla para continuar ...",380,460,BLACK);

getch();
V_trabajo(110,40,600,500,BLACK,SOLID_FILL);
}
```

Rutina que despliega la pantalla de presentación del programa ROBOT (DERECHOS.CPP)

```
/*
     Función que presenta la pantalla de presentación del programa
*/
void derechos(void);

void derechos(void)
{

  A_trabajo(175,130,500,350,LIGHTGRAY,SOLID_FILL,0);
  boton(180,135,400,345,1);
     fuente(9,4,0);
  _textos(" Robot",230,135,BLACK);
     fuente(3,4,0);
  carita_presenta();

  _textos("Control de un Móvil Mecánico",210,190,BLACK);
  _textos(" Mediante Redes Neuronales",210,202,BLACK);
  _textos(" Empleando el Algoritmo de",210,214,BLACK);
  _textos("Retropropagación del Error.",210,226,BLACK);

  _textos("   Proyecto realizado por",210,260,BLACK);
  _textos("Agustín de Jesús Astorga de Riquer",193,280,BLACK);
  _textos("   José Luis Cano García",210,290,BLACK);
  _textos("Departamento de Ingeniería de Control",185,315,BLACK);
  _textos("Facultad de Ingeniería. UNAM.",210,325,BLACK);
  _textos("Versión 1.00",420,320,BLACK);
  _textos("  1995-1996",420,330,BLACK);

     getch();
     V_trabajo(175,130,500,350,BLACK,SOLID_FILL);
}
```

Programa principal (ROBOT.CPP)

```
#include <dos.h>
#include <process.h>
#include <bios.h>
#include <fstream.h>
#include <graphics.h>
#include <math.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>
#include <string.h>
#include <alloc.h>
```

```
# include <dir.h>
# include <search.h>

#include "spider.h"

float U_patron_ent[SIZE_ENTRADA], U_patron_sal[SIZE_SALIDA];
int         entrada[ENT_BASE][ENT_ALTURA], salida[SAL_BASE][SAL_ALTURA];
char nombre[12];
int         salir;

#include "mouse.cpp"
#include "video.cpp"

#include "design.cpp"
#include "reticula.cpp"
#include "caritax.cpp"
#include "espacio.cpp"
#include "botones.cpp"
#include "cursor.cpp"

#include "address.cpp"
#include "derechos.cpp"
#include "ayuda.cpp"
#include "menus.cpp"
#include "salvaret.cpp"
#include "mvcarita.cpp"
#include "retropro.cpp"
#include "leeret.cpp"
#include "lista.cpp"
#include "opmenus.cpp"

void main()
{
      salir=FALSE;
      outp(888,0);
      lectura_pesos();
      inicializa_graficas();
      mouse_off(1);
      fuente(3,4,0);
      derechos();
      reticula();
      inicia_reticula();
      menu_principal();
      marco_carita();
      carita_normal();
      while (salir==FALSE)
            opciones_menu_principal();
      finaliza_graficas();
}
```

Apéndice C

En el presente apéndice se explicará el funcionamiento del programa de aplicación robot.exe, así como sus partes fundamentales.

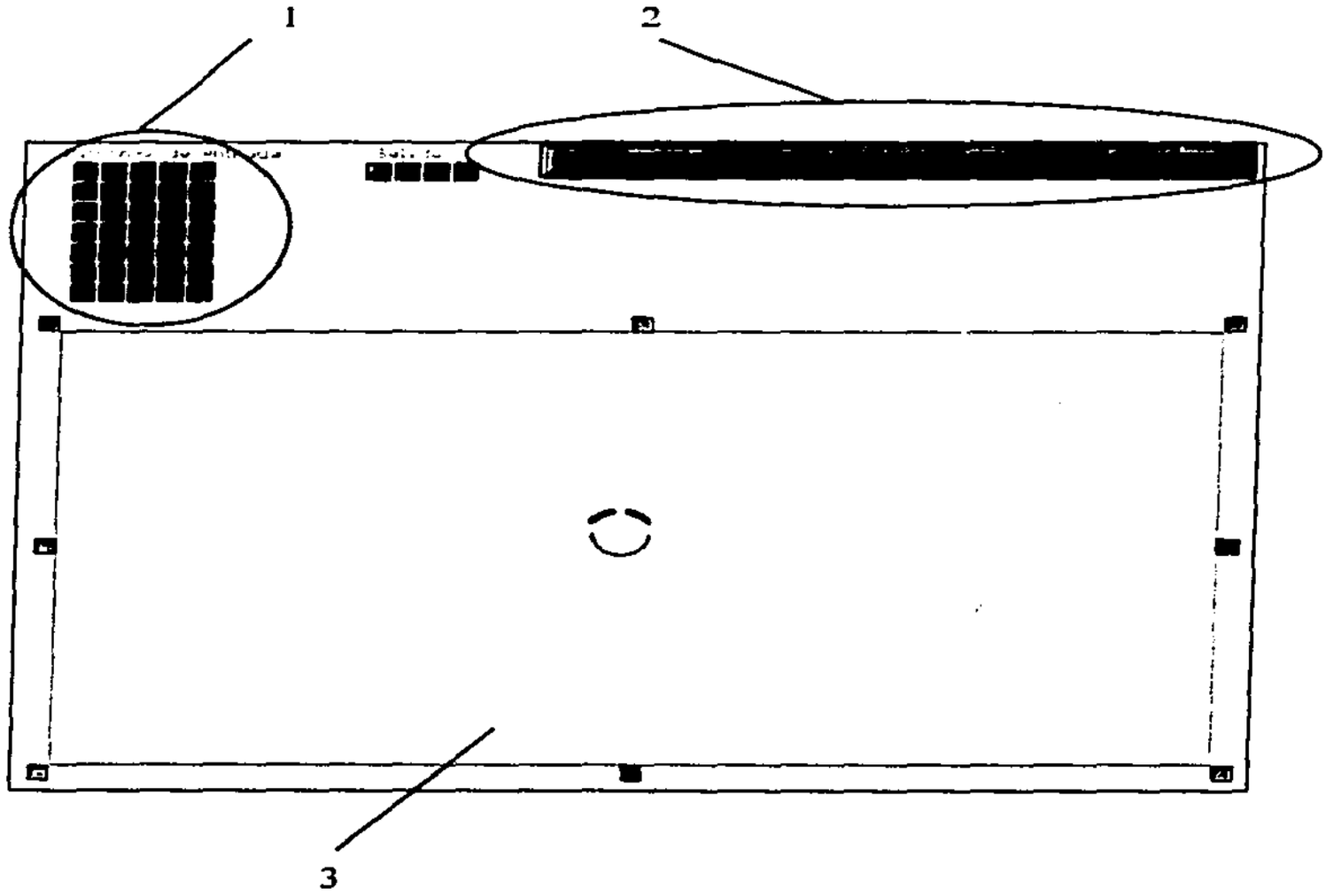
También se muestran dos ejemplos prácticos, uno con patrones ideales y el otro con patrones ruidosos para observar el funcionamiento de la red.

AC.1 Partes y mensajes del programa.

Pantalla de presentación cuando se ejecuta el programa robot.exe

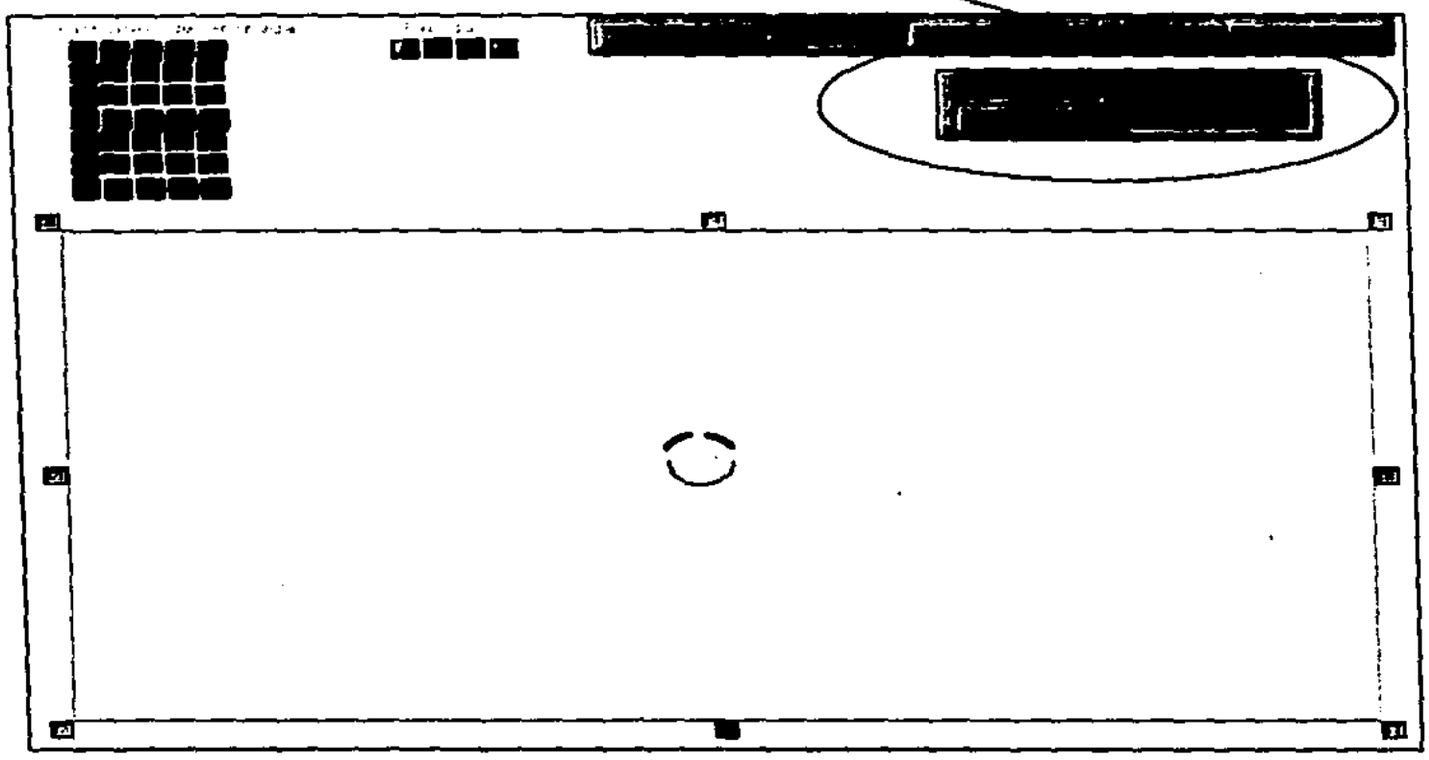
Pantalla principal del programa en la que se pueden distinguir las siguientes áreas:

1. Área de diseño.
2. Barra de herramientas.
3. Área de visualización.

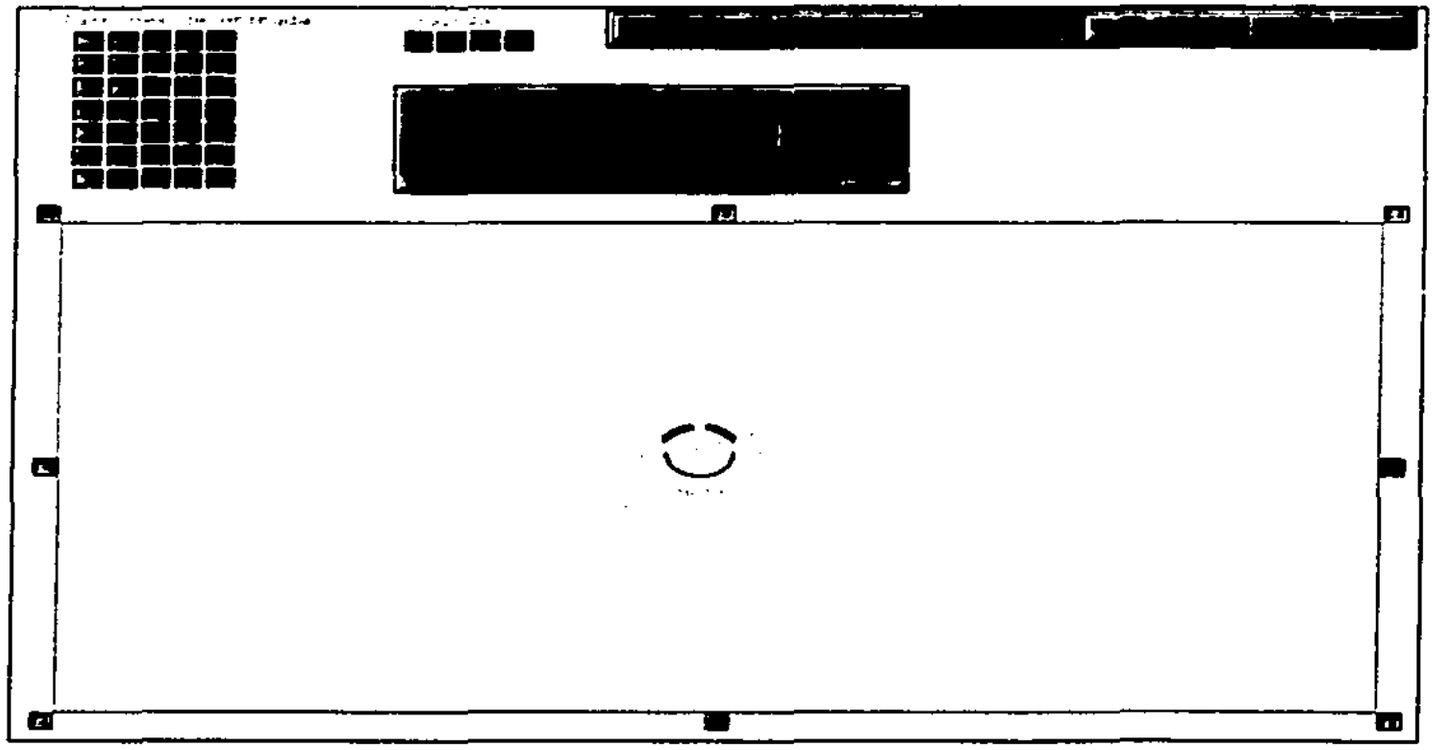


Pantalla con la muestra de un patrón capturado. Después de que se dibuja un patrón de entrada, se aprieta la opción de **Aceptar** para que guarde el patrón en memoria.

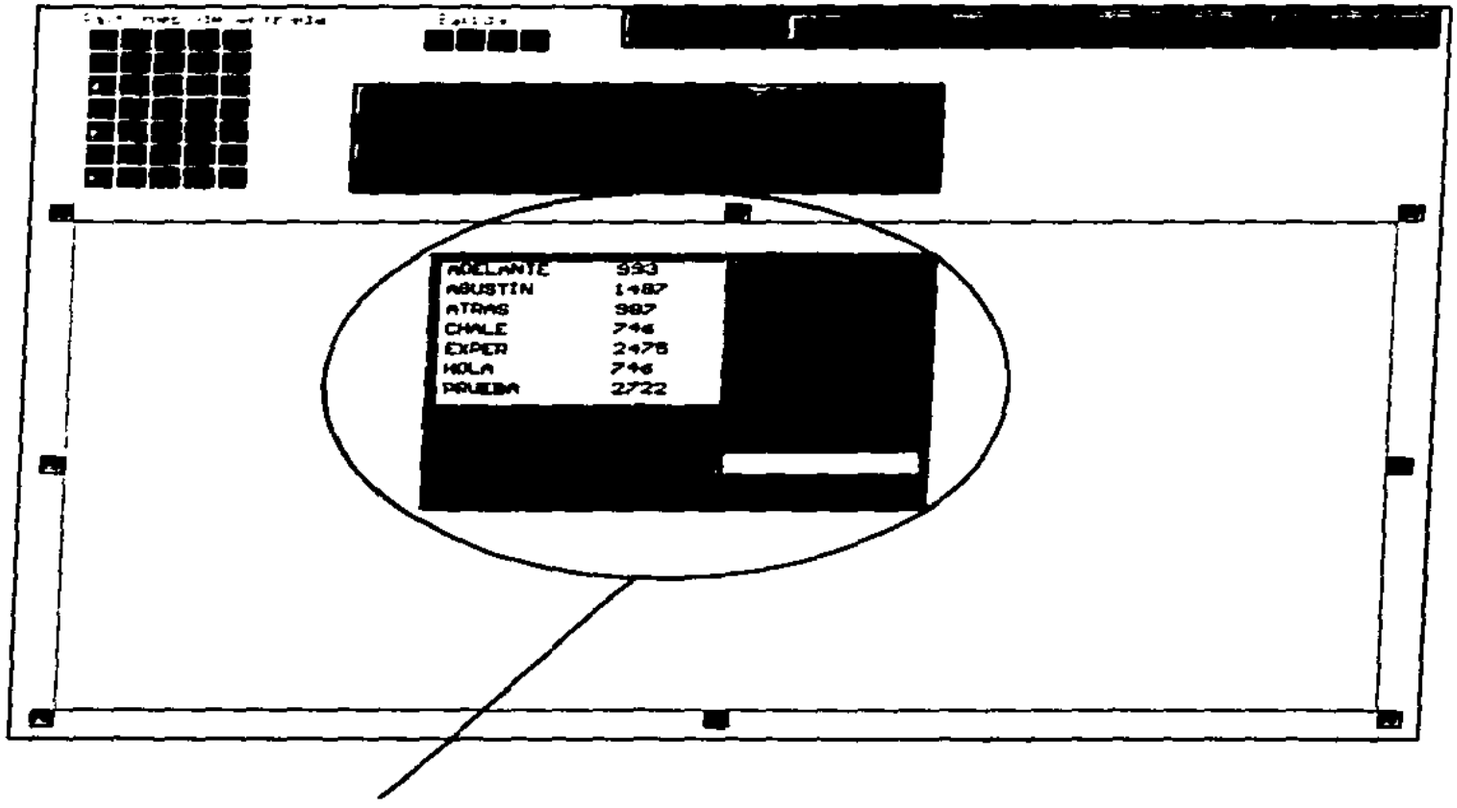
Ventana del menú **Programar**



Pantalla que muestra el mensaje que aparece cuando se quiere ejecutar un archivo de patrones y este no está cargado en memoria



Pantalla para guardar un archivo de patrones y marca error si no hay patrones que acaben de ser creados.



Cuadro de diálogo del menú **Archivo**

Al seleccionar **Ayuda** del menú, aparece la siguiente ventana:

Apéndice D

Apéndice D

Productos que actualmente usan la tecnología de las Redes Neuronales.[1]

Negocios

- Comerciales
    - Adaptive Decision Systems.
        - Aplicación: Evalúa directamente decisiones de mercado.
    - BehavHeuristics Inc.
        - Aplicación: Pronóstico de la demanda de vuelos aéreos.
        - Producto: Asistencia de mercadotecnia a aerolíneas.
    - Britvic
        - Aplicación: Predicción de ventas de bebidas.
    - Churchill Systems
        - Aplicación: Optimización de estrategias de mercado.
        - Producto: Sistema maestro de ventas.
        - Clientes: Veratex Corp.
    - Microsoft.
        - Aplicación: Ventas directas por correo.
    - NeuralWare.
        - Aplicación: Determina qué clientes deben de recibir un catálogo.
        - Clientes: Spiegal Inc.
    - HNC Software Inc.
        - Aplicación: Modelado predictivo de systemas para ventas directas.
        - Producto: DataBase Mining® Marksman.
    - HNC Software Inc.
        - Aplicación: Servidor Web para localidades en conflicto.
        - Producto: SelectCast
    - major US supermarket chain.
        - Aplicación: Encuentra la conexión entre las ventas de cerveza y pañales.

[1] Datos obtenidos de la página de Internet: http://www.emsl.pnl.gov:2080/docs/cie/neural/products

- Estado Real
    - Day & Zimmerman, Inc.
        - Aplicación: Estado real de valuaciones.
    - HNC Software.
        - Aplicación: Sistemas de valuación de bienes.
        - Producto: Automated Real Estate Appraisal System (AREAS).

Documentos y Procesamiento de formas

- Reconocimiento de caracteres impresos a máquina.
    - Audre Recognition Systems.
        - Aplicación: Reconocimiento óptico de caracteres.
        - Producto: Audre Neural Network.
    - Caere Corporation.
        - Aplicación: Reconocimiento óptico de caracteres.
        - Producto: OmniPage 6.0 and 7.0 Pro for Windows.
        - Producto: OmniPage 6.0 Pro for MacOS.
        - Producto: AnyFax OCR engine.
        - Producto: FaxMaster.
        - Producto: WinFax Pro 3.0 (from Delrina Technology Inc.).
    - Electronic Data Publishing, Inc.
        - Aplicación: Reconocimiento óptico de caracteres.
    - Synaptics.
        - Aplicación: Revisor de lectura.
        - Producto: VeriFone Oynx.
- Reconocimiento de caracteres impresos a mano.
    - Cardiff Software.
        - Aplicación: Reconocimiento de caracteres escritos a mano para faxes.
        - Producto: Teleform.
    - Eastman Kodak.
        - Aplicación: Procesador de formas.
        - Clientes: UK motor vehicles bureau.
    - HNC Software.
        - Aplicación: Reconocimiento de caracteres escritos a mano.
        - Producto: Quickstroke Automated Data Entry System.
        - Clientes: Avon, state of Wyoming.

- Poqet Computer/Fujitsu.
  - Aplicación: Computadoras de mano (palmtops).
- Synaptics.
  - Aplicación: Entrada a computadoras basadas en Windows.
  - Producto: HR1200.

- Reconocimiento de caracteres cursivos escritos a mano.
  - Apple Computer.
    - Aplicación: Entrada para asistentes personales de datos.
    - Producto: Apple Newton 120 (no todas las versiones).
  - Lexicus.
    - Aplicación: Sistema de escritura de letra cursiva a mano para PC's.
- Reconocimiento de Gráficos.
  - Fein-Marquart Associates, Inc.
    - Aplicación: Translación de dibujos químicos a tablas de conexión.

Industria Alimenticia

- Análisis de Fragancias y olores.
  - AlphaMOS.
    - Aplicación: Análisis de olores vía una naríz electrónica.
  - AromaScan Inc.
    - Aplicación: Análisis de olores vía una naríz electrónica.
    - Producto: AromaScan electronic nose
    - Clientes: Coca Cola, Food and Drug Administration, otros.
  - Neotronics Scientific.
    - Aplicación: Detección y reconocimiento de olores y fragancias.
    - Producto: e-NOSE 4000 electronic nose.
  - Sharp Corporation.
    - Aplicación: Control de cocina vía una naríz electrónica en hornos de microondas.
- Productos en desarrollo.
  - M&M/Mars.
    - Aplicación: Búsqueda para fórmulas químicas improvisadas.

- Control de Calidad.
    - Anheuser-Busch.
        - Aplicación: Prueba de la cerveza.
    - Florida Department of Citrus.
        - Aplicación: Pruebas de pureza y detección de limpieza de la pulpa.
    - Frito-Lay.
        - Aplicación: Control de calidad para papas fritas.

Aplicaciones Financieras

- Transacciones de mercado.
    - Carl & Associates.
        - Aplicación: Pronóstico de existencias.
    - Citibank.
        - Aplicación: Análisis de tendencias de mercados internacionales.
    - Daiwa Securities Co., Ltd. And NEC Corporation.
        - Aplicación: Pronóstico de precios de existencias.
    - Enonostat Ltd.
        - Aplicación: Administración de cartera de fianzas.
        - Producto: Global Bond.
    - Gerber Baby Foods.
        - Aplicación: Manejo de ventas futuras de ganado.
    - G.R. Pugh & Company.
        - Aplicación: Corporativo de clasificación de bonos.
    - John Deere & Company.
        - Aplicación: Administración de fondos de pensión.
    - LBS Capital Management Inc.
        - Aplicación: Administración de fondos mutuos.
    - Mellon Equility Associates.
        - Aplicación: Administración de carteras.
    - Neural Applications Corporation.
        - Aplicación: Pronóstico de existencias.
    - O'Sullivan Brothers Investments, Ltd.
        - Aplicación: Control monetario.
    - Walkrich Investment Advisors.
        - Aplicación: Pronóstrico de existencias.

- Dunn and Bradstreet.
  - Aplicación: Aprobación de cheques.
- HNC Software.
  - Aplicación: Detección de fraudes con tarjeta de crédito.
  - Producto: Falcon
  - Clientes: First Bank USA, Household Credit Services and National Colonial Bank USA.
- MasterCard.
  - Aplicación: Identificación de desviaciones en costumbres de consumo.
- Nestor Inc. and Fraud Detection Systems.
  - Aplicación: Detección de fraudes con tarjeta de crédito.
  - Clientes: Bank of America, Canadian Imperial Bank of Commerce and Europay International S.A. Belgium.
- NeuroMetric Vision System Inc.
  - Aplicación: Verificación de firmas en cheques.
  - Producto: Check Signature Verification System.
  - Clientes: US banking industry and Federal Reserve System.
- Visa.
  - Aplicación: Identificación de desviaciones en costumbres de consumo.

- Clasificación de créditos.
  - Adaptive Decision Systems.
    - Aplicación: Control de crédito.
  - Chase Financial Technologies.
    - Aplicación: Monto de crédito.
    - Producto: Creditview.
  - HNC Software.
    - Aplicación: Control de hipotecas y administración de riesgos.
    - Producto: Colleague.
    - Producto: Aquarius.

Energía

- Nivel de carga eléctrica.
  - Bayernwerk AG
    - Aplicación: Nivel de carga eléctrica.
  - Britvic.
    - Aplicación: Nivel de carga.
  - Electric Power Research Institute.
    - Aplicación: Nivel de carga.
  - Energie Versorgung Schwaben.
    - Aplicación: Nivel de carga.
  - Pacific Gas & Electric.
    - Aplicación: Nivel de carga a corto y largo plazo.
  - PUB Singapore.
    - Aplicación: Nivel de carga.
  - Vattenfall.
    - Aplicación: Estimación de carga.
- Operación de presas hidroeléctricas.
  - Tauernkraftwerke.
    - Aplicación: Predicción de desplazamiento de presas.
- Gas Natural.
  - Northern Natural Gas.
    - Aplicación: Predicción de índices de precio del gas.

Manufactura

- Procesos de Control
  - Fujitsu.
    - Aplicación: Controlador para moldeado contínuo.
  - Honeywell.
    - Aplicación: Systemas de procesos de control.
  - Neural Applications Corporation.
    - Aplicación: Controlador para la manufactura del acero.
    - Producto: Intelligen Arc Furnace.
  - Nippon Steel.
    - Aplicación: Controlador para moldeado contínuo.
  - Pavilion Technologies.
    - Aplicación: Procesos de control, reducción de gastos.

        - Producto: Process Insights®
        - Clientes: Eastman Kodak.
    - Siemens.
        - Aplicación: Proceso de control de molinos rotativos.
    - Texaco's Puget Sound Refinery.
        - Aplicación: Sistema de procesos de control en refinerías.
- Systemas de Control de Calidad.
    - Anheuser-Busch.
        - Aplicación: Prueba de la cerveza.
    - Applied Intelligent Systems.
        - Aplicación: Sistemas de reconocimiento de visión.
    - CTS Electronics.
        - Aplicación: Clasificación de defectos de altavoces.
    - Dunlop.
        - Aplicación: Prueba de llantas.
    - Intel.
        - Aplicación: Control de calidad en la manufactura de chips para computadora.
    - Monsanto.
        - Aplicación: Predicción de calidad en plásticos.
    - Netrologic, Inc.
        - Aplicación: Control de calidad en soldadura.
    - NLK-Celpap.
        - Aplicación: Predicción de calidad en el papel.
    - Volvo.
        - Aplicación: Inspección de pintura.

Medicina y cuidado de la salud

- Análisis de imágenes.
    - NeuroMedical Systems Inc.
        - Aplicación: Diagnóstico del SIDA.
        - Producto: PapNet
- Desarrollo de drogas.
    - Vysis Inc.
        - Aplicación: Análisis de proteínas para el desarrollo de drogas.

Ciencias e Ingeniería

- Ingeniería Química.
  - AI Ware Inc.
    - Aplicación: Optimización de ingredientes y características de procesamiento.
    - Producto: CAD/Chem Custom Formulation System.
  - Chemometric Solutions.
    - Aplicación: Análisis de datos químicos.
  - StellarNet Inc.
    - Aplicación: Espectroscopía.
- Ingeniería Eléctrica.
  - NeuroCad Inc.
    - Aplicación: Optimización de ruteo en circuitos.
    - Producto: NeuroRoute.
- Agua.
  - National Weather Service.
    - Aplicación: Niveles cuantitavos de agua.

Transporte y comunicación

- Transporte.
  - London Underground.
    - Aplicación: Detección de fallas.
  - Rolls Royce.
    - Aplicación: Detección de fallas.
- Comunicaciones.
  - AT&T/Lucent Technologies.
    - Aplicación: Sistemas de cancelación de ecos.

Productos en Hardware.[2]

- Chips y/o sistemas comercialmente disponibles.
  - Accurate Automation MIMD Processor.
    - http://www.accurate-automation.com
  - Adaptive Solutions CNAPS.
    - 1400 N.W. Compton Drive, Suite 340, Beaverton, OR 97006, USA.
  - HNC 100 NAP Chip and SNAP Neurocomputer.
    - Oberlin Dr., San Diego Ca. 92121-1718.
  - IBM ZISC036.
    - e-mail: info@neuroptics.com
  - IC Tech Feedback Neural Net Chips.
    - http://www.webcom.com/~ictech/ictech.html.
  - Intel ETANN 80170NX (Electrically Trainable Analog Neural Network).
  - Micro Devices MD-1220.
    - 5695B Beggs Rd., Orlando, FL. 32810-2603, USA.
  - MCE MT19003.
    - Alexandra Way, Ashchurch Business Center, Tewkesbury, Gloucestershire GL20 8TB.
  - National Semiconductor NeuFuz/Cop8 Microcontrollers.
    - 2900 Semiconductor Dr., Santa Clara, Ca. 95952-8090. USA.
  - Nestor NI1000.
    - One Richmond Square, Providence, RI 02906, USA.
  - NeuraLogix (Ahora Adaptive Logic) NLX110, NLX-230, NLX420.
    - http://www.adaptivelogic.com
  - Oxford Computer Chips and Modules.
    - 39 Old Good Hill Rd., Oxford, Conn. 06478.
  - Philips L-Neuro chips.
    - Avenue Descartes, BP 15, 94453 Limeil-Brevannes Cedex, France.
  - Sensory Circuits RSC-164 Chip de reconocimiento del habla.
    - http://www.sensoryc.com

[2] Datos obtenidos de la pagina de Internet: http: www1.cern.ch NeuralNets/nnwInHepHard.html

- Siemens MA-16 chip, Synapse-1, Neurocomputer.
  - D-81730 Munich, Ge., e-mail: juergen.pampus@mch.sni.de
- Synaptics I10XX Chip de reconocimiento de objetos.
  - 2860 Zanker Rd., Suite 206, San Jose Ca. 95134.
- UCLi Ltd Pram-256 Chip.
  - http://crg.eee.kcl.ac.uk/clarkson/pram.html

- Aceleradores y otras tarjetas.
  - Accurate Automation VME Card.
  - Adaptive Solutions CNAPS/PC.
  - BrainMaker Accelerators.
    - 10024 Newtown Rd., Nevada City, Ca. 95959-9794.
  - Adaptive Solutions CNAPS ISA, PCI, and VME Cards.
  - Current Technology MM32k.
    - e-mail: mag@curtech.mv.com
  - HNC Balboa System.
    - 5501 Oberlin Dr., San Diego Ca. 92121-1718.
  - IBM ZISC/ISA Accelerator for PC.
    - Microelectronics Division, Essonnes Development Lab., IBM France, 225 Boulevard Jon Kennedy, F 91105 Corbeil-Essonnes, France.
  - IBM ZISC PCMCIA Card.
    - Neuroptics Technologies, USA, 131A Stony Circle, Suite 500, Santa Rosa, California.
  - Mosaic QED.
    - http://www.mosaic-industries.com
  - Neural Network Systems -NT5000 & NT6000.
    - Neural Technologies, Ltd., Peterfield, UK.
  - NeuroDynamX Neural-Accelerators.
    - NeuroDynamX, P.O. Box 323, Boulder, Co.80306.
  - Nestor ISA, PCI and VME Cards.
    - One Richmond Square, Providence, RI 02906, USA.
  - Rapid Imaging VME and ISA Cards.
    - 5955 T. G. Lee Blvd., Suite 150, Orlando, Fla. 32822.
  - Sundance SMT306 Neural Procesing'C40 TIM.
    - http://www.sundance.com/index.html
  - Telebyte Model 1000 NeuroEngine.
    - http://www.telebyteusa.com

    - Vision Harvest NeuroSimulator.
        - HCR Box 36, Hatch, N.M. 87937.
    - Ward Systems NeuralBoard.

- Chips y/o Sistemas no comerciales o prototipos.
    - Bellcore CLNN learning chip.
    - Bellcore Cascadable VME Card with 3 ETANN's
    - Hughes M1718 chip.
    - Motora.
    - MESA Institute NeuroClassifier.
        - e-mail: masa@ice.el.utwente.nl
    - Ricoh RN-200 chip.
    - Univ. Rome Circuits Lab's Cellular Neural Network chips.
        - http://www.eln.utovrm.it/~sargeni/circuiti.html
    - Univ. Trento's TOTEM chip (ESPRIT project).
        - e-mail: sartori@irst.it

- Hardware.
    - Bologna Univ. ETANN and MA-16 VME Cards.
    - CDF/Michigan ETANN Fastbus Cards.
    - CP-LEAR/Basel ECL Cards and DSP NNW's.
    - H. Haggerty discrete component network.
    - RIT ETTAN VME Cards.
    - RIT ZISC PC/ISA and VME Cards.